

---

# **Qubes OS**

***Release 0.1***

**Qubes OS Project**

**Jan 08, 2024**



# INTRODUCTION

1	Table of contents
---	-------------------

1
---





## TABLE OF CONTENTS

### 1.1 Introduction

#### 1.1.1 What is Qubes OS?

Qubes OS is a free and open-source, security-oriented operating system for single-user desktop computing. Qubes OS leverages *Xen-based virtualization* to allow for the creation and management of isolated compartments called *qubes*.

These qubes, which are implemented as *virtual machines VMs*, have specific:

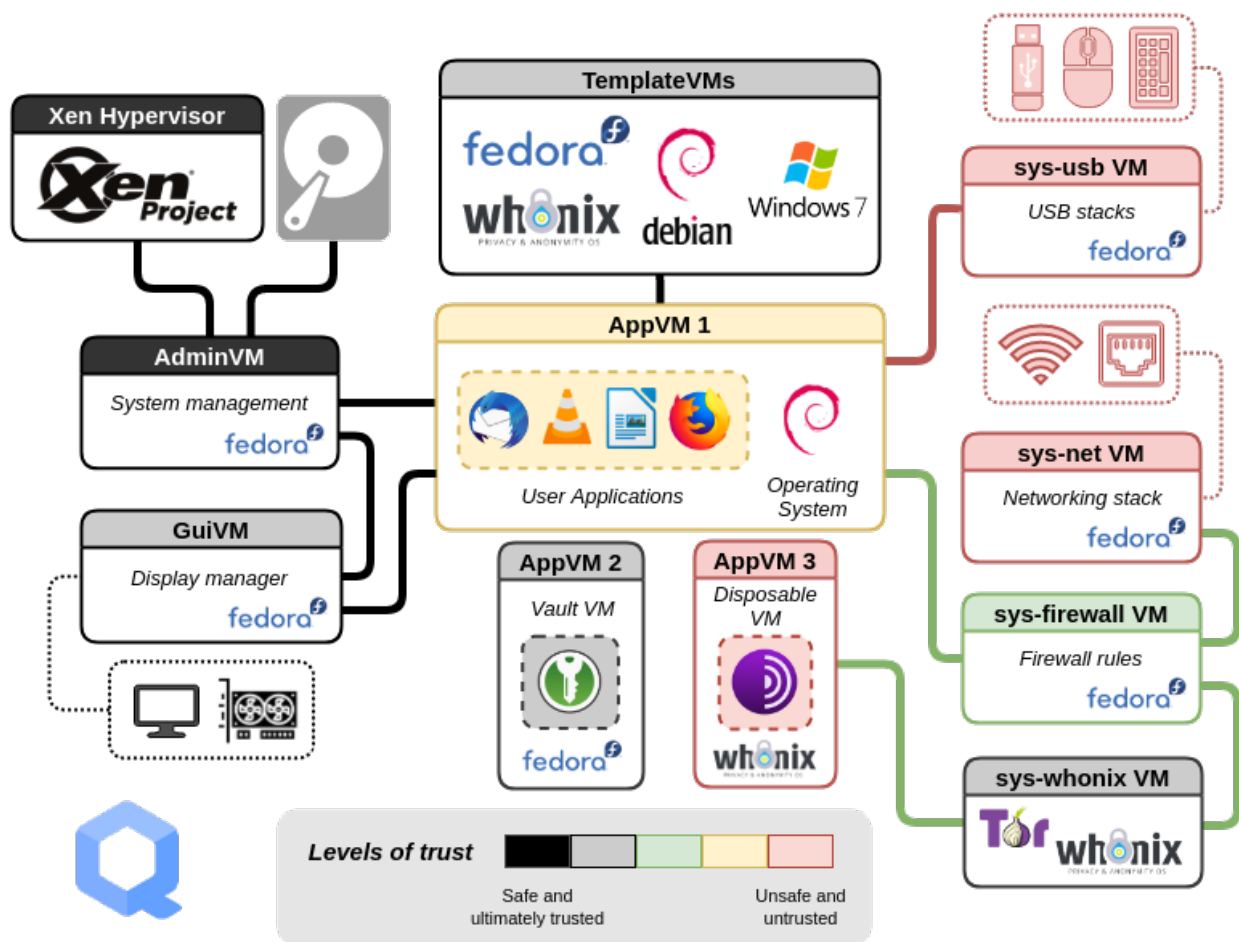
- **Purposes:** with a predefined set of one or many isolated applications, for personal or professional projects, to manage the *network stack*, *the firewall*, or to fulfill other user-defined purposes.
- **Natures:** *full-fledged* or *stripped-down* virtual machines based on popular operating systems, such as *Fedora*, *Debian*, and *Windows*.
- **Levels of trust:** from complete to non-existent. All windows are displayed in a unified desktop environment with *unforgeable colored window borders* so that different security levels are easily identifiable.

Note: See our glossary and FAQ for more information.

#### 1.1.2 Features

- **Strong isolation** Isolate different pieces of software as if they were installed on separate physical machines using advanced virtualization techniques.
- **Template system** Use *app qubes* to share a root file system without sacrificing security using the innovative *Template system*.
- **Multiple operating systems** Use multiple operating systems at the same time, including *Fedora*, *Debian*, and *Windows*
- **Disposables** Create *disposables* on the fly that self-destruct when shut down.
- **Whonix integration** Run *Tor* securely system-wide using *Whonix with Qubes*.
- **Device isolation** Secure *device handling* through isolation of network cards and USB controllers.
- **Split GPG** Utilize *Split GPG* to keep your private keys safe.
- **U2F proxy** Operate *Qubes U2F proxy* to use your two-factor authentication devices without exposing your web browser to the full USB stack.
- **Open-source** Users are free to use, copy, and modify Qubes OS and *are encouraged to do so!*

Note: Given the technical nature of Qubes OS, prior experience with Linux can be helpful.



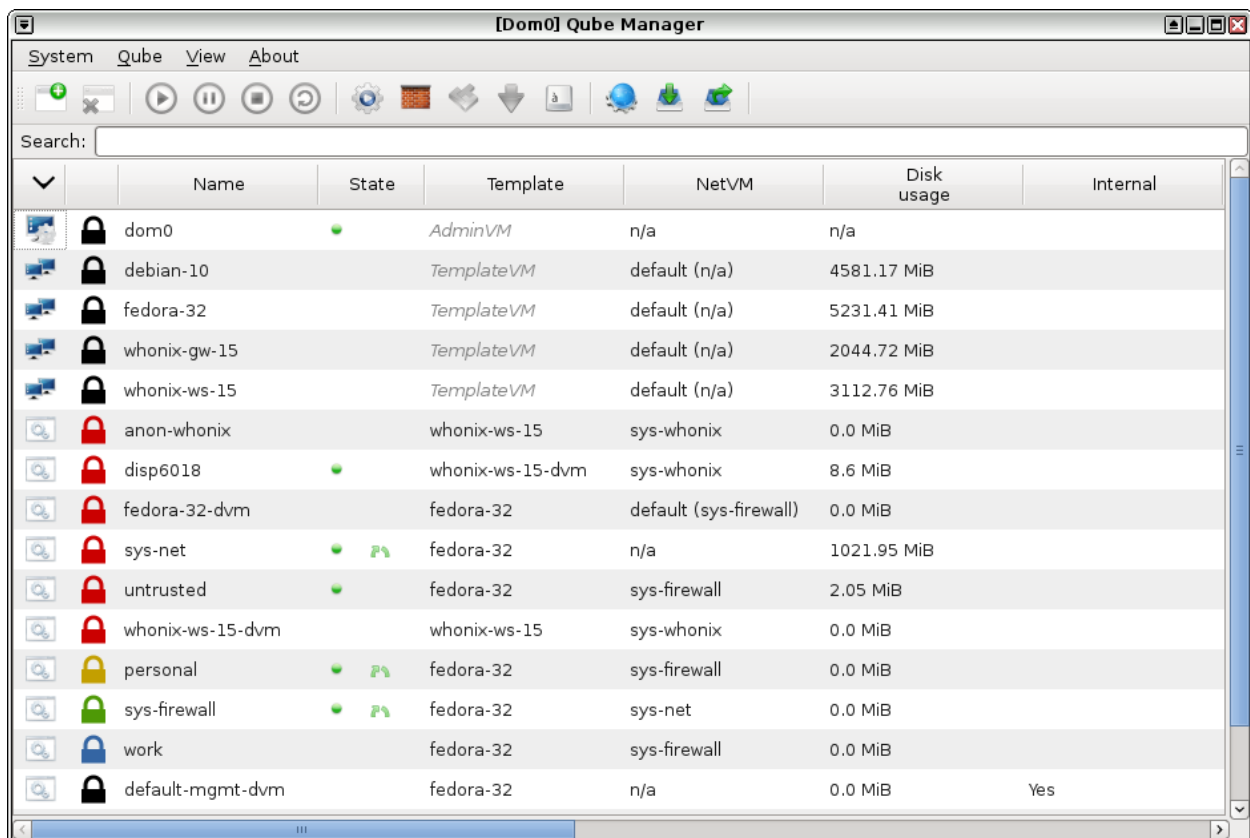
### 1.1.3 Why Qubes OS?

#### Physical isolation is a given safeguard that the digital world lacks

Throughout our lives, we engage in various activities, such as going to school, working, voting, taking care of our families, and visiting with friends. These activities are spatially and temporally bound: They happen in isolation from one another, in their own compartments, which often represent an essential safeguard, as in the case of voting.

In our digital lives, the situation is quite different: All of our activities typically happen on a single device. This causes us to worry about whether it's safe to click on a link or install an app, since being hacked imperils our entire digital existence.

Qubes eliminates this concern by allowing us to divide a device into many compartments, much as we divide a physical building into many rooms. Better yet, it allows us to create new compartments whenever we need them, and it gives us sophisticated tools for securely managing our activities and data across these compartments.



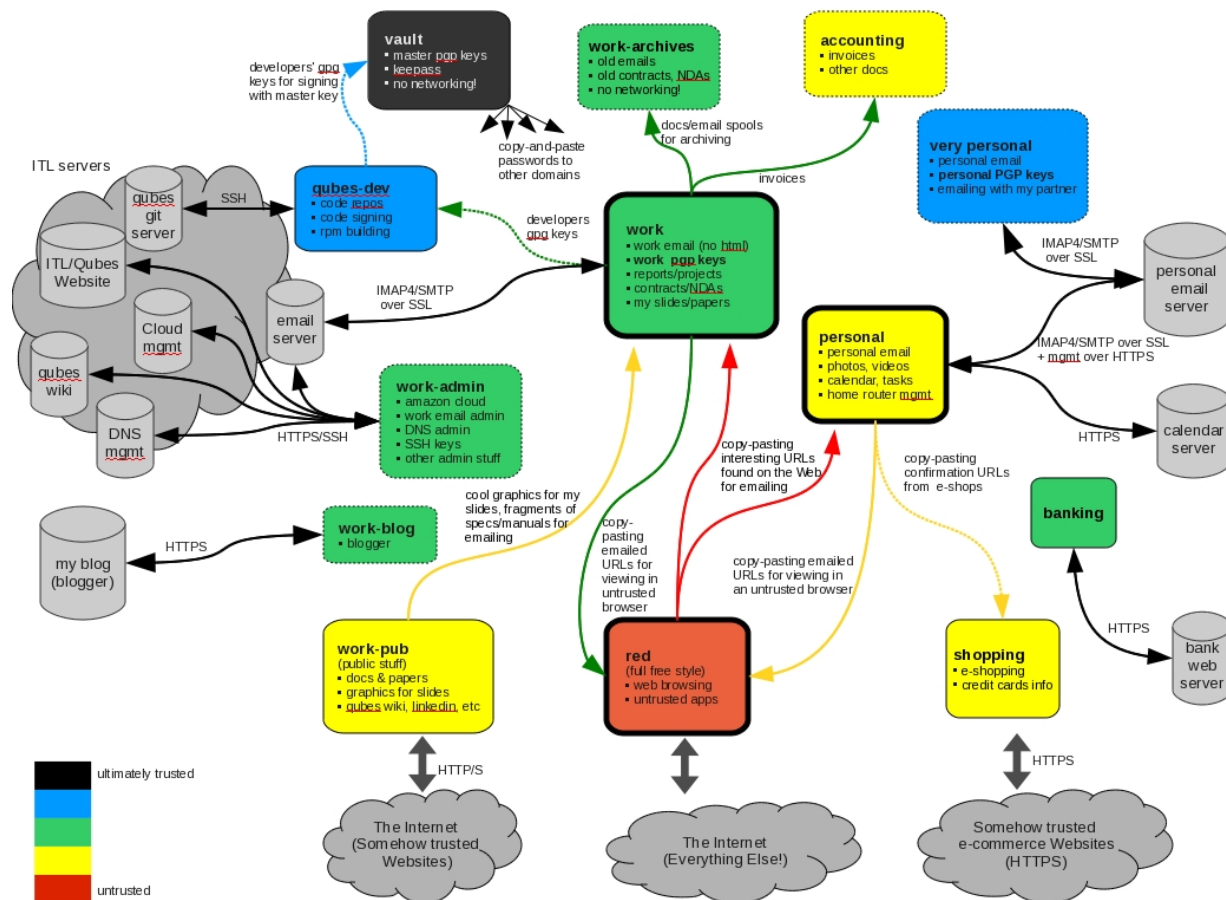
The screenshot shows the [Dom0] Qube Manager interface. It features a menu bar with 'System', 'Qube', 'View', and 'About'. Below the menu is a toolbar with various icons for managing VMs. A search bar is located above the table. The table lists VMs with columns for Name, State, Template, NetVM, Disk usage, and Internal. The VMs are listed in descending order of disk usage.

	Name	State	Template	NetVM	Disk usage	Internal
	dom0		AdminVM	n/a	n/a	
	debian-10		TemplateVM	default (n/a)	4581.17 MiB	
	fedora-32		TemplateVM	default (n/a)	5231.41 MiB	
	whonix-gw-15		TemplateVM	default (n/a)	2044.72 MiB	
	whonix-ws-15		TemplateVM	default (n/a)	3112.76 MiB	
	anon-whonix		whonix-ws-15	sys-whonix	0.0 MiB	
	disp6018		whonix-ws-15-dvm	sys-whonix	8.6 MiB	
	fedora-32-dvm		fedora-32	default (sys-firewall)	0.0 MiB	
	sys-net		fedora-32	n/a	1021.95 MiB	
	untrusted		fedora-32	sys-firewall	2.05 MiB	
	whonix-ws-15-dvm		whonix-ws-15	sys-whonix	0.0 MiB	
	personal		fedora-32	sys-firewall	0.0 MiB	
	sys-firewall		fedora-32	sys-net	0.0 MiB	
	work		fedora-32	sys-firewall	0.0 MiB	
	default-mgmt-dvm		fedora-32	n/a	0.0 MiB	Yes

### 1.1.4 Qubes allows you to compartmentalize your digital life

Many of us are initially surprised to learn that our devices do not support the kind of secure compartmentalization that our lives demand, and we're disappointed that software vendors rely on generic defenses that repeatedly succumb to new attacks.

In building Qubes, our working assumption is that all software contains bugs. Not only that, but in their stampeding rush to meet deadlines, the world's stressed-out software developers are pumping out new code at a staggering rate - far faster than the comparatively smaller population of security experts could ever hope to analyze it for vulnerabilities, much less fix everything. Rather than pretend that we can prevent these inevitable vulnerabilities from being exploited,



we've designed Qubes under the assumption that they **will** be exploited. It's only a matter of time until the next zero-day attack.

In light of this sobering reality, Qubes takes an eminently practical approach: confine, control, and contain the damage. It allows you to keep valuable data separate from risky activities, preventing cross-contamination. This means you can do everything on the same physical computer without having to worry about a single successful cyberattack taking down your entire digital life in one fell swoop. In fact, Qubes has **distinct advantages over physical air gaps**.

## 1.1.5 Made to support vulnerable users and power users alike

Qubes provides practical, usable security to vulnerable and actively-targeted individuals, such as journalists, activists, whistleblowers, and researchers. Qubes is designed with the understanding that people make mistakes, and it allows you to protect yourself from your own mistakes. It's a place where you can click on links, open attachments, plug in devices, and install software free from worry. It's a place where **you** have control over your software, not the other way around. (See some *examples of how different types of users organize their qubes*.)

Qubes is also powerful. Organizations like the **Freedom of the Press Foundation**, **Mullvad**, and **Let's Encrypt** rely on Qubes as they build and maintain critical privacy and security internet technologies that are in turn relied upon by countless users around the world every day. Renowned security **experts** like Edward Snowden, Daniel J. Bernstein, Micah Lee, Christopher Soghoian, Isis Agora Lovecruft, Peter Todd, Bill Budington, and Kenn White use and recommend Qubes.

Qubes is one of the few operating systems that places the security of its users above all else. It is, and always will be, free and open-source software, because the fundamental operating system that constitutes the core infrastructure of our digital lives **must** be free and open-source in order to be trustworthy.

The screenshot shows the Qubes OS desktop environment. On the left, a document editor displays a report titled "Report on using Qubes OS for security critical operations". The report discusses Qubes as a security-focused operating system and mentions its compatibility with ThinkPads. On the right, a web browser displays the Qubes OS introduction page, which includes a diagram of the system architecture, a list of features, and a section on why Qubes OS is used.

**Report on using Qubes OS for security critical operations**

Qubes is a security-focused operating system. It aims to provide this security through compartmentalization/isolation. Allowing one application and others that need to access or interact with security critical servers and software to do so using the same computer they do all their other work on. Organizations like the Freedom of the Press Foundation and Let's Encrypt are already relying on Qubes.

Most, if not all, of our current ThinkPads are compatible with Qubes (sometimes with minor issues). Therefore the biggest problem is teaching people to use it. Qubes has a lot of differences to standard OS's, and so such previous experience or teaching people is required.

Loem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

**Qubes OS - Wikipedia**

Introduction | Qubes OS | Hardware Compatibility | Thinkpad e5 at DuckDuckGo

Qubes OS is a security-focused operating system. It aims to provide this security through compartmentalization/isolation. Allowing one application and others that need to access or interact with security critical servers and software to do so using the same computer they do all their other work on. Organizations like the Freedom of the Press Foundation and Let's Encrypt are already relying on Qubes.

Most, if not all, of our current ThinkPads are compatible with Qubes (sometimes with minor issues). Therefore the biggest problem is teaching people to use it. Qubes has a lot of differences to standard OS's, and so such previous experience or teaching people is required.

Loem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

**Features**

<b>Strong isolation</b> Isolate different pieces of software as if they were installed on separate physical machines using PV or HVM virtualization techniques.	<b>Template system</b> Use AppVMs to share a root file system without sacrificing security using the innovative Template system.	<b>Multiple operating systems</b> Use multiple operating systems at the same time, including Fedora, Debian, and Windows.
<b>DisposableVMs</b> Create DisposableVMs on the fly that self-destruct when shut down.	<b>Whonix integration</b> Run Tor security system-wide using Whonix with Qubes.	<b>Device isolation</b> Secure device handling through isolation of network cards and USB controllers.
<b>Split GPG</b> Utilize Split GPG to keep your private keys safe.	<b>U2F proxy</b> Operate Qubes U2F proxy to use your two-factor authentication devices without exposing your web browser to the full USB stack.	<b>Open-source</b> Users are free to use, copy, and modify Qubes OS and are encouraged to do so!

**Why Qubes OS?**

Physical isolation is a given safeguard that the digital world lacks

## Video Tours

Want to see Qubes OS in action? Sit back and watch a guided *tour*!

## Screenshots

See what using Qubes actually looks like with these *screenshots* of various applications running in Qubes.

## Getting Started

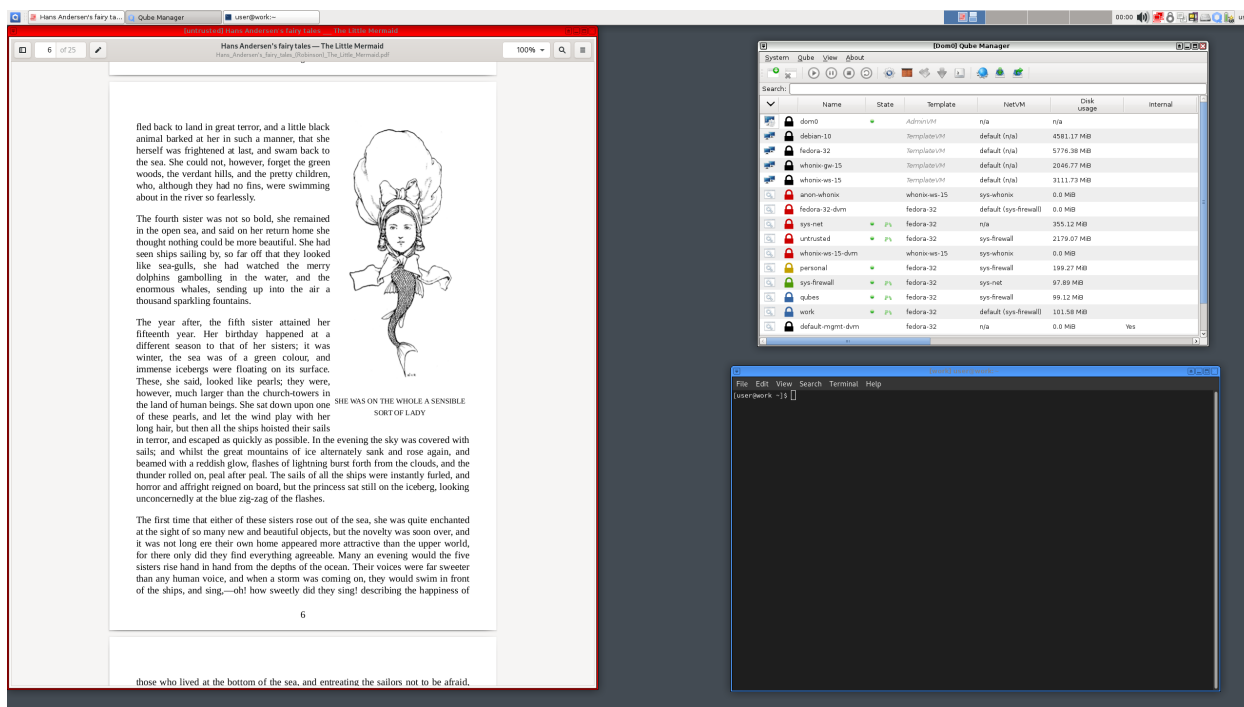
Ready to get started with Qubes? *Here's* what you need to know after installing.

### 1.1.6 More information

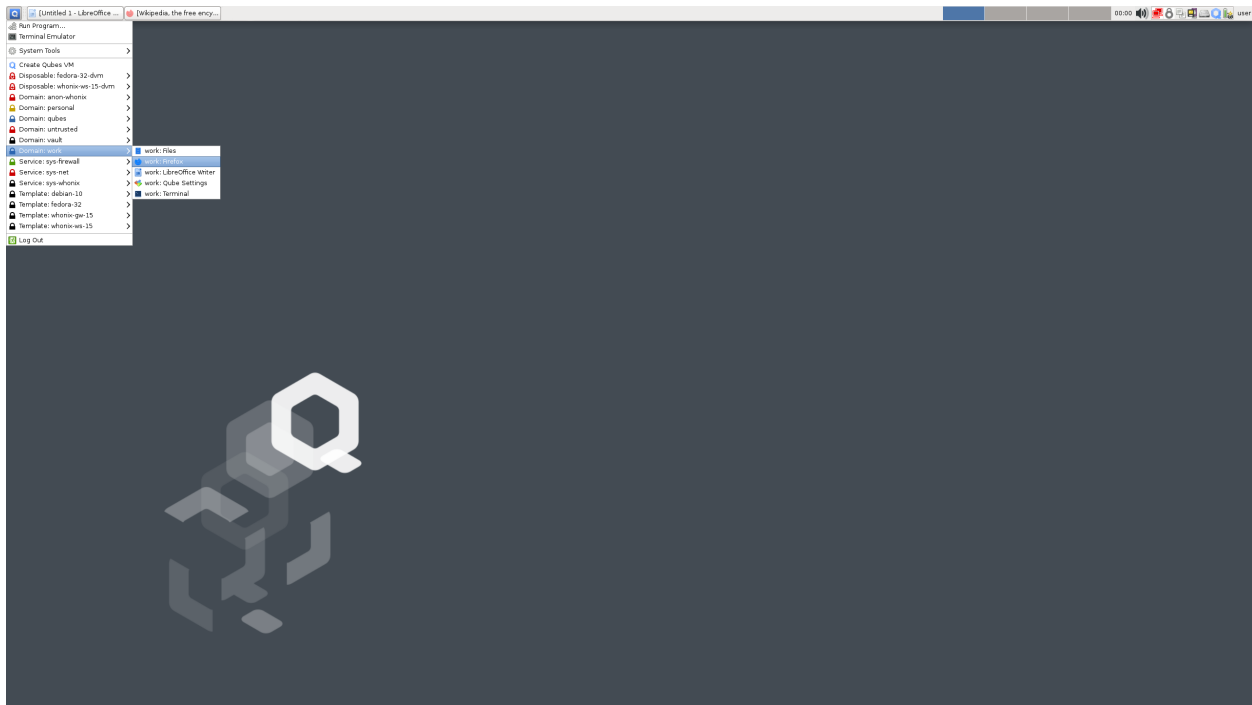
This page is just a brief introduction to what Qubes is all about, and many technical details have been omitted here for the sake of presentation.

- If you're a current or potential Qubes user, you may want to check out the *documentation* and the *user FAQ*.
- If you're a developer, there's dedicated *developer documentation* and a *developer FAQ* just for you.
- Ready to give Qubes a try? Head on over to the *downloads* page, and read the *installation guide*.
- Need help, or just want to join the conversation? Learn more about *help, support, the mailing lists, and the forum*.

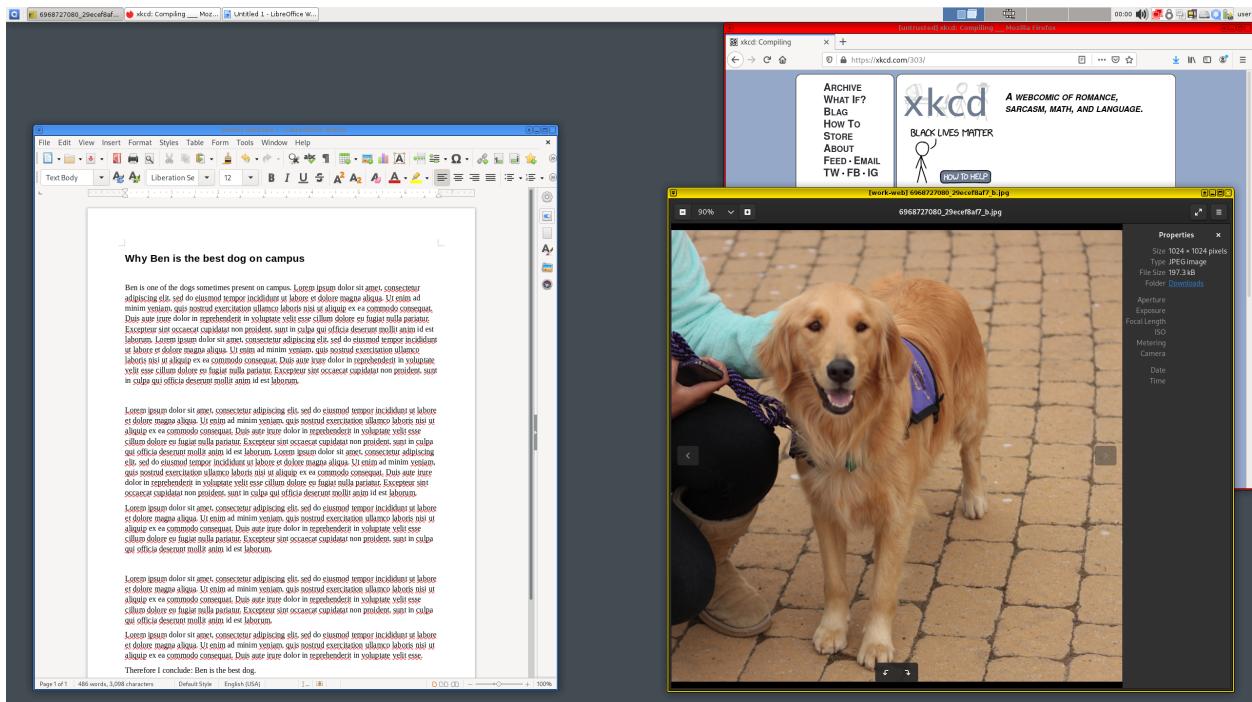
## 1.2 Screenshots



The default desktop environment is Xfce4.



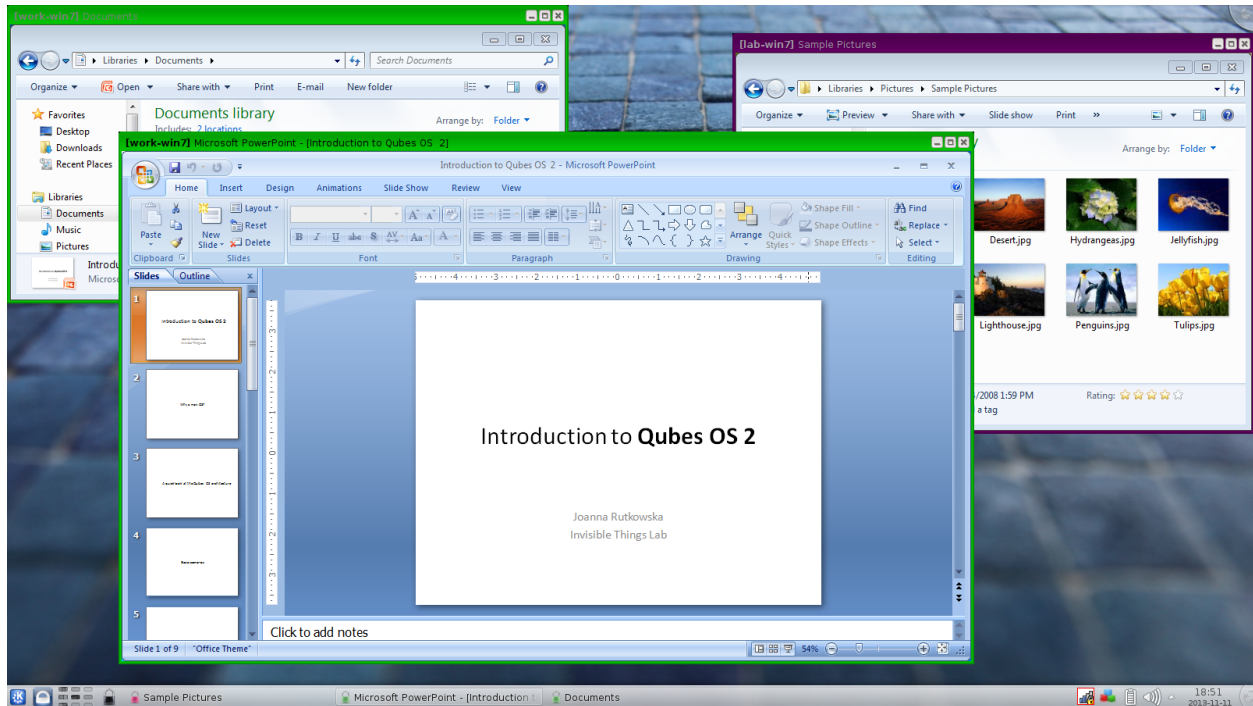
Starting applications from different domains (AppVMs) is very easy.



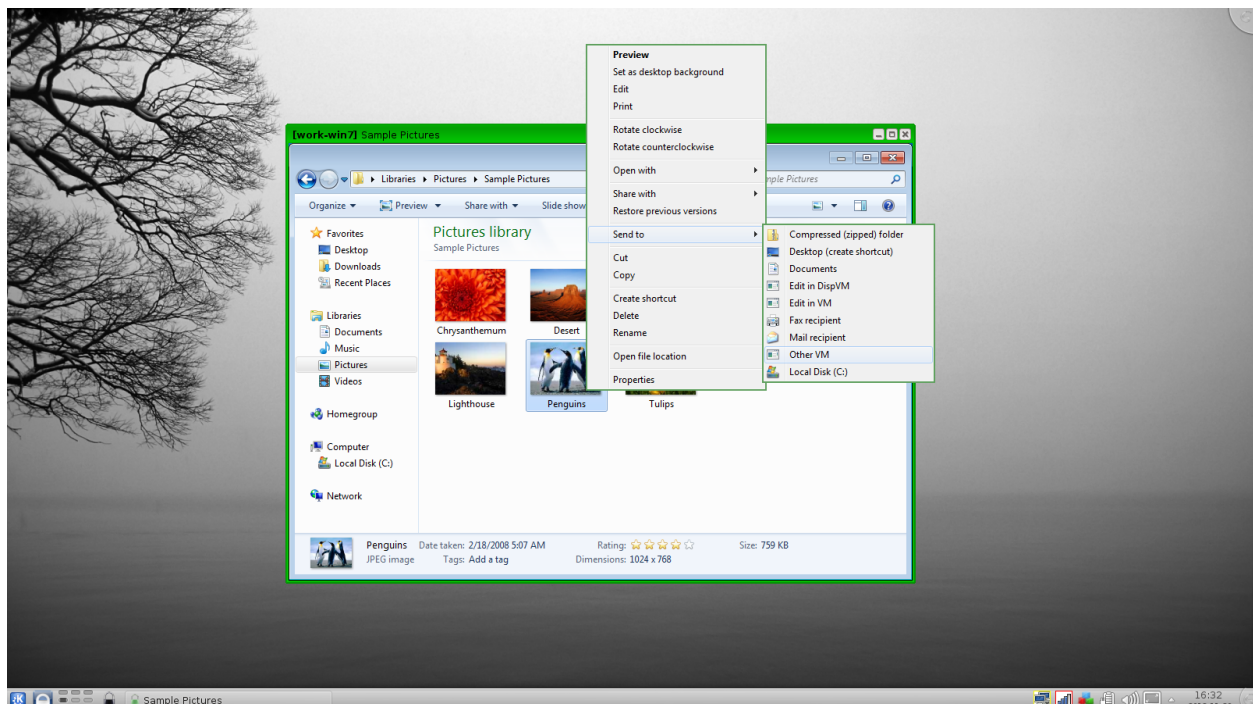
In this example, the word processor runs in the “work” domain, which has been assigned the “blue” label. It is fully isolated from other domains, such as the “untrusted” domain (assigned the “red” label – “Watch out!”, “Danger!”) used for random Web browsing, news reading, as well as from the “work-web” domain (assigned the “yellow” label), which is used for work-related Web browsing that is not security critical. Apps from different domains run in different AppVMs and have different X servers, filesystems, etc. Notice the different color frames (labels) and VM names in



the titlebars. These are drawn by the trusted Window Manager running in Dom0, and apps running in domains cannot fake them:

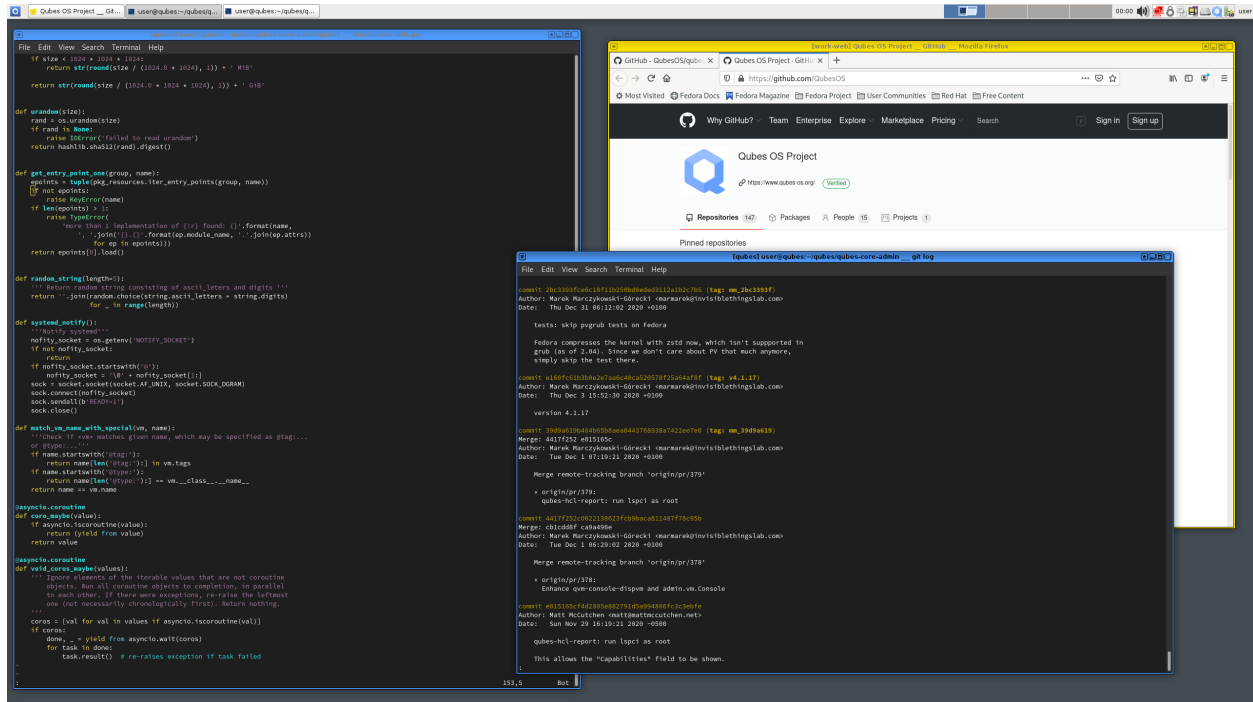


Qubes Release 2 can also run Windows AppVMs in seamless mode, integrated onto the common Qubes trusted desktop, just like Linux AppVMs! The seamless GUI integration has been introduced in Qubes R2 Beta 3. This requires [Qubes Windows Tools](#) to be installed in the Windows VMs first.

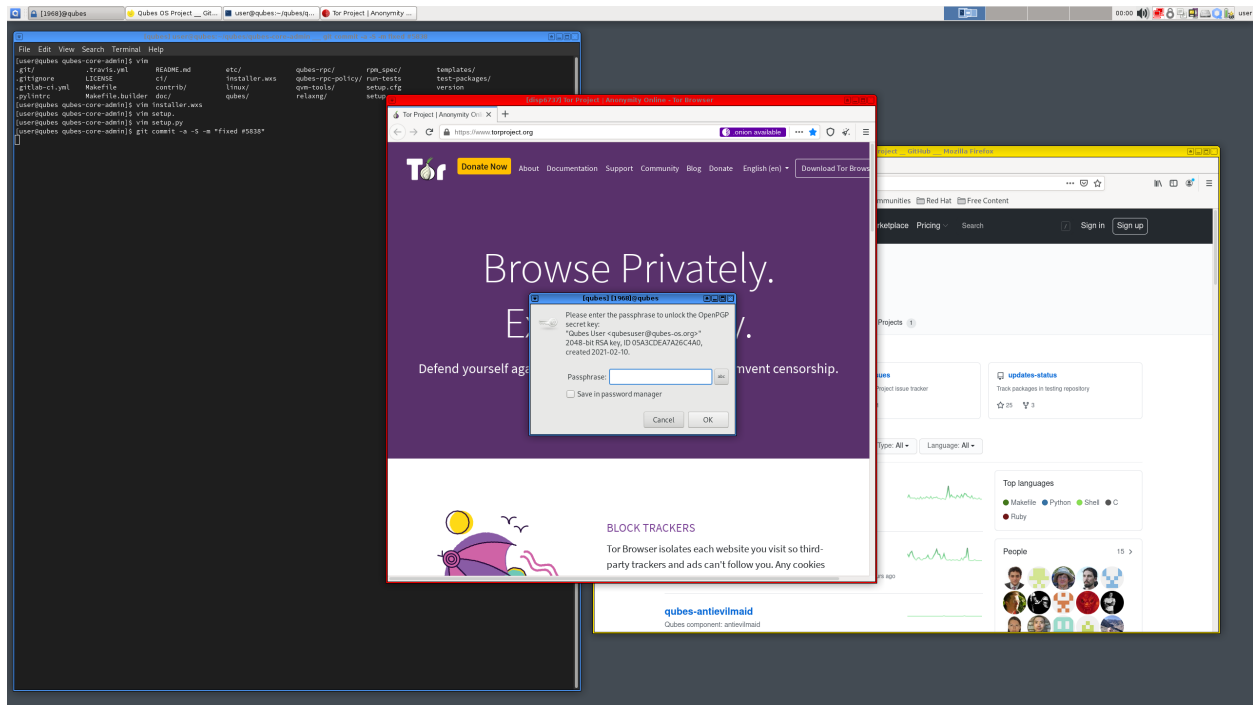




Windows AppVMs are fully integrated with the rest of the Qubes OS system, which includes things such as secure, policy governed, inter-VM file copy, clipboard, and generally our whole elastic qrexec infrastructure for secure inter-VM RPC! Starting with Qubes R2 Beta 3 we also support HVM-based templates allowing to instantly create many Windows AppVMs with shared “root filesystem” from the Template VM (but one should ensure their license allows for such instantiation of the OS in the template). Just like with Linux AppVMs!



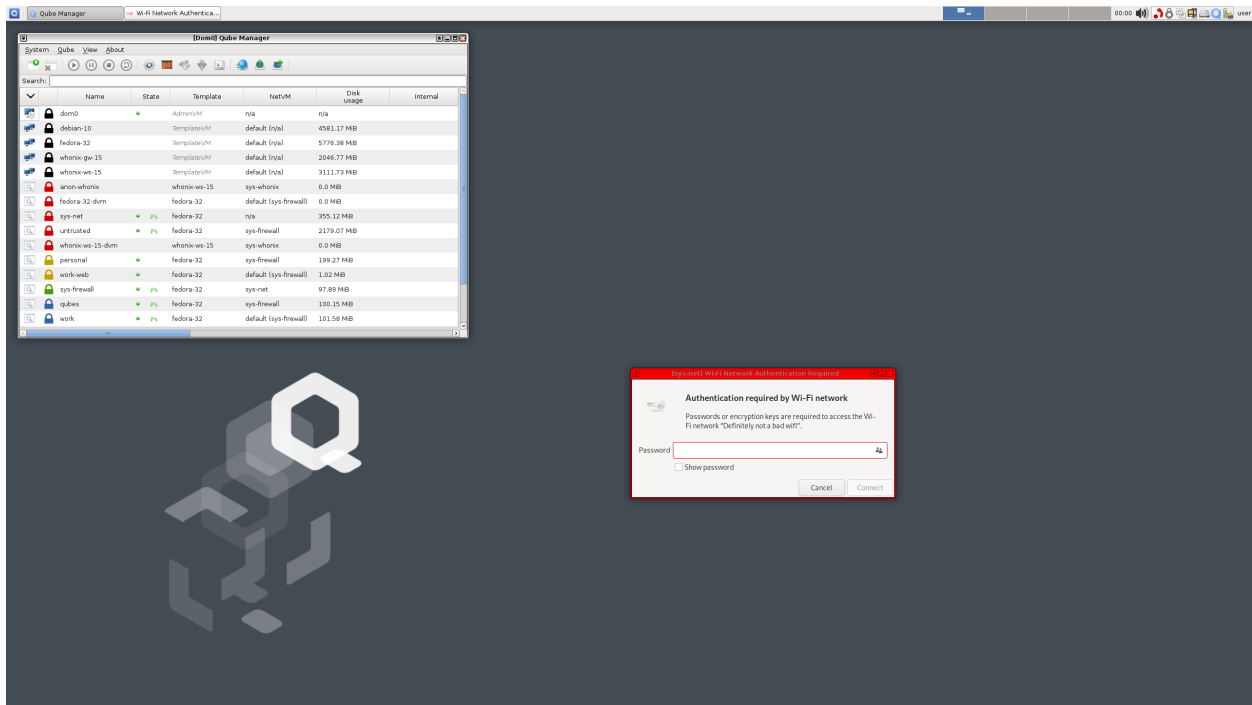
Here we see Xfce4.14 Window Manager running in Dom0 (instead of KDE as on previous versions). Qubes supports customized Xfce4 in dom0 beginning with R2 Beta 2!



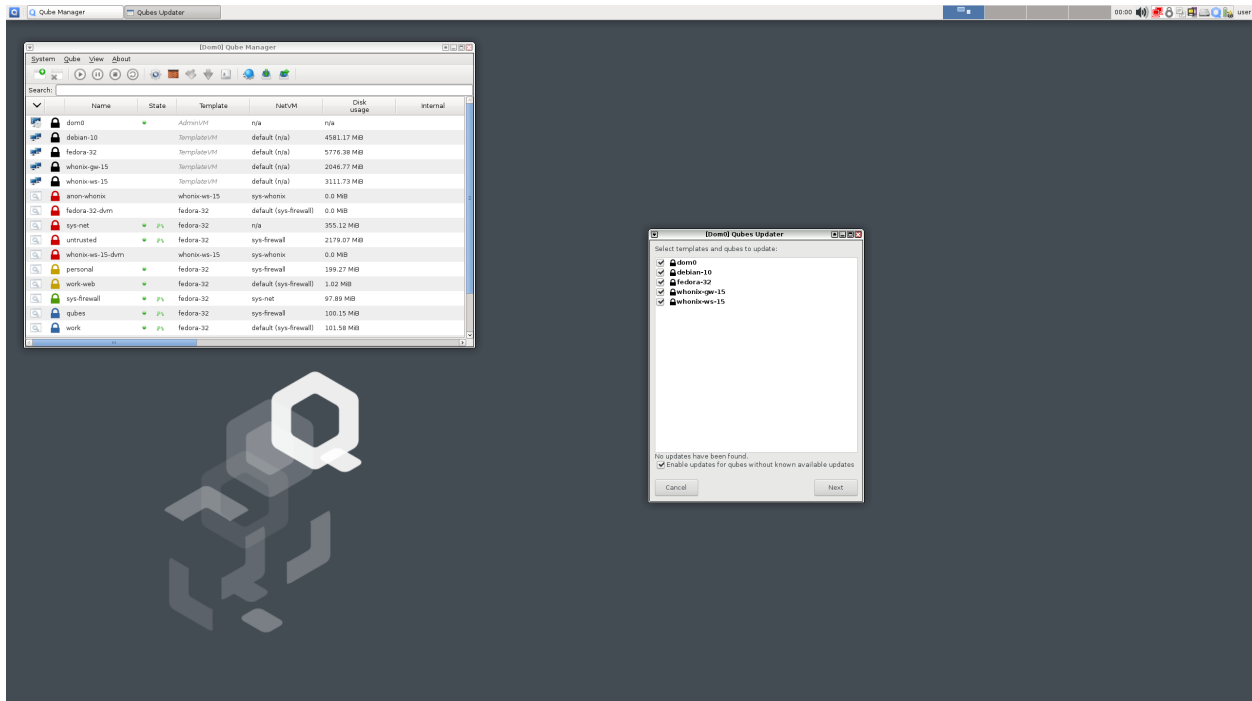
It is always clearly visible to which domain a given window belongs. Here it's immediately clear that the passphrase-prompting window belongs to some domain with the “blue” label. When we look at the titlebar, we see “[qubes]”, which is the name of the actual domain. Theoretically, the untrusted application (here, the red Tor Browser running in a DisposableVM) beneath the prompt window could draw a similar looking window within its contents. In practice, this would be very hard, because it doesn't know, e.g., the exact decoration style that is in use. However, if this is a concern, the user can simply try to move the more trusted window onto some empty space on the desktop such that no other window is present beneath it. Or, better yet, use the Expose-like effect (available via a hot-key). A malicious application from an untrusted domain cannot spoof the whole desktop because the trusted Window Manager will never let any domain “own” the whole screen. Its titlebar will always be visible.



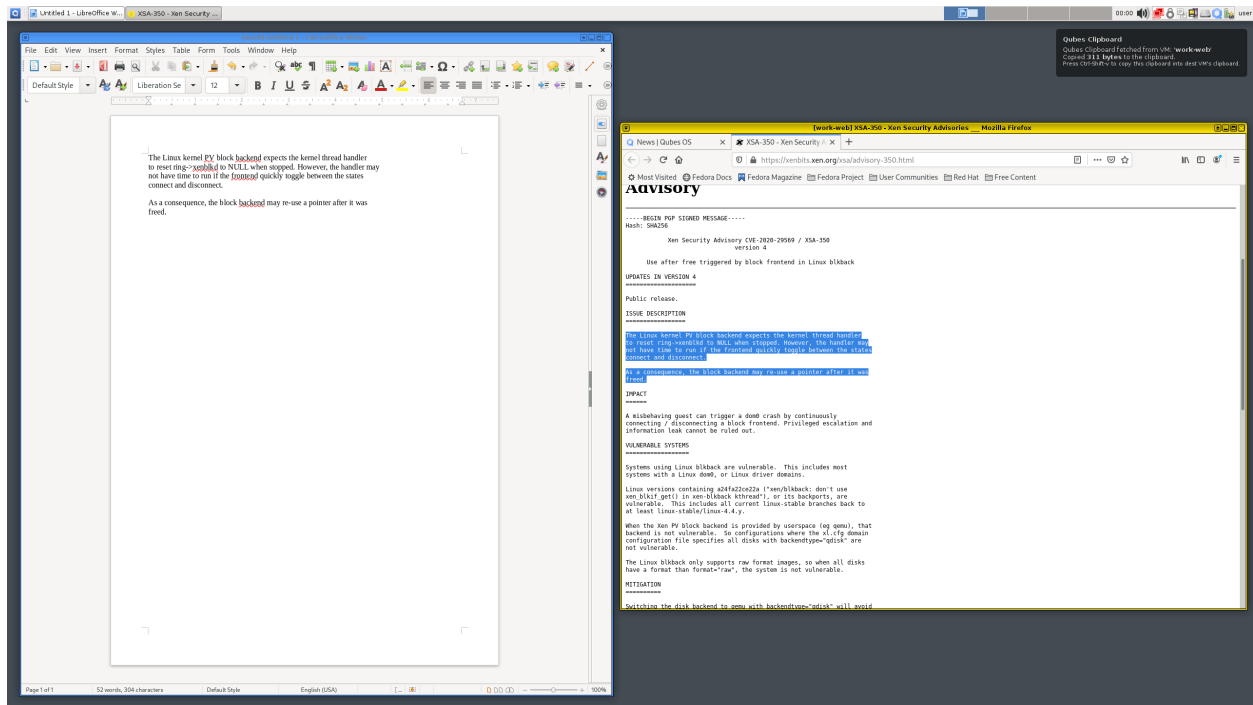
Qubes is all about seamless integration from the user's point of view. Here you can see how it virtualizes tray icons from other domains. Notice the network icon is in red. This icon is in fact managed by the Network Manager running in a separate NetVM.



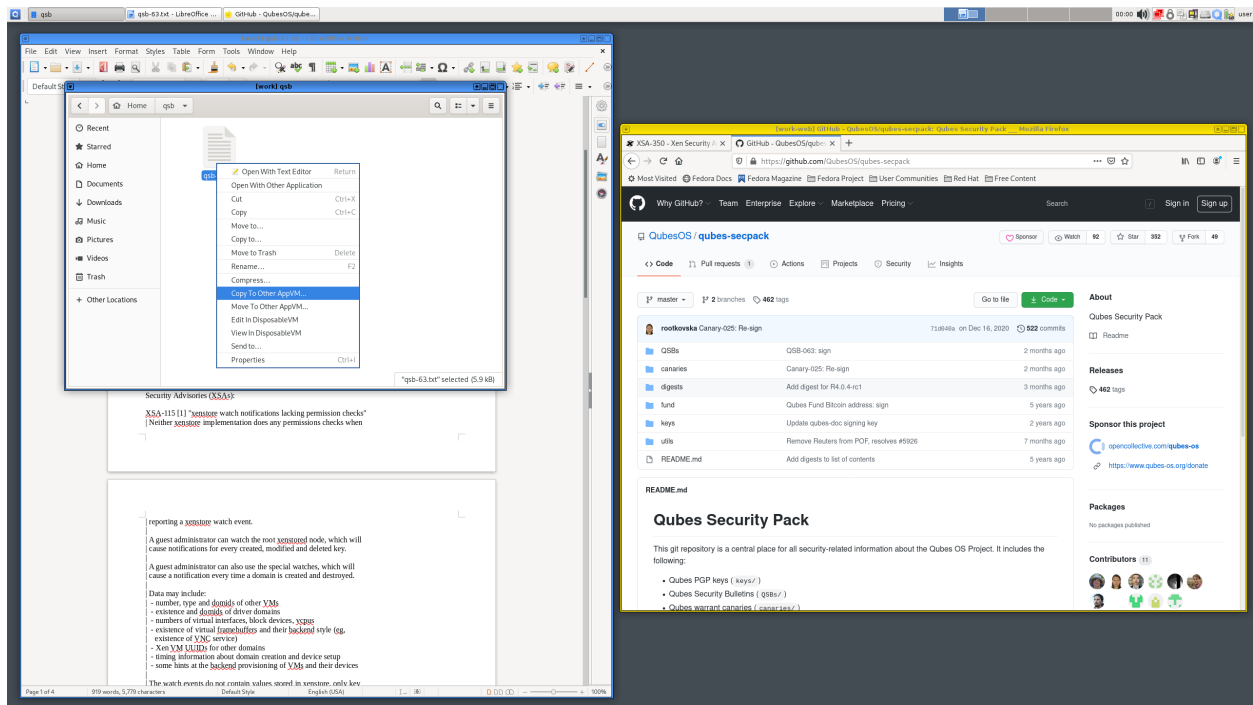
All the networking runs in a special, unprivileged NetVM. (Notice the red frame around the Network Manager dialog box on the screen above.) This means that in the event that your network card driver, Wi-Fi stack, or DHCP client is compromised, the integrity of the rest of the system will not be affected! This feature requires Intel VT-d or AMD IOMMU hardware (e.g., Core i5/i7 systems) \* \* \* \*

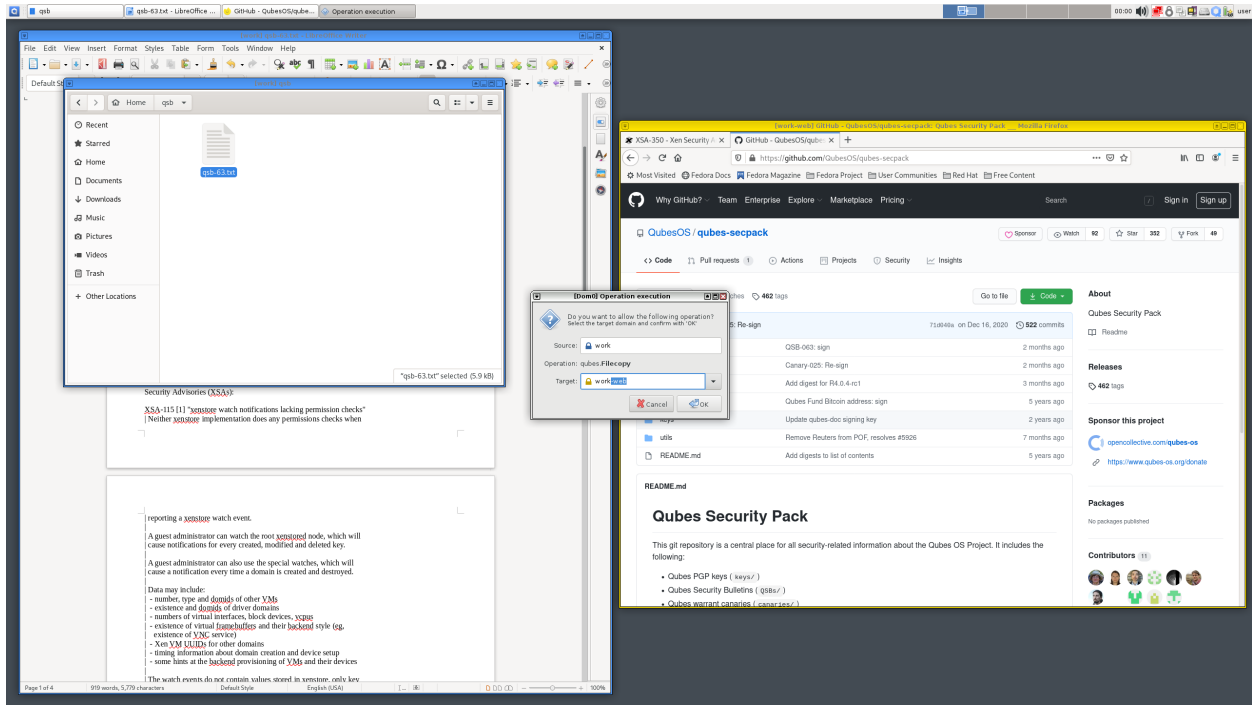


Qubes lets you update all the software in all the domains all at once, in a centralized way. This is possible thanks to Qubes' unique TemplateVM technology. Note that the user is not required to shut down any AppVMs (domains) for the update process. This can be done later, at a convenient moment, and separately for each AppVM.

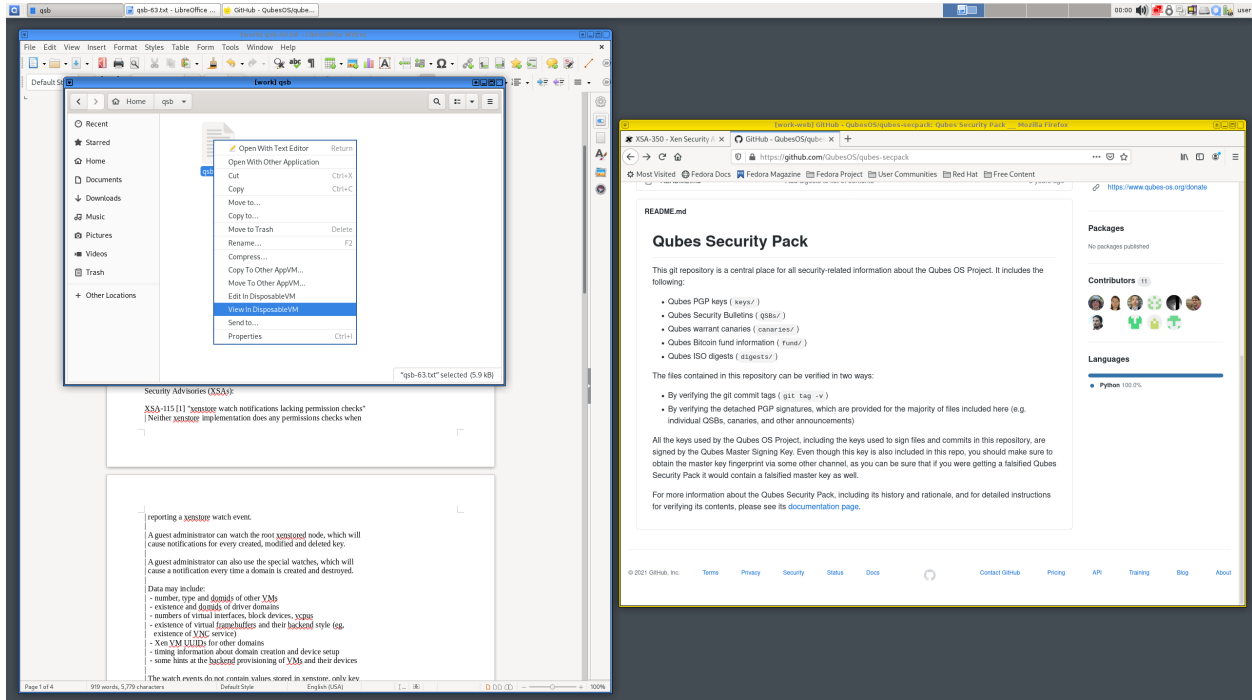


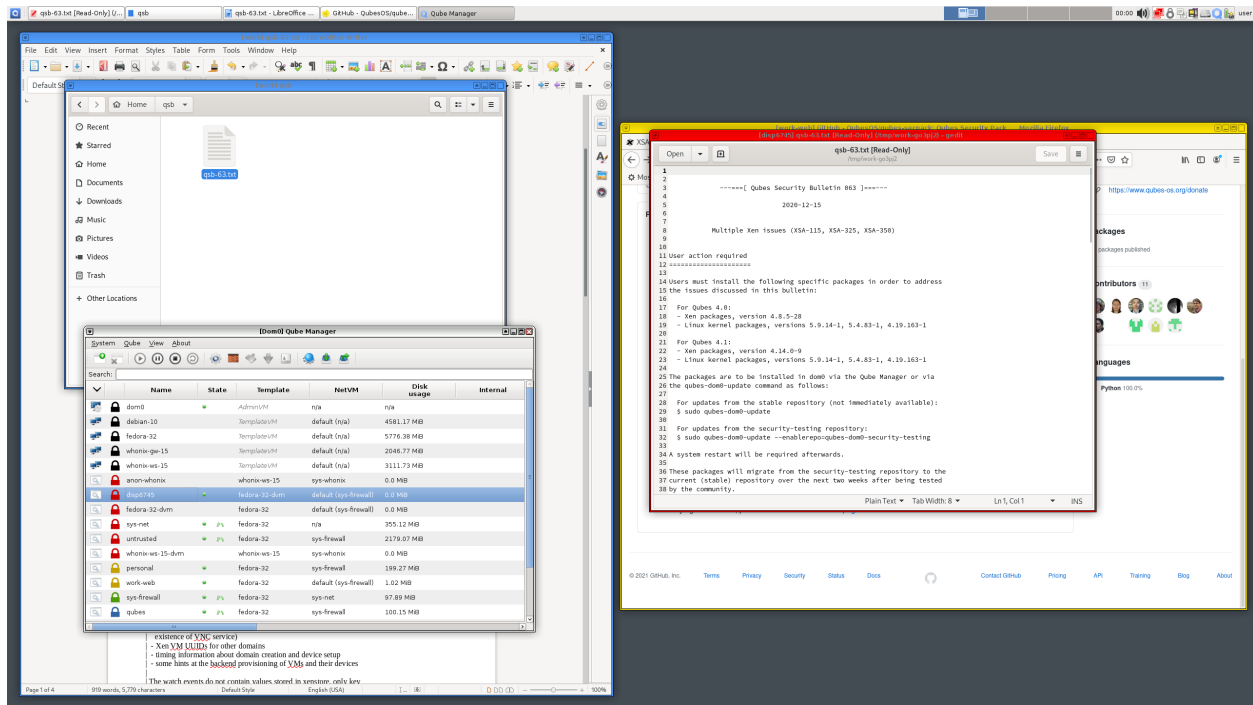
Qubes supports secure copy-and-paste operations between AppVMs. Only the user can initiate a copy or paste operation using a special key combination (Ctrl-Shift-C/V). Other AppVMs have no access to the clipboard buffer, so they cannot steal data from the clipboard. Only the user decides which AppVM should be given access to the clipboard. (This is done by selecting the destination AppVM's window and pressing the Ctrl-Shift-V combination.)



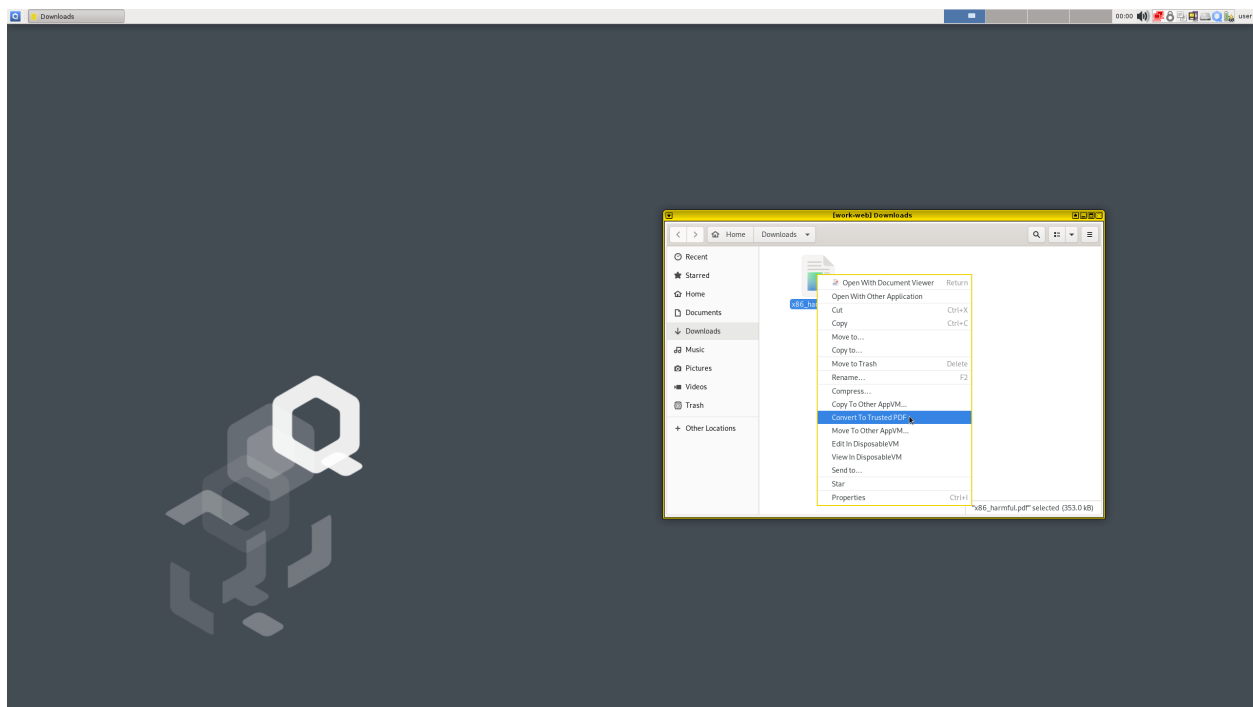


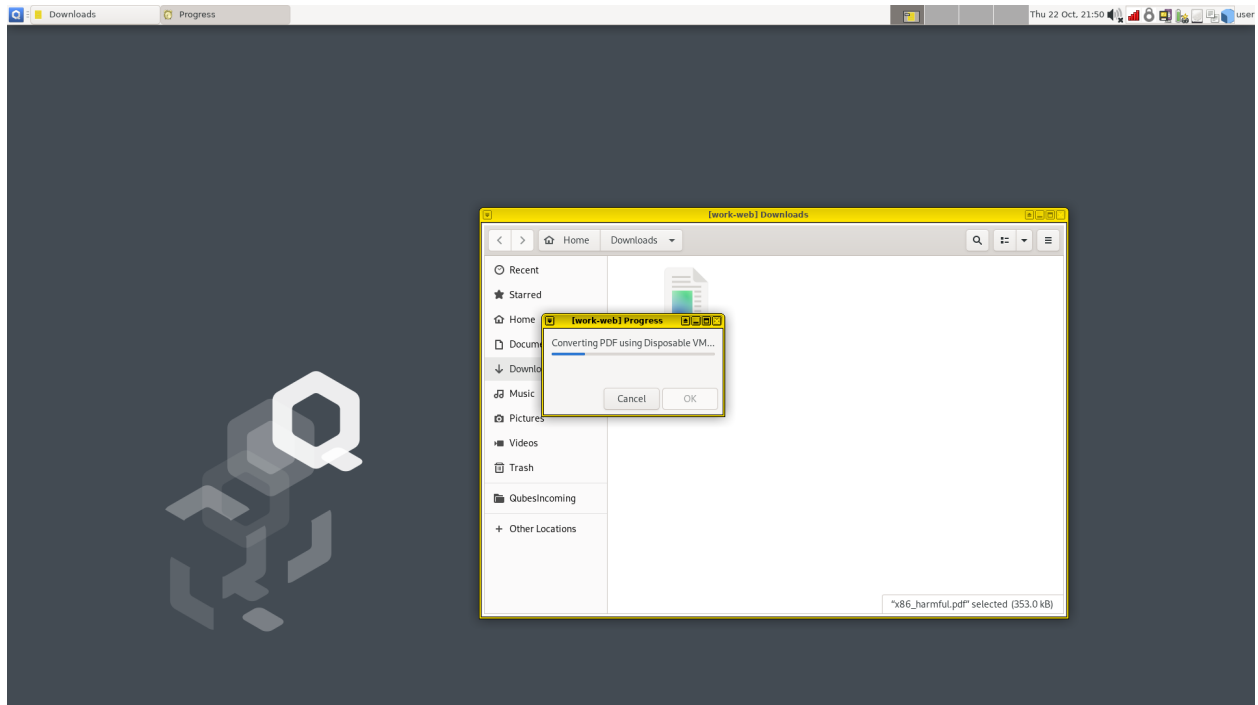
Qubes also supports secure file copying between AppVMs.



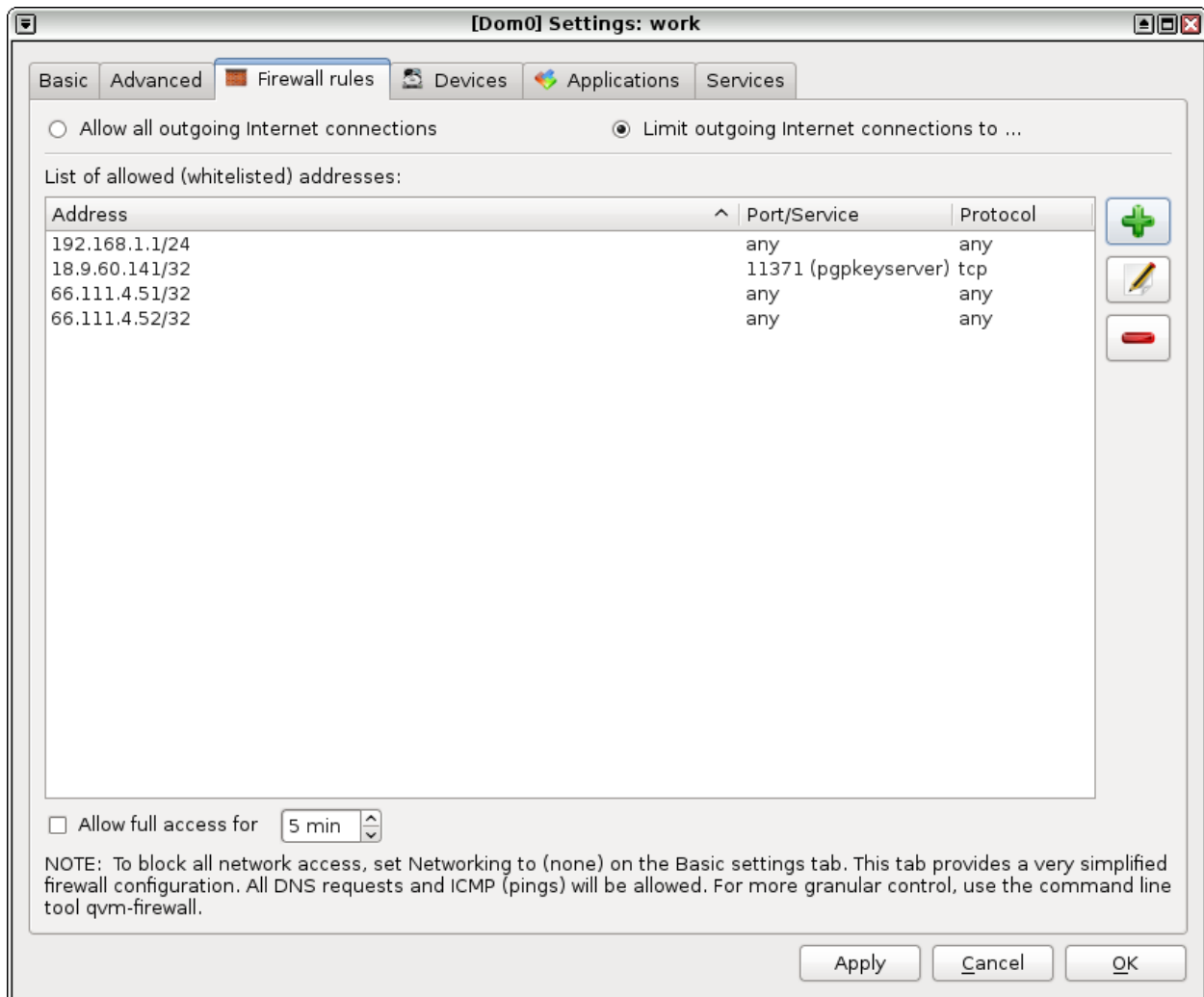


Qubes' unique DisposableVMs (DispVMs) allow the user to open any file in a disposable VM in a matter of seconds! A file can be edited in a disposable VM, and any changes are projected back onto the original file. Currently, there is no way to mark files to be automatically opened in a disposable VM (one needs to right-click on the file and choose the "View in DisposableVM" or "Edit in DisposableVM" option), but this is planned for the R2 Beta 3 release.





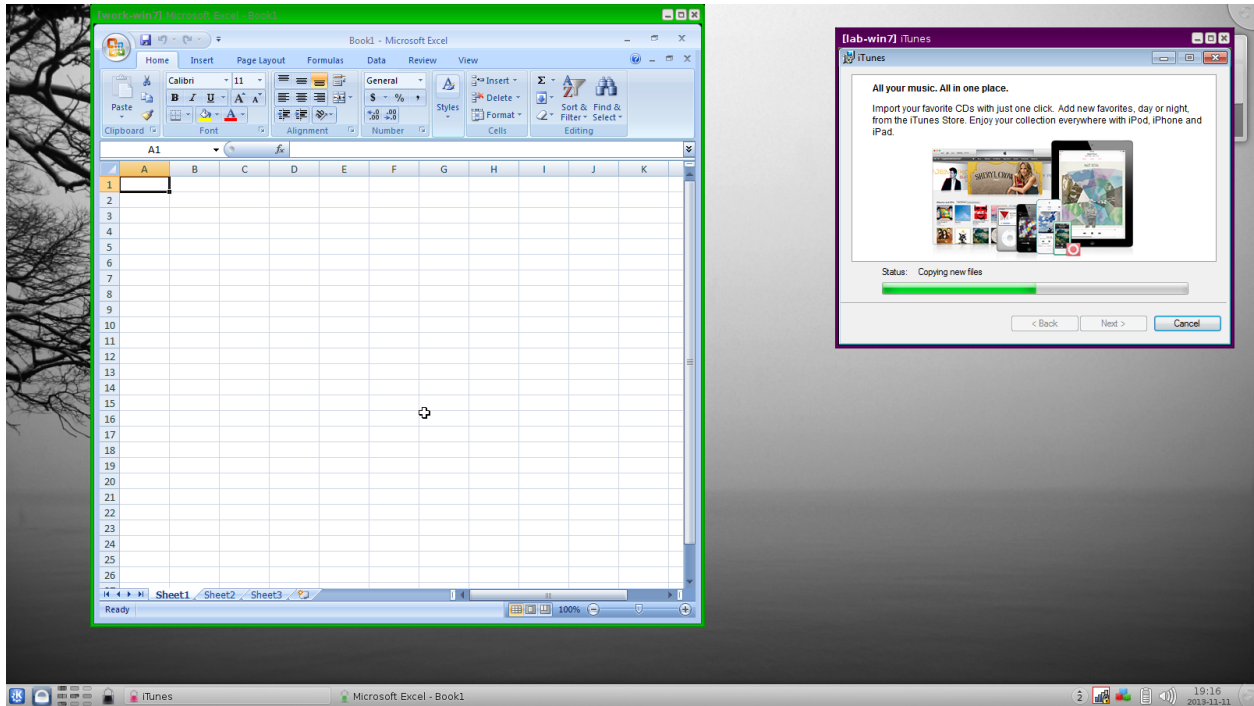
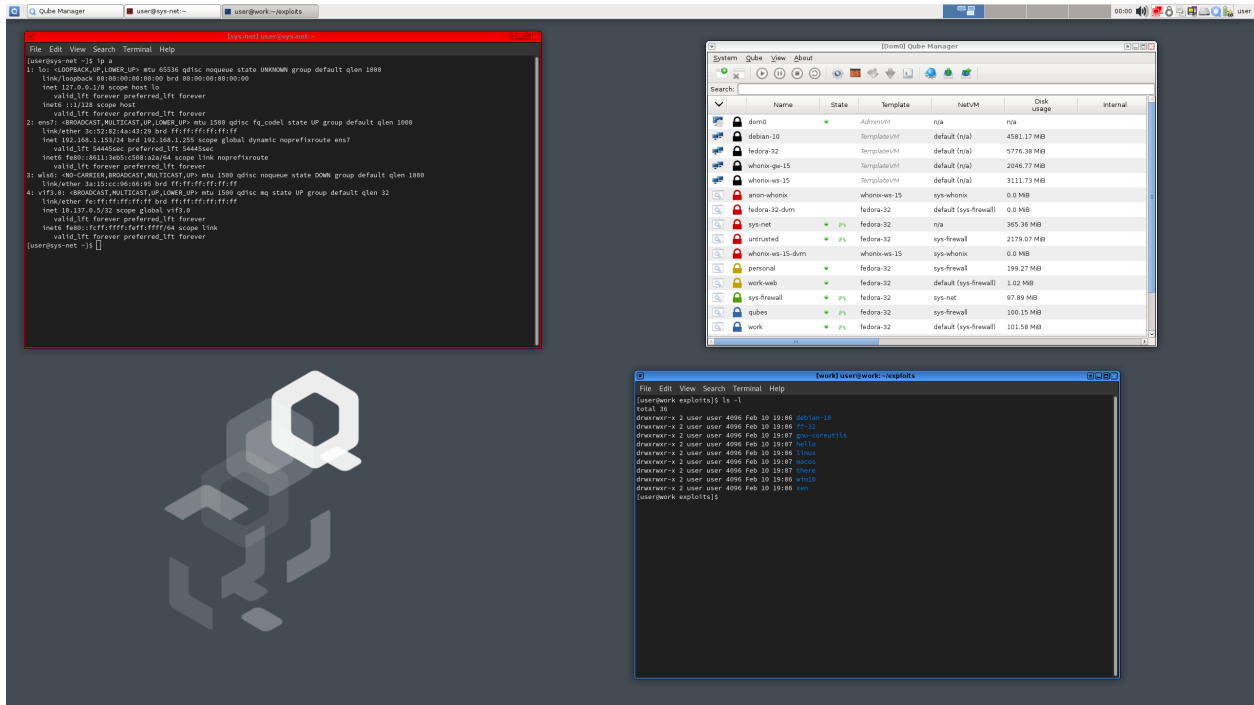
Qubes provides an advanced infrastructure for programming inter-VM services, such as a PDF converter for untrusted files (which is described in [this article](#)).



Qubes provides a dedicated firewall that itself runs in an isolated FirewallVM.

And some more screenshots:





## 1.3 Video tours

### 1.3.1 Qubes OS Summit 2022

Watch all the talks from Qubes OS Summit 2022, which took place September 9-11, 2022 in Berlin, Germany.

---

---

---

### 1.3.2 Micah Lee presents “Qubes OS: The Operating System That Can Protect You Even If You Get Hacked”

Micah Lee, a long-time Qubes advocate, presented [Qubes OS: The Operating System That Can Protect You Even If You Get Hacked](#) at the [Circle of HOPE](#) conference, which took place July 20-22, 2018 in New York City.

---

### 1.3.3 Explaining Computers presents “Qubes OS: Security Oriented Operating System”

## 1.4 Getting started

After downloading and *installing* Qubes OS, it’s time to dive in and get to work! (Already know your way around? Dive right in to *organizing your qubes*.)

### 1.4.1 The Basics

Qubes OS is an operating system built out of securely-isolated compartments called *qubes*. For example, you might have a work qube, a personal qube, a banking qube, a web browsing qube, and so on. You can have as many qubes as you want! Most of the time, you’ll be using an *app qube*, which is a qube intended for running software programs like web browsers, email clients, and word processors. Each app qube is based on another type of qube called a *template*. More than one qube can be based on the same template. Importantly, a qube cannot modify its template in any way. This means that, if a qube is ever compromised, its template and any other qubes based on that template will remain safe. This is what makes Qubes OS so secure. Even if an attack is successful, the damage is limited to a single qube.

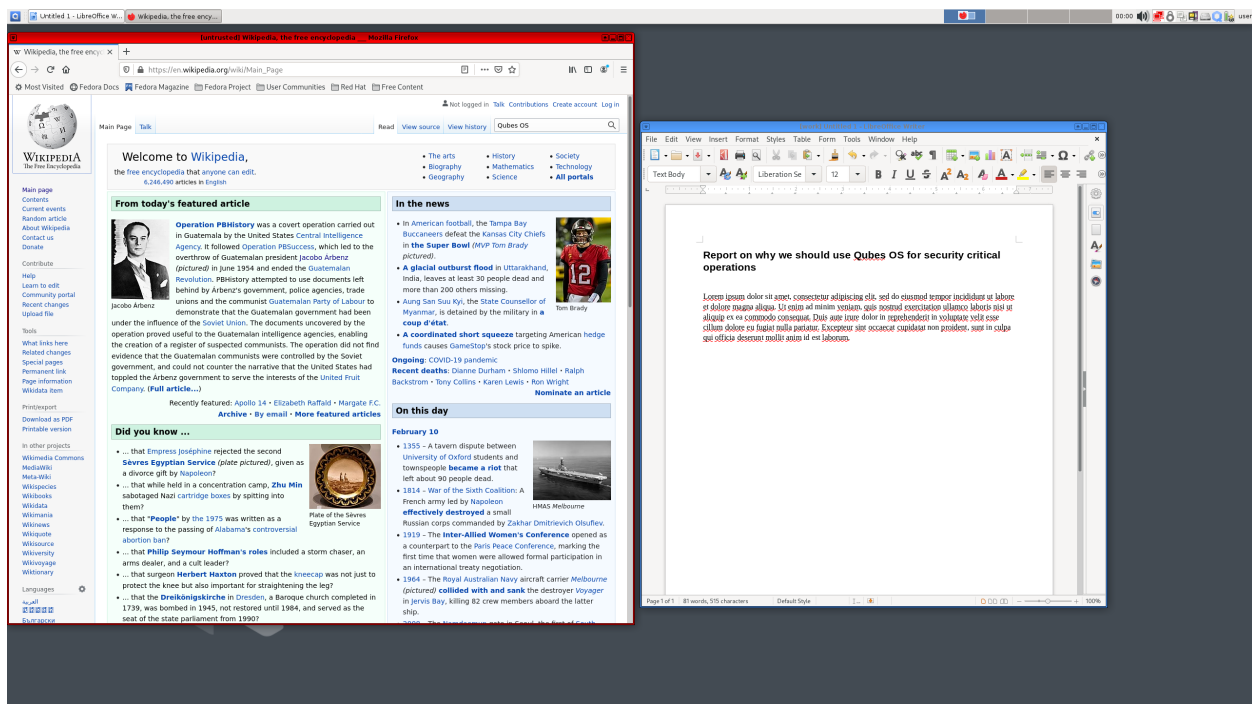
Suppose you want to use your favorite web browser in several different qubes. You’d install the web browser in a template, then every qube based on that template would be able to run the web browser software (while still being forbidden from modifying the template and any other qubes). This way, you only have to install the web browser a single time, and updating the template serves to update all the qubes based on it. This elegant design saves time and space while enhancing security.

There are also some “helper” qubes in your system. Each qube that connects to the Internet does so through a network-providing *service qube*. If you need to access USB devices, another service qube will do that. There’s also a *management qube* that automatically handles a lot of background housekeeping. For the most part, you won’t have to worry about it, but it’s nice to know that it’s there. As with app qubes, service qubes and management qubes are also based on templates. Templates are usually named after their operating system (often a [Linux distribution](#)) and corresponding version number. There are many ready-to-use *templates* to choose from, and you can download and have as many as you like.

Last but not least, there's a very special *admin qube* which, as the name suggests, is used to administer your entire system. There's only one admin qube, and it's called *dom0*. You can think of it as the master qube, holding ultimate power over everything that happens in Qubes OS. Dom0 is more trusted than any other qube. If dom0 were ever compromised, it would be "game over." The entire system would effectively be compromised. That's why everything in Qubes OS is specifically designed to protect dom0 and ensure that doesn't happen. Due to its overarching importance, dom0 has no network connectivity and is used only for running the *desktop environment* and *window manager*. Dom0 should never be used for anything else. In particular, you should never run user applications in dom0. (That's what your app qubes are for!)

## Color & Security

You'll choose a **color** for each of your qubes out of a predefined set of colors. Each window on your desktop will have its frame colored according to the color of that qube. These colored frames help you keep track of which qube each window belongs to and how trustworthy it is. This is especially helpful when you have the same app running in multiple qubes at the same time. For example, if you're logged in to your bank account in one qube while doing some random web surfing in a different qube, you wouldn't want to accidentally enter your banking password in the latter! The colored frames help to avoid such mistakes.



Most Qubes users associate red with what's untrusted and dangerous (like a red light: stop! danger!), green with what's safe and trusted, and yellow and orange with things in the middle. This color scheme also extends to include blue and black, which are usually interpreted as indicating progressively more trusted domains than green, with black being ultimately trusted. Color and associated meanings are ultimately up to you, however. The system itself does not treat the colors differently. If you create two identical qubes — black and red, say — they'll be the same until you start using them differently. Feel free to use the colors in whatever way is most useful to you. For example, you might decide to use three or four qubes for work activities and give them all the same color — or all different colors. It's entirely up to you.

### User Interface

On operating systems like Windows and macOS, the desktop environment is unchangeable and part of that operating system. With Linux, any of a number of desktop environments are an option. Qubes OS is installed with XFCE as its default desktop environment, but it also supports *KDE*, as well as the window managers *i3* and *AwesomeWM*.



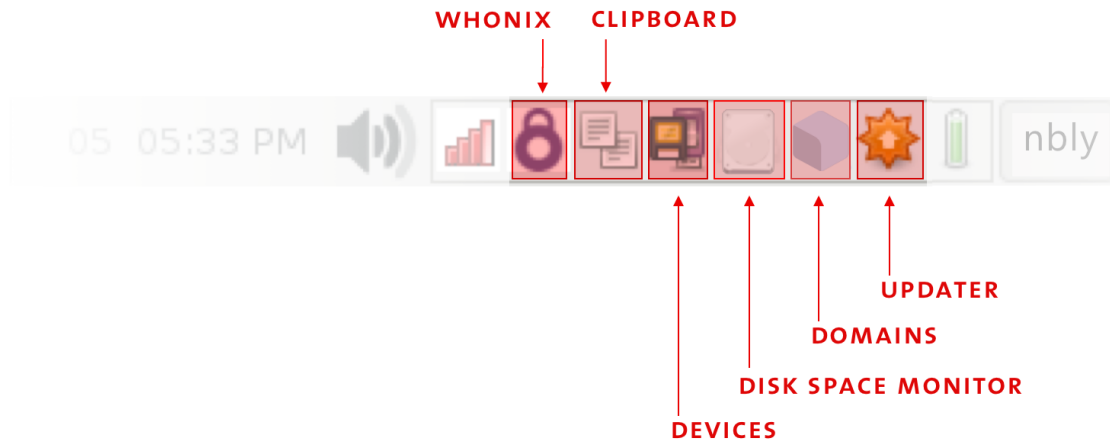
The bar at the top of your screen in Qubes 4.0 includes the following XFCE component areas:

- The **Tray**, where many functional widgets live.
- **Spaces**, an interface for [virtual desktops](#). Virtual desktops do not have any inherent security isolation properties, but some users find them useful for organizing things.
- The **Task Bar** where buttons for open and hidden windows live.
- The **App Menu**, where you go to open an application within a qube, to open a dom0 terminal, to access administrative UI tools such as the Qube Manager, or to access settings panels for your desktop environment.

To learn more about how to customize your desktop environment, we recommend you spend some time going through [XFCE's documentation](#).

There are several tray widgets that are unique to Qubes OS:

- The **Whonix SDWDate** allows you to control the Tor connection in your [sys-whonix](#) qube.
- The **Qubes Clipboard** lets you easily copy text from dom0.
- The **Qubes Devices** widget allows you to attach and detach devices — such as USB drives and cameras — to qubes.
- The **Qubes Disk Space** widget shows you how much storage you're using. It'll notify you if you're ever running out of space.
- The **Qubes Domains** widget allows you to manage running qubes, turn them on and off, and monitor RAM and CPU usage.
- The **Qubes Updater** widget informs you when updates are available and helps you install them.



## Qube Manager

To see all of your qubes at the same time, you can use the **Qube Manager** (go to the App Menu → Qubes Tools → Qube Manager), which displays the states of all the qubes in your system, even the ones that aren't running.

[Dom0] Qube Manager						
System Qube View About						
Search:						
	Name	State	Template	NetVM	Disk usage	Internal
	dom0		AdminVM	n/a	n/a	
	debian-10		TemplateVM	default (n/a)	4581.17 MiB	
	fedora-32		TemplateVM	default (n/a)	5231.41 MiB	
	whonix-gw-15		TemplateVM	default (n/a)	2044.72 MiB	
	whonix-ws-15		TemplateVM	default (n/a)	3112.76 MiB	
	anon-whonix		whonix-ws-15	sys-whonix	0.0 MiB	
	disp6018		whonix-ws-15-dvm	sys-whonix	8.6 MiB	
	fedora-32-dvm		fedora-32	default (sys-firewall)	0.0 MiB	
	sys-net		fedora-32	n/a	1021.95 MiB	
	untrusted		fedora-32	sys-firewall	2.05 MiB	
	whonix-ws-15-dvm		whonix-ws-15	sys-whonix	0.0 MiB	
	personal		fedora-32	sys-firewall	0.0 MiB	
	sys-firewall		fedora-32	sys-net	0.0 MiB	
	work		fedora-32	sys-firewall	0.0 MiB	
	default-mgmt-dvm		fedora-32	n/a	0.0 MiB	Yes

### Command-line interface

All aspects of Qubes OS can be controlled using command-line tools. Opening a terminal emulator in dom0 can be done in several ways:

- Go to the App Menu and select **Terminal Emulator** at the top.
- Press **Alt+F3** and search for `xfce terminal`.
- Right-click on the desktop and select **Open Terminal Here**.

Terminal emulators can also be run in other qubes as normal programs. Various command-line tools are described as part of this guide, and the whole reference can be found [here](#).

### 1.4.2 First boot

When you install Qubes OS, a number of qubes are pre-configured for you:

- **Templates:** `fedora-XX` (XX being the version number)
- **Admin qube:** `dom0`
- **Service qubes:** `sys-usb`, `sys-net`, `sys-firewall`, and `sys-whonix`
- **App qubes** configured to prioritize security by compartmentalizing tasks and types of data: `work`, `personal`, `untrusted`, and `vault`. (There is nothing special about these qubes. If you were to create a black qube and name it `vault`, it would be the same as the pre-configured `vault` qube. They're just suggestions to get you started. )

A variety of open-source applications such as file managers, command-line terminals, printer managers, text editors, and “applets” used to configure different things like audio or parts of the user interface are also installed by default—most within the templates. Most are bundled with each template.

### Adding, removing, and listing qubes

You can easily create a new qube with the **Create Qubes VM** option in the App Menu. If you need to add or remove qubes, simply use the Qube Manager's **Add** and **Remove** buttons. You can also add, remove, and list qubes from the command line using the following tools:

- `qvm-create`
- `qvm-remove`
- `qvm-ls`

### How many qubes do I need?

That's a great question, but there's no one-size-fits-all answer. It depends on the structure of your digital life, and this is at least a little different for everyone. If you plan on using your system for work, then it also depends on what kind of job you do.

It's a good idea to start out with the qubes created automatically by the installer: `work`, `personal`, `untrusted`, and `vault`. If and when you start to feel that some activity just doesn't fit into any of your existing qubes, or you want to partition some part of your life, you can easily create a new qube for it. You'll also be able to easily *copy any files* you need to the newly-created qube.

Want to see some examples? Check out our in-depth guide on [how to organize your qubes](#), which walks through several common use cases based on our user research and years of experience from veteran Qubes users.

### 1.4.3 Secure Habits

It is *very important* to *keep Qubes updated* to ensure you have the latest security updates. Frequently updating is one of the best ways to remain secure against new threats.

It's also *very important* to make regular backups so that you don't lose your data unexpectedly. The *Qubes backup system* allows you to do this securely and easily.

### 1.4.4 How-To Guides

Here are some basic tasks you're likely to want to perform often that are unique to Qubes as a multi-environment system. A full list is available in the *How-To Guides* section in the docs.

- *How to organize your qubes*
- *How to Update*
- *How to Back Up, Restore, and Migrate*
- *How to Copy and Paste Text*
- *How to Copy and Move Files*
- *How to Copy from Dom0*
- *How to Install Software*
- *How to Use Devices (block storage, USB, and PCI devices)*

If you encounter any problems, please visit the *Help, Support, Mailing Lists, and Forum* page.

### 1.4.5 Compatible Hardware

Make sure your hardware satisfies the *system requirements*, as Qubes OS cannot run on every type of computer. You may also want to check out *Qubes-certified Hardware* and take a look at the Hardware Compatibility List (HCL).

### 1.4.6 Downloads

Download an ISO, learn how to *verify its authenticity*, and follow our *guide to install Qubes OS*. Looking for the *source code*? You'll find it on [GitHub](#).

### 1.4.7 Documentation

Peruse our extensive library of *documentation* for users and developers of Qubes OS. You can even help us improve it!

## 1.5 Frequently asked questions (FAQ)

### 1.5.1 General & Security

#### What is Qubes OS?

Qubes OS is a security-focused operating system that allows you to organize your digital life into compartments called “qubes.” If one qube is compromised, the others remain safe, so a single cyberattack can no longer take down your entire digital life in one fell swoop. You can think of using Qubes OS as having many different computers on your desk for different activities but with the convenience of a single physical machine, a single unified desktop environment, and a set of tools for using qubes together securely as parts of a unified system.

#### Is Qubes OS free and open-source software?

There are two distinct senses of the word “free” when it comes to free software. The difference is commonly expressed by the phrases “free as in beer” and “free as in speech.”

The first sense is straightforward. Qubes OS is “free as in beer,” meaning that it is provided at no cost (*gratis*), though [donations](#) are greatly appreciated.

The second sense is more complicated. Qubes OS is *mostly* “free as in speech,” but not entirely. All the software created by the Qubes OS Project *itself* is [free](#) (or “*libre*”) and [open-source](#) software (FOSS or FLOSS). This means that everyone is allowed to use, copy, study, and change the software in accordance with its *license*. It also means that the *source code* is [publicly available](#) so everyone can audit and contribute to it.

However, since Qubes OS is a security-focused operating system, it includes some non-free firmware that was not created by the Qubes OS Project (such as CPU microcode), which is necessary in order to protect against known security vulnerabilities. Moreover, the *architecture* of Qubes OS as a meta-operating system means that it incorporates other software (including entire operating systems) from various upstream projects, some of which may include non-free software of their own. In order to make the installation process easier for a wide range of users across many different devices, standard Qubes *templates* also include some non-free firmware and drivers.

Also see: [Will Qubes seek to get certified under the GNU Free System Distribution Guidelines \(GNU FSDG\)?](#)

#### Why is OS security important?

Most people use an operating system like Windows or macOS on their desktop and laptop computers. These OSes are popular because they tend to be easy to use and usually come pre-installed on the computers people buy. However, they present problems when it comes to security. For example, you might open an innocent-looking email attachment or website, not realizing that you’re actually allowing malware (malicious software) to run on your computer. Depending on what kind of malware it is, it might do anything from showing you unwanted advertisements to logging your keystrokes to taking over your entire computer. This could jeopardize all the information stored on or accessed by this computer, such as health records, confidential communications, or thoughts written in a private journal. Malware can also interfere with the activities you perform with your computer. For example, if you use your computer to conduct financial transactions, the malware might allow its creator to make fraudulent transactions in your name.



## Aren't antivirus programs and firewalls enough?

Unfortunately, conventional security approaches like antivirus programs and (software and/or hardware) firewalls are no longer enough to keep out sophisticated attackers. For example, nowadays it's common for malware creators to check to see if their malware is recognized by any signature-based antivirus programs. If it's recognized, they scramble their code until it's no longer recognizable by the antivirus programs, then send it out. The best of these programs will subsequently get updated once the antivirus programmers discover the new threat, but this usually occurs at least a few days after the new attacks start to appear in the wild. By then, it's too late for those who have already been compromised. More advanced antivirus software may perform better in this regard, but it's still limited to a detection-based approach. New zero-day vulnerabilities are constantly being discovered in the common software we all use, such as our web browsers, and no antivirus program or firewall can prevent all of these vulnerabilities from being exploited.

## How does Qubes OS provide security?

Qubes takes an approach called **security by compartmentalization**, which allows you to compartmentalize the various parts of your digital life into securely isolated compartments called *qubes*.

This approach allows you to keep the different things you do on your computer securely separated from each other in isolated qubes so that one qube getting compromised won't affect the others. For example, you might have one qube for visiting untrusted websites and a different qube for doing online banking. This way, if your untrusted browsing qube gets compromised by a malware-laden website, your online banking activities won't be at risk. Similarly, if you're concerned about malicious email attachments, Qubes can make it so that every attachment gets opened in its own single-use *disposable qube*. In this way, Qubes allows you to do everything on the same physical computer without having to worry about a single successful cyberattack taking down your entire digital life in one fell swoop.

Moreover, all of these isolated qubes are integrated into a single, usable system. Programs are isolated in their own separate qubes, but all windows are displayed in a single, unified desktop environment with unforgeable colored window borders so that you can easily identify windows from different security levels. Common attack vectors like network cards and USB controllers are isolated in their own hardware qubes while their functionality is preserved through secure *networking*, *firewalls*, and *USB device management*. Integrated *file* and *clipboard* copy and paste operations make it easy to work across various qubes without compromising security. The innovative *Template* system separates software installation from software use, allowing qubes to share a root filesystem without sacrificing security (and saving disk space, to boot). Qubes even allows you to sanitize PDFs and images in a few clicks. Those concerned about physical hardware attacks will benefit from *Anti Evil Maid*.

## How does Qubes OS provide privacy?

There can be no privacy without security, since security vulnerabilities allow privacy measures to be circumvented. This makes Qubes exceptionally well-suited for implementing effective privacy tools.

Users concerned about privacy will appreciate the *integration of Whonix into Qubes*, which makes it easy to use Tor securely. For more information about how to use this powerful tool correctly and safely, please see *Qubes-Whonix Guides*.

For the privacy policies covering our website, repositories, Qubes OS itself, and more, please see *Privacy Policy*.

## What about privacy in non-Whonix qubes?

The main way Qubes OS *provides privacy* is via its *integration with Whonix*. Qubes OS does not claim to provide special privacy (as opposed to security) properties in non-Whonix qubes. This includes *disposables*.

Privacy is far more difficult than is commonly understood. In addition to the *web browser*, there is also *VM fingerprinting* and *advanced deanonymization attacks* that most users have never considered (and this is just to mention a few examples). The *Whonix Project* specializes in *protecting against these risks*.

In order to achieve the same results in non-Whonix qubes (including disposables), one would have to reinvent Whonix. Such duplication of effort makes no sense when Whonix already exists and is already integrated into Qubes OS.

Therefore, when you need privacy, you should use Whonix qubes. Remember, though, that privacy is difficult to achieve and maintain. Whonix is a powerful tool, but no tool is perfect. Read the *documentation* thoroughly and exercise care when using it.

## How does Qubes OS compare to using a “live CD” OS?

Booting your computer from a live CD (or DVD) when you need to perform sensitive activities can certainly be more secure than simply using your main OS, but this method still preserves many of the risks of conventional OSes. For example, popular live OSes (such as *Tails* and other Linux distributions) are still **monolithic** in the sense that all software is still running in the same OS. This means, once again, that if your session is compromised, then all the data and activities performed within that same session are also potentially compromised.

## How does Qubes OS compare to running VMs in a conventional OS?

Not all virtual machine software is equal when it comes to security. You may have used or heard of VMs in relation to software like VirtualBox or VMware Workstation. These are known as “Type 2” or “hosted” hypervisors. (The **hypervisor** is the software, firmware, or hardware that creates and runs virtual machines.) These programs are popular because they’re designed primarily to be easy to use and run under popular OSes like Windows (which is called the **host** OS, since it “hosts” the VMs). However, the fact that Type 2 hypervisors run under the host OS means that they’re really only as secure as the host OS itself. If the host OS is ever compromised, then any VMs it hosts are also effectively compromised.

By contrast, Qubes uses a “Type 1” or “bare-metal” hypervisor called *Xen*. Instead of running inside an OS, Type 1 hypervisors run directly on the “bare metal” of the hardware. This means that an attacker must be capable of subverting the hypervisor itself in order to compromise the entire system, which is vastly more difficult.

Qubes makes it so that multiple VMs running under a Type 1 hypervisor can be securely used as an integrated OS. For example, it puts all of your application windows on the same desktop with special colored borders indicating the trust levels of their respective VMs. It also allows for things like secure copy/paste operations between VMs, securely copying and transferring files between VMs, and secure networking between VMs and the Internet.

## How does Qubes OS compare to using a separate physical machine?

Using a separate physical computer for sensitive activities can certainly be more secure than using one computer with a conventional OS for everything, but there are still risks to consider. Briefly, here are some of the main pros and cons of this approach relative to Qubes:

---

### Important: Pros

---

- Physical separation doesn’t rely on a hypervisor. (It’s very unlikely that an attacker will break out of Qubes’ hypervisor, but if one were to manage to do so, one could potentially gain control over the entire system.)

- Physical separation can be a natural complement to physical security. (For example, you might find it natural to lock your secure laptop in a safe when you take your unsecure laptop out with you.)

*<i class="fa fa-times"></i> <strong>Cons</strong>*

- Physical separation can be cumbersome and expensive, since we may have to obtain and set up a separate physical machine for each security level we need.
- There's generally no secure way to transfer data between physically separate computers running conventional OSes. (Qubes has a secure inter-VM file transfer system to handle this.)
- Physically separate computers running conventional OSes are still independently vulnerable to most conventional attacks due to their monolithic nature.
- Malware which can bridge air gaps has existed for several years now and is becoming increasingly common.

(For more on this topic, please see the paper [Software compartmentalization vs. physical separation](#).)

### What is the main concept behind Qubes?

To build security on the “Security by Compartmentalization (or Isolation)” principle.

### What about other approaches to security?

The other two popular [approaches](#) are “Security by Correctness” and “Security by Obscurity.” We don't believe either of these approaches are capable of providing reasonable security today, nor do we believe that they will be capable of doing so in the foreseeable future.

### How is Qubes different from other security solutions?

Please see this [article](#) for a thorough discussion.

### Is Qubes just another Linux distribution?

If you really want to call it a distribution, then it's more of a “Xen distribution” than a Linux one. But Qubes is much more than just Xen packaging. It has its own VM management infrastructure, with support for template VMs, centralized VM updating, etc. It also has a very unique GUI virtualization infrastructure.

### What about safe languages and formally verified microkernels?

In short: these are non-realistic solutions today. We discuss this in further depth in our [Architecture Specification document](#).

### Why does Qubes use virtualization?

We believe that this is currently the only practically viable approach to implementing strong isolation while simultaneously providing compatibility with existing applications and drivers.

### Does Qubes use full disk encryption (FDE)?

By default, Qubes OS uses [LUKS/dm-crypt](#) to encrypt everything except the /boot partition.

### What do all these terms mean?

All Qubes-specific terms are defined in the [glossary](#)

### Does Qubes run every app in a separate VM?

No! This would not make much sense. Qubes uses lightweight VMs to create security qubes (e.g., “work,” “personal,” and “banking,”). A typical user would likely need around five qubes. Very paranoid users, or those who are high-profile targets, might use a dozen or more qubes.

### Why does Qubes use Xen instead of KVM or some other hypervisor?

In short: we believe the Xen architecture allows for the creation of more secure systems (i.e. with a much smaller TCB, which translates to a smaller attack surface). We discuss this in much greater depth in our [Architecture Specification document](#).

### How is Qubes affected by Xen Security Advisories (XSAs)?

See the [XSA Tracker](#).

### What about this other/new (micro)kernel/hypervisor?

Whenever starting a discussion about another (micro)kernel or hypervisor in relation to Qubes, we strongly suggest including answers to the following questions first:

1. What kinds of containers does it use for isolation? Processes? PV VMs? Fully virtualized VMs (HVMs)? And what underlying h/w technology is used (ring0/3, VT-x)?
2. Does it require specially written/built applications (e.g. patched Firefox)?
3. Does it require custom drivers, or can it use Linux/Windows ones?
4. Does it support VT-d, and does it allow for the creation of untrusted driver domains?
5. Does it support S3 sleep?
6. Does it work on multiple CPUs/Chipsets?
7. What are the performance costs, more or less? (e.g. “XYZ prevents concurrent execution of two domains/processes on shared cores of a single processor”, etc.)
8. Other special features? E.g. eliminates cooperative covert channels between VMs?

Here are the answers for Xen 4.1 (which we use as of 2014-04-28):

1. PV and HVM Virtual Machines (ring0/3 for PV domains, VT-x/AMD-v for HVMs).

2. Runs unmodified usermode apps (binaries).
3. Runs unmodified Linux drivers (dom0 and driver domains). PV VMs require special written pvdrivers.
4. Full VT-d support including untrusted driver domains.
5. S3 sleep supported well.
6. Works on most modern CPUs/Chipsets.
7. Biggest performance hit on disk operations (especially in Qubes when complex 2-layer mapping used for Linux qubes). No GPU virtualization.
8. Mostly Works™ :)

### Which virtualization modes do VMs use?

Here is an overview of the VM virtualization modes:

VM type	Mode
Default VMs without PCI devices (most VMs)	PVH
Default VMs with PCI devices	HVM
Stub domains - Default VMs w/o PCI devices	N/A
Stub domains - Default VMs w/ PCI devices	PV
Stub domains - HVMs	PV

### What's so special about Qubes' GUI virtualization?

We have designed the GUI virtualization subsystem with two primary goals: security and performance. Our GUI infrastructure introduces only about 2,500 lines of C code (LOC) into the privileged domain (Dom0), which is very little, and thus leaves little space for bugs and potential attacks. At the same time, due to the smart use of Xen shared memory, our GUI implementation is very efficient, so most virtualized applications really feel as if they were executed natively.

### Why passwordless sudo?

Please refer to [this page](#).

### Why is dom0 so old?

Please see:

- [Installing and updating software in dom0](#)
- [Note on dom0 and EOL](#)

## **Do you recommend coreboot as an alternative to vendor BIOS?**

Yes, where it is possible to use it an open source boot firmware ought to be more trustable than a closed source implementation. [coreboot](#) is as a result a requirement for [Qubes Certified Hardware](#). The number of machines coreboot currently supports is limited and the use of some vendor supplied blobs is generally still required. Where coreboot does support your machine and is not already installed, you will generally need additional hardware to flash it. Please see the [coreboot website](#) / their IRC channel for further information.

## **How should I report documentation issues?**

If you can fix the problem yourself, please see how to [edit the documentation](#). If not, please see [issue tracking](#).

## **Will Qubes seek to get certified under the GNU Free System Distribution Guidelines (GNU FSDG)?**

We wish we could, but the unfortunate reality right now is that an operating system *cannot be secure* without a certain minimum number of proprietary closed-source “blobs” (e.g., CPU microcode updates). A 100% free operating system that excludes all such blobs is vulnerable to known exploits and is therefore unsuitable for any use case where security matters.

Instead, Qubes aims to be as free as possible *without sacrificing security*. All of the code created by the Qubes OS Project itself is 100% free. However, in order for users to actually run that code securely on their hardware, we must pair it with a small number of non-free blobs, which disqualifies Qubes, [along with the vast majority of open-source Linux distributions](#), from GNU FSDG certification.

The [four essential freedoms](#) are part of the core of our philosophy, but so is security. Together, they inform our decisions and motivate our actions. Qubes aims to maximize both security and software freedom to the extent that they are compatible in the world today.

Also see [Is Qubes OS free and open-source software?](#) and the [Qubes OS software license](#).

## **Should I trust this website?**

This website is hosted on [GitHub Pages](#) ([why?](#)). Therefore, it is largely outside of our control. We don’t consider this a problem, however, since we explicitly [distrust the infrastructure](#). For this reason, we don’t think that anyone should place undue trust in the live version of this site on the Web. Instead, if you want to obtain your own trustworthy copy of this website in a secure way, you should clone our [website repo](#), [verify the PGP signatures on the commits and/or tags](#) signed by the [doc-signing keys](#) (which indicates that the content has undergone review), then either [render the site on your local machine](#) or simply read the source, the vast majority of which was intentionally written in Markdown so as to be readable as plain text for this very reason. We’ve gone to special effort to set all of this up so that no one has to trust the infrastructure and so that the contents of this website are maximally available and accessible.

## **What does it mean to “distrust the infrastructure”?**

A core tenet of the Qubes philosophy is “distrust the infrastructure,” where “the infrastructure” refers to things like hosting providers, CDNs, DNS services, package repositories, email servers, PGP key servers, etc. As a project, we focus on securing endpoints instead of attempting to secure “the middle” (i.e., the infrastructure), since one of our primary goals is to free users from being forced to entrust their security to unknown third parties. Instead, our aim is for users to be required to trust as few entities as possible (ideally, only themselves and any known persons whom they voluntarily decide to trust).

Users can never fully control all the infrastructure they rely upon, and they can never fully trust all the entities who do control it. Therefore, we believe the best solution is not to attempt to make the infrastructure trustworthy, but instead to concentrate on solutions that obviate the need to do so. We believe that many attempts to make the infrastructure

appear trustworthy actually provide only the illusion of security and are ultimately a disservice to real users. Since we don't want to encourage or endorse this, we make our distrust of the infrastructure explicit.

Also see: *Should I trust this website?*

### Why do you use GitHub?

Three main reasons:

1. We *distrust the infrastructure* including GitHub (though there are aspects we're still *working on*).
2. It's free (as in beer). We'd have to spend either time or money to implement a solution ourselves or pay someone to do so, and we can't spare either one right now.
3. It has low admin/overhead requirements, which is very important, given how little time we have to spare.

Also see: *Should I trust this website?*

### Why doesn't this website have security feature X?

Although we caution users against *placing undue trust in this website* because we *distrust the infrastructure*, we have no objection to enabling website security features when doing so is relatively costless and provides some marginal benefit to website visitors. So, if feature X isn't enabled, it's most likely for one of three reasons:

1. Our GitHub Pages platform doesn't support it.
2. Our platform supports it, but we've decided not to enable it.
3. Our platform supports it, but we're not aware that we can enable it or have forgotten to do so.

If it seems like a feature that we can and should enable, please *let us know*!

### Why do the mailing lists require a Google account?

They don't. This is a common misconception. The mailing lists have never required a Google account. It has always been possible to use them purely via email (see the *mailing lists* section for instructions).

A lot of people probably see that the mailing lists use Google Groups and just assume that a Google account must be required, but it's not true. Google Groups is simply used for the infrastructure. Of course, you *can* use the web interface with a Google account, but there are many people in the Qubes community who participate on the mailing lists without one.

### Why do you use Google Groups for the mailing lists?

For the same general reasons as listed in *FAQ: Why do you use GitHub?*

## 1.5.2 Users

### Can I watch YouTube videos in qubes?

Absolutely.

### Can I run applications, like games, which require hardware acceleration?

Those won't fly. We do not provide GPU virtualization for Qubes. This is mostly a security decision, as implementing such a feature would most likely introduce a great deal of complexity into the GUI virtualization infrastructure. However, Qubes does allow for the use of accelerated graphics (e.g. OpenGL) in dom0's Window Manager, so all the fancy desktop effects should still work. App qubes use a software-only (CPU-based) implementation of OpenGL, which may be good enough for basic games and applications.

For further discussion about the potential for GPU passthrough on Xen/Qubes, please see the following threads:

- [GPU passing to HVM](#)
- [Clarifications on GPU security](#)

### Is Qubes a multi-user system?

No. Qubes does not pretend to be a multi-user system. Qubes assumes that the user who controls Dom0 controls the whole system. It is very difficult to **securely** implement multi-user support. See [here](#) for details.

However, in Qubes 4.x we will be implementing management functionality. See [Admin API](#) and [Core Stack](#) for more details.

### What are the system requirements for Qubes OS?

See the *system requirements*.

### Is there a list of hardware that is compatible with Qubes OS?

See the Hardware Compatibility List.

### Is there any certified hardware for Qubes OS?

See *Certified Hardware*.

### How much disk space does each qube require?

Each qube is created from a template and shares the root filesystem with this template (in a read-only manner). This means that each qube needs only as much disk space as is necessary to store its own private data. This also means that it is possible to update the software for several qubes simultaneously by running a single update process in the template upon which those qubes are based. (These qubes will then have to be restarted in order for the update to take effect in them.)



## How much memory is recommended for Qubes?

Please see the *system requirements*.

## Can I install Qubes on a system without VT-x/AMD-V or VT-d/AMD-Vi/AMD IOMMU?

Please see the *system requirements* for the latest information. If you are receiving an error message on install saying your “hardware lacks the features required to proceed”, check to make sure the virtualization options are enabled in your BIOS/UEFI configuration. You may be able to install without the required CPU features for testing purposes only, but VMs (in particular, sys-net) may not function correctly and there will be no security isolation. For more information, see *Qubes-certified hardware*.

## Why is VT-x/AMD-V important?

By default, Qubes uses Xen’s PVH and HVM virtualization modes, which require VT-x/AMD-V. This means that, without VT-x/AMD-V, no VMs will start in a default Qubes installation. In addition, if your system lacks VT-x/AMD-V, then it also lacks VT-d/AMD-Vi/AMD IOMMU. (See next question.)

## Why is VT-d/AMD-Vi/AMD IOMMU important?

On a system without VT-d/AMD-Vi/AMD IOMMU, there will be no real security benefit to having a separate NetVM, as an attacker could always use a simple *DMA attack* to go from the NetVM to Dom0. Nonetheless, all of Qubes’ other security mechanisms, such as qube separation, work without VT-d/AMD-Vi/AMD IOMMU. Therefore, a system running Qubes without VT-d/AMD-Vi/AMD IOMMU would still be significantly more secure than one running Windows, Mac, or Linux.

## What is a DMA attack?

Direct Memory Access (DMA) is mechanism for PCI devices to access system memory (read/write). Without VT-d/AMD-Vi/AMD IOMMU, any PCI device can access all the memory, regardless of the VM to which it is assigned (or if it is left in dom0). Most PCI devices allow the driver to request an arbitrary DMA operation (like “put received network packets at this address in memory”, or “get this memory area and send it to the network”). So, without VT-d/AMD-Vi/AMD IOMMU, it gives unlimited access to the whole system. Now, it is only a matter of knowing where to read/write to take over the system, instead of just crashing. But since you can read the whole memory, it isn’t that hard.

Now, how does this apply to Qubes OS? The above attack requires access to a PCI device, which means that it can be performed only from the NetVM or USB VM, so someone must first break into one of those VMs. But this isn’t that hard, because there is a lot of complex code handling network traffic. There is a history of bugs in DHCP clients, DNS clients, etc. Most attacks on the NetVM and USB VM (but not all of them!) require being somewhat close to the target system, for example, being connected to the same Wi-Fi network, or in the case of a USB VM, having physical access to a USB port.

### Can I use AMD-v instead of VT-x?

Yes, and see [this message](#).

### Can I install Qubes in a virtual machine (e.g., on VMware)?

Some users have been able to do this, but it is neither recommended nor supported. Qubes should be installed bare-metal. (After all, it uses its own bare-metal hypervisor!)

### How many qubes should I have? What's a good way to organize them?

*How to organize your qubes* walks through several examples of how different types of users can set up their Qubes OS system to support their unique use cases.

### What is a terminal?

A [terminal emulator](#), nowadays often referred to as just a *terminal*, is a program which provides a text window. Inside that window, a [shell](#) is typically running in it. A shell provides a [command-line interface](#) where the user can enter and run [commands](#).

See introductions on Wikibooks: [here](#), [here](#) and [here](#).

### Why does my network adapter not work?

You may have an adapter (wired, wireless), that is not compatible with open-source drivers shipped by Qubes. You may need to install a binary blob, which provides drivers, from the [linux-firmware](#) package.

Open a terminal and run `sudo dnf install linux-firmware` in the template upon which your NetVM is based. You have to restart the NetVM after the template has been shut down.

### Can I install Qubes OS together with other operating system (dual-boot/multi-boot)?

You shouldn't do that, because it poses a security risk for your Qubes OS installation. But if you understand the risk and accept it, read [documentation on multibooting](#). It begins with an explanation of the risks with such a setup.

### Which version of Qubes am I running?

See [here](#).

### My qubes lost internet access after a template update. What should I do?

See [Update Troubleshooting](#).

**My keyboard layout settings are not behaving correctly. What should I do?**

See *Hardware Troubleshooting*.

**My dom0 and/or template update stalls when attempting to update via the GUI tool. What should I do?**

This can usually be fixed by updating via the command line.

In dom0, open a terminal and run `sudo qubes-dom0-update`.

In your templates, open a terminal and run `sudo dnf upgrade`.

**How do I run a Windows HVM in non-seamless mode (i.e., as a single window)?**

Enable “debug mode” in the qube’s settings, either by checking the box labeled “Run in debug mode” in the Qubes VM Manager qube settings menu or by running the `qvm-prefs` command.

**I created a USB VM and assigned USB controllers to it. Now the USB VM won’t boot.**

This is probably because one of the controllers does not support reset. See the *USB Troubleshooting guide*.

**I assigned a PCI device to a qube, then unassigned it/shut down the qube. Why isn’t the device available in dom0?**

This is an intended feature. A device which was previously assigned to a less trusted qube could attack dom0 if it were automatically reassigned there. In order to re-enable the device in dom0, either:

- Reboot the physical machine.

or

- Go to the sysfs (`/sys/bus/pci`), find the right device, detach it from the `pciback` driver and attach back to the original driver. Replace `<BDF>` with your device, for example `00:1c.2`:

```
echo 0000:<BDF> > /sys/bus/pci/drivers/pciback/unbind
MODALIAS=`cat /sys/bus/pci/devices/0000:<BDF>/modalias`
MOD=`modprobe -R $MODALIAS | head -n 1`
echo 0000:<BDF> > /sys/bus/pci/drivers/$MOD/bind
```

See also *here*.

**How do I play video files?**

If you’re having trouble playing a video file in a qube, you’re probably missing the required codecs. The easiest way to resolve this is to install VLC Media Player and use that to play your video files. You can do this in multiple different template distros (Fedora, Debian, etc.).

For Debian:

1. (Recommended) Clone an existing Debian template
2. Install VLC in that template:

```
$ sudo apt install vlc
```

3. Use VLC to play your video files.

For Fedora:

1. (Recommended) Clone an existing Fedora template
2. *Enable the appropriate RPMFusion repos in the desired Fedora template.*
3. Install VLC in that template:

```
$ sudo dnf install vlc
```

4. Use VLC to play your video files.

### How do I access my external drive?

The recommended approach is to pass only the specific partition you intend to use from *sys-usb* to another qube via *qvm-block*. They will show up in the destination qube as */dev/xvd\** and must be mounted manually. Another approach is to attach the entire USB drive to your destination qube. However, this could theoretically lead to an attack because it forces the destination qube to parse the device's partition table. If you believe your device is safe, you may proceed to attach it.

In Qubes 4.0, this is accomplished with the Devices Widget located in the tool tray (default top right corner, look for an icon with a yellow square). From the top part of the list, click on the drive you want to attach, then select the qube to attach it to. Although you can also attach the entire USB device to a qube by selecting it from the bottom part of the list, in general this approach should not be used because you are exposing the target qube to unnecessary additional attack surface.

Although external media such as external hard drives or flash drives plugged in via USB are available in the USB qube, it is not recommended to access them directly from inside the USB qube. See *Block (Storage) Devices* for more information.

### My encrypted drive doesn't appear in Debian qube.

This is an issue that affects qubes based on Debian Jessie. The problem is fixed in Stretch, and does not affect Fedora-based qubes.

A mixed drive with some encrypted partitions appears correctly in Nautilus. The encrypted partitions are identified and the user is prompted for password on attempting to mount the partition.

A fully encrypted drive does not appear in Nautilus.

The workaround is to manually decrypt and mount the drive:

1. Attach USB device to qube - it should be attached as */dev/xvdi* or similar.
2. `sudo cryptsetup open /dev/xvdi bk --type luks`
3. `sudo cryptsetup status /dev/mapper/bk` (Shows useful status info.)
4. `sudo mount /dev/mapper/bk /mnt`

The decrypted device is now available at */mnt* - when you have finished using it unmount and close the drive.

1. `sudo umount /mnt`
2. `sudo cryptsetup close bk --type luks`
3. Remove USB from qube.

**Windows Update is stuck.**

This has nothing to do with Qubes. It's a longstanding Windows bug.

**Fullscreen Firefox is frozen.**

Press F11 twice.

**I have weird graphics glitches like the screen turning partially black.**

If it seems like the issue described in [this thread](#), try disabling the window compositor:

- Q → System Tools → Window Manager Tweaks → Compositor → uncheck “Enable display compositing”

Please report (via the mailing lists) if you experience this issue, and whether disabling the compositor fixes it for you or not.

**My HVM in Qubes R4.0 won't let me start/install an OS**

I see a screen popup with SeaBios and 4 lines, last one being Probing EDD (edd=off to disable!... ok.

From a dom0 prompt, enter:

```
qvm-prefs <HVMname> kernel ""
```

**When I try to install a template, it says no match is found.**

See *VM Troubleshooting*.

**I keep getting “Failed to synchronize cache for repo” errors when trying to update my Fedora templates**

See *Update Troubleshooting*.

**I see a “Failed to start Load Kernel Modules” message on boot**

The full message looks like:

```
[FAILED] Failed to start Load Kernel Modules.
See 'systemctl status systemd-modules-load.service' for details.
```

This is cosmetic only, and can safely be ignored.

### Why is Qubes so slow and how can I make it faster?

During boot, Qubes starts several virtual machines. Having so many qubes running at once inevitably strains the resources of your computer and causes slowness. The most effective way to speed up Qubes is to get more powerful hardware – a fast CPU, a lot of memory and fast SSDs. Qubes is slower when reading from the disk because of the VM overhead, which is why we recommend installing it on a fast SSD.

### Could you please make my preference the default?

It would be great if Qubes were configured just the way we like it by default with all of our favorite programs and settings. Then, we could just install Qubes without having to install any programs in it or adjust any settings. We might even think that if a particular program or setting works so well for *us*, it would work well for *everyone*, so we'd actually be doing everyone a favor! The problem is that Qubes has *tens of thousands of different users* with radically different needs and purposes. There is no particular configuration that will be ideal for everyone (despite how much we might feel that our preference would be better for everyone), so the best we can do is to put power in the hands of users to configure their Qubes installations the way they like (subject to security constraints, of course). For this reason, we generally do not grant requests for people's favorite programs to be installed by default or for some setting that obviously varies by user preference to be changed so that it matches the requester's preference.

See also: *What is Qubes' attitude toward changing guest distros?*

### Software installed in a qube is gone after restarting. Why?

Software must be *installed in the template* on which your qube is based.

## 1.5.3 Developers

### Are there restrictions on the software that the Qubes developers are willing to use?

Yes. In general, the Qubes developers will not use a piece of software unless there is an *easy* way to verify both its **integrity** and **authenticity**, preferably via PGP signatures (see *Verifying Signatures*). Specifically:

- If PGP signatures are used, the signing key(s) should have well-publicized fingerprint(s) verifiable via multiple independent channels or be accessible to the developers through a web of trust.
- If the software is security-sensitive and requires communication with the outside world, a “split” implementation is highly preferred (for examples, see *Split GPG* and *Split Bitcoin*).
- If the software has dependencies, these should be packaged and available in repos for a *current, Qubes-supported version* of Fedora (preferred) or Debian (unless all the insecure dependencies can run in an untrusted VM in a “split” implementation).
- If the software must be built from source, the source code and any builders must be signed. (Practically speaking, the more cumbersome and time-consuming it is to build from source, the less likely the developers are to use it.)

## Why does dom0 need to be 64-bit?

Since 2013 [Xen has not supported 32-bit x86 architecture](#) and Intel VT-d, which Qubes uses to isolate devices and drivers, is available on Intel 64-bit processors only.

In addition, with features like improved ASLR, it is often more difficult to exploit a bug on x64 Linux than x86 Linux. While we designed Qubes from the beginning to limit potential attack vectors, we still realize that some of the code running in Dom0, e.g. our GUI daemon or xen-store daemon, however simple, might contain some bugs. Plus since we haven't implemented a separate storage domain, the disk backends are in Dom0 and are "reachable" from the VMs, which adds up to the potential attack surface. So, having faced a choice between 32-bit and 64-bit OS for Dom0, it was almost a no-brainer. The 64-bit option provides some (little perhaps, but some) more protection against some classes of attacks, and at the same time does not have any disadvantages except the extra requirement of a 64 bit processor. And even though Qubes now "needs" a 64 bit processor, it didn't make sense to run Qubes on a system without 3-4GB of memory, and those have 64-bit CPUs anyway.

## What is the recommended build environment for Qubes OS?

Any rpm-based, 64-bit environment, the preferred OS being Fedora.

## How do I build Qubes from sources?

See [these instructions](#).

## How do I submit a patch?

See the [Qubes Source Code Repositories](#) article.

## What is Qubes' attitude toward changing guest distros?

In general, we try to respect each distro's culture, but we reserve the right to make modifications that we deem appropriate. See the discussion on issue [#1014](#) for an example.

The policy is there mostly to ease maintenance, on several levels:

- Less modifications means easier migration to new upstream distribution releases.
- The upstream documentation matches the distribution running in the Qubes VM.
- We're less likely to introduce Qubes-specific issues.
- Each officially supported distribution (ideally) should offer the same set of Qubes-specific features - a change in one supported distribution should be followed also in others, including new future distributions.

## Why don't you fix upstream bugs that affect Qubes OS?

First, a bit of background in case you're new to the open-source software world: There are a huge number of different open-source projects that each focus on the software they create and maintain. Some focus on specific frameworks, libraries, and background subsystems that most users never see. Others focus on specific tools and apps that use these frameworks, libraries, and background subsystems. Still others focus on combining many different tools and apps. And some, like Qubes OS, are entire operating systems that include all kinds of other software. When one piece of software uses a different piece of software, the piece of software being used is said to be "upstream," while the piece of software using it said to be "downstream." For example, Qubes OS uses the Xen hypervisor, so Xen is upstream relative

to Qubes, and Qubes is downstream relative to Xen (and likewise for the respective project that creates and maintains each piece of software).

Many open-source operating systems, including Qubes OS, are transparent about the fact that they are “compilations” of upstream software. By contrast, proprietary, commercial operating systems like Windows and macOS tend to either obscure this fact or avoid using upstream software in favor of doing everything in-house, because they have the huge workforce and commercial revenue that allows them to do so. If you’re accustomed to using a proprietary, commercial operating system, then you may need some time to get used to the fact that Qubes OS is a compilation of many different pieces of open-source software.

Now, let’s get to the original question: Why don’t we fix upstream bugs that affect Qubes OS? This question can come up in different ways. For example, many people, especially those who aren’t familiar with how open-source software development works, wonder why we sometimes close *issues* as “not our bug.” Don’t we care about the Qubes users who are affected by these bugs? Are we really so cold and heartless?

On the contrary, it is precisely because we care so much about Qubes users that we do this. It’s important to understand that Qubes OS combines different pieces of software from a very large number of upstream projects (especially since it includes entire separate OSes inside of itself) and that many of these projects have much larger workforces and much more funding than we do. They are better equipped to fix bugs in their own software. Not only are they the ones who wrote the code, they’re also more familiar with how best to integrate any fixes into the entire code base for maintainability. Besides, they own the code. We can’t force any other project to accept a patch, even if we sincerely believe it’s a good bug fix. In some cases, we have to maintain our own fork of an upstream project, which just adds to our ongoing maintenance burden.

In contrast to some of the large upstream projects whose software we use, the Qubes OS Project is small, lean, and focused on one goal: creating and maintaining a reasonably secure operating system for regular desktop users. The Qubes core developers are specialists. They are among the best in the world at virtualization security, low-level system security, and implementing security-by-compartmentalization at the operating-system level. There are many aspects of Qubes OS engineering work for which they are uniquely qualified. Recognizing this, it only makes sense to focus their time where it will provide the greatest benefit, on doing security-related work that only they can do. By contrast, it would be a wasteful misallocation of skill and talent (to the long-term detriment of Qubes users) to have them fixing bugs that are in code they didn’t write, that doesn’t belong to them, that (in some cases) belongs to a huge upstream project with ample time and resources, and that the upstream project is equally capable of fixing (and, in many cases, is *better* suited to fix, as that’s *their* area of specialization).

Moreover, the question is based on a faulty assumption in the first place, because we already *do* in fact fix some upstream bugs that affect Qubes OS. For example, the Qubes core developers have made significant upstream Xen contributions, particularly in the area of security, as that’s where our developers specialize. So, the original question should really be rephrased to ask, “Why don’t you fix *every* upstream bug that affects Qubes OS?” In light of the foregoing explanation, we hope you agree that this would be an unreasonable expectation.

“Very well,” you might be thinking, “but there’s still an upstream bug that affects me! What can I do about it?” Recall what we discussed above about how the open-source world works. If there’s a bug in some piece of upstream software, then there’s an open-source project responsible for creating and maintaining that software. They’re the ones who wrote the code and who are best equipped to fix the bug. You should file a bug report in *that* project’s issue tracker instead. Not only will you be helping all other affected Qubes users, you’ll also be helping *all* downstream users of that software!

(Note: If you’re wondering about cases in which a bug has already been fixed upstream but hasn’t yet arrived in your Qubes OS release, please see *backports*. These are *not* cases in which an issue is closed as “not our bug.”)



### Is the I/O emulation component (QEMU) part of the Trusted Computing Base (TCB)?

No. Unlike many other virtualization systems, Qubes takes special effort to keep QEMU *outside* of the TCB. This has been achieved thanks to the careful use of Xen's stub domain feature. For more details about how we improved on Xen's native stub domain use, see [here](#).

### Is Secure Boot supported?

UEFI Secure Boot is not supported out of the box as UEFI support in Xen is very basic. Arguably secure boot reliance on UEFI integrity is not the best design. The relevant binaries (shim.efi, xen.efi, kernel / initramfs) are not signed by the Qubes Team and secure boot has not been tested. Intel TXT (used in *Anti Evil Maid*) at least tries to avoid or limit trust in BIOS. See the Heads project [1] [2] for a better-designed non-UEFI-based secure boot scheme with very good support for Qubes.

### What is the canonical way to detect Qubes VM?

Check `/usr/share/qubes/marker-vm` file existence. Additionally, its last line contains Qubes release version (e.g., 4.0). The file was introduced after the initial Qubes 4.0 release. If you need to support not-fully-updated systems, check for the existence of `/usr/bin/qrexec-client-vm`.

### Is there a way to automate tasks for continuous integration or DevOps?

Yes, Qubes natively supports automation via *Salt (SaltStack)*. There is also the unofficial *ansible-qubes* toolkit. (**Warning:** Since this is an external project that has not been reviewed or endorsed by the Qubes team, allowing it to manage `dom0` may be a security risk.)

## 1.6 Issue tracking

We use *GitHub Issues* as our issue tracking system. All issues pertaining to the Qubes OS Project (including auxiliary infrastructure such as this website) are tracked in *qubes-issues*.

### 1.6.1 How to open a new issue

First, let's make sure the issue tracker is the right place.

#### I need help, have a question, or want to discuss something.

We're happy to help, answer questions, and have discussions, but the issue tracker is not the right place for these activities. Instead, please see *Help, Support, Mailing Lists, and Forum*.

### I see something that should be changed in the documentation.

We encourage you to submit the change yourself! Please see the [how to edit the documentation](#) for instructions on how to do so. If it's something you can't do yourself, please proceed to open an issue.

### I would like to report a security vulnerability.

Thank you! If the vulnerability is confidential, please do not report it in our public issue tracker. Instead, please see *Reporting Security Issues in Qubes OS*.

### I still want to open an issue.

Great! Thank you for taking the time and effort to help improve Qubes! To ensure the process is efficient and productive for everyone involved, please follow these steps:

1. Carefully read our issue tracking [guidelines](#). If your issue would violate any of the guidelines, **stop**. Please do not submit it.
2. [Search through the existing issues](#), both open and closed, to see if your issue already exists. If it does, **stop**. *Do not open a duplicate*. Instead, comment on the existing issue.
3. Go [here](#).
4. Select the *type* of issue you want to open.
5. Enter a descriptive title.
6. Do not delete the provided issue template. Fill out every applicable section.
7. Make sure to mention any relevant documentation and other issues you've already seen. We don't know what you've seen unless you tell us. If you don't list it, we'll assume you haven't seen it.
8. If any sections of the issue template are *truly* not applicable, you may remove them.
9. Submit your issue.
10. Respond to any questions the official team asks. For example, you may be asked to provide specific logs or other additional information.

Eventually, your issue may be closed. See [how issues get closed](#) for details about when, why, and how this occurs.

## 1.6.2 Labels and projects

Labels and projects are features of GitHub's issue tracking system that we use to keep [qubes-issues](#) organized.

### Labels

When an issue is first created, certain [labels](#) may automatically be applied to it based on the type of issue the reporter selected. For example, if someone selects the "Bug report" template, then the T: bug label will automatically be applied to that issue. After that, only Qubes team members have permission to modify labels. Many labels have descriptions on them that can be viewed by hovering over them or on the [list of labels](#). Let's go over some of the most important ones.

## Type

There are three issue **types**: T: bug, T: enhancement, and T: task.

- T: bug — Type: bug report. A problem or defect resulting in unintended behavior in something that exists.
- T: enhancement — Type: enhancement. A new feature that does not yet exist **or** improvement of existing functionality.
- T: task — Type: task. An action item that is neither a bug nor an enhancement.

Every open issue should have **exactly one** type. An open issue should not have more than one type, and it should not lack a type entirely. Bug reports are for problems in things that already exist. If something doesn't exist yet, but you think it ought to exist, then use T: enhancement instead. If something already exists, but you think it could be improved in some way, you should again use T: enhancement. T: task is for issues that fall under neither T: bug nor T: enhancement.

## Priority

There are several issue **priority** levels ranging from P: minor to P: blocker (see [here](#) for the full list). Every open issue should have **exactly one** priority. An open issue should not have more than one priority, and it should not lack a priority entirely. See [here](#) for details about how the developers use these priorities.

## Component

There are many **component** labels, each beginning with C: (see [here](#) for the full list). Every open issue should have **at least one** component. An open issue may have more than one component, but it should not lack a component entirely. When no other component applies, use C: other.

## Affected release

A label of the form affects-<RELEASE\_NUMBER> indicates that an issue affects the corresponding Qubes OS release. An issue can have more than one of these labels if it affects multiple releases.

## Projects

The issue tracker has several [projects](#). A project is a way to create a group of multiple related issues. This is the preferred method of grouping issues, whereas trying to use normal issues as “meta-issues” or “epics” is discouraged.

### 1.6.3 Search tips

- Search both open and closed issues. For example, you may be experiencing a bug that was just fixed, in which case the report for that bug is probably closed. In this case, it would be useful to view [all bug reports, both open and closed](#), with the most recently updated sorted to the top.
- Search with labels. For example, you can search issues by priority (blocker, critical, major, etc.) and by component (core, manager/widget, Xen, etc.).
- Search by closure reason: `reason:completed` and `reason:"not planned"`.
- Search by project.

## 1.6.4 Guidelines

### The issue tracker is not a discussion forum

The issue tracker is a tool to help the developers be more productive and efficient in their work. It is not a place for discussion. If you wish to discuss something in the issue tracker, please do so on the forum or mailing lists (see [Help, Support, Mailing Lists, and Forum](#)). You can simply link to the relevant issue in your discussion post.

This guideline is important for keeping issues focused on *actionable information*, which helps the developers to stay focused on their work. When developers come back to an issue to work on it, we do not want them to have to sift through a large number of unnecessary comments before they can get started. In many cases, an issue that gets “too big” essentially becomes more trouble than it’s worth, and no developer will touch it (also see *every issue must be about a single, actionable thing*). In these cases, we sometimes have to close the issue and open a new one. This is a waste of energy for everyone involved, so we ask that everyone help to avoid repeating this pattern.

### Do not submit questions

[qubes-issues](#) is not the place to ask questions. This includes, but is not limited to, troubleshooting questions and questions about how to do things with Qubes. Instead, see [Help, Support, Mailing Lists, and Forum](#) for appropriate places to ask questions. By contrast, [qubes-issues](#) is meant for tracking more general bugs, enhancements, and tasks that affect a broad range of Qubes users.

### Use the issue template

When you open a new issue, an issue template is provided for you. Please use it. Do not delete it. The issue template is carefully designed to elicit important information. Without this information, the issue is likely to be incomplete. (If certain sections are not applicable, you may remove them, but please do so only sparingly and only if they are *truly* not applicable.)

It is also important to note the placement and content of the HTML comments in the issue template. These help us to have issues with a consistent format.

### Every issue must be about a single, actionable thing

If your issue is not actionable, please see [Help, Support, Mailing Lists, and Forum](#) for the appropriate place to post it. If your issue would be about more than one thing, file them as separate issues instead. This means we should generally not try to use a single issue as a “meta” or “epic” issue that exists only to group, contain, or track other issues. Instead, when there is a need to group multiple related issues together, use [projects](#).

This guideline is extremely important for making the issue tracker a useful tool for the developers. When an issue is too big and composite, it becomes intractable and drastically increases the likelihood that nothing will get done. Such issues also tend to encourage an excessive amount of general discussion that is simply not appropriate for a technical issue tracker (see *the issue tracker is not a discussion forum*).

### New issues should not be duplicates of existing issues

Before you submit an issue, check to see whether it has already been reported. Search through the existing issues – both open and closed – by typing your key words in the **Filters** box. If you find an issue that seems to be similar to yours, read through it. If you find an issue that is the same as or subsumes yours, leave a comment on the existing issue rather than filing a new one, even if the existing issue is closed. If an issue affects more than one Qubes version, we usually keep only one issue for all versions. The Qubes team will see your comment and reopen the issue, if appropriate. For example, you can leave a comment with additional information to help the maintainer debug it. Adding a comment will subscribe you to email notifications, which can be helpful in getting important updates regarding the issue. If you don't have anything to add but still want to receive email updates, you can click the “Subscribe” button at the side or bottom of the comments.

### Every issue must be of a single type

Every issue must be exactly one of the following types: a bug report (bug), a feature or improvement request (enhancement), or a task (task). Do not file multi-typed issues. Instead, file multiple issues of distinct types. The Qubes team will classify your issue according to its type.

### New issues should include all relevant information

When you file a new issue, you should be sure to include the version of Qubes you're using, as well as versions of related software packages (*how to copy information out of dom0*). If your issue is related to hardware, provide as many details as possible about the hardware. A great way to do this is by *generating and submitting a Hardware Compatibility List (HCL) report*, then linking to it in your issue. You may also need to use command-line tools such as `lspci`. If you're reporting a bug in a package that is in a *testing* repository, please reference the appropriate issue in the *updates-status* repository. Project maintainers really appreciate thorough explanations. It usually helps them address the problem more quickly, so everyone wins!

### There are no guarantees that your issue will be addressed

Keep in mind that *qubes-issues* is an issue tracker, not a support system. Creating a new issue is simply a way for you to submit an item for the Qubes team's consideration. It is up to the Qubes team to decide whether or how to address your issue, which may include closing the issue without taking any action on it. Even if your issue is kept open, however, you should not expect it to be addressed within any particular time frame, or at all. At the time of this writing, there are well over one thousand open issues in *qubes-issues*. The Qubes team has its own roadmap and priorities, which will govern the manner and order in which open issues are addressed.

### Issues and comments must be written in English

If English is not your native language, you may post a machine translation. If you wish, you may also include the original non-English text in a *collapsible section*.

### Use collapsible sections for long, nonessential content

On GitHub, create collapsible sections in Markdown like so:

```
<details>
<summary>Summary goes here. This line is optional.</summary>

Long, nonessential content goes here. You can put a code block here, but make sure to
↳ leave empty lines before and after the fence lines (```).

</details>
```

**Tip:** Use the “Preview” tab to make sure it renders correctly before posting.

### 1.6.5 How issues get closed

If the Qubes developers make a code change that resolves an issue, then the issue will typically be **closed from the relevant commit or merged pull request (PR)**.

#### Bug reports

In the case of bugs, the package containing the change will move to the appropriate *testing* repository, then to the appropriate stable repository. If you so choose, you can test the fix while it’s in the *testing* repository, or you can wait for it to land in the stable repository. If, after testing the fix, you find that it does not really fix the reported bug, please leave a comment on the issue explaining the situation. When you do, we will receive a notification and respond on the issue or reopen it (or both). Please **do not** create a duplicate issue or attempt to contact the developers individually about a problem.

#### Resolution

In GitHub, an issue can be *closed as either ‘‘completed’ or not planned*.

Being closed as **completed** means that the issue has been fixed (in the case of bugs) or done (in the case of enhancements and tasks). More precisely, it means that a commit containing the relevant work has been pushed. It takes time for this work to make its way into a package, which must then go through the *testing* process before finally landing in the relevant stable repository. Automated comments on the issue will announce when key events in this process occur.

Being closed as **not planned** means that the issue will *not* be fixed (in the case of bugs) or done (in the case of enhancements and tasks). When an issue is closed as **not planned**, we add a **resolution** label starting with R: that specifies the reason for the closure, such as R: **duplicate** or R: **cannot reproduce**. Each of these labels has a description that briefly explains the label. We also leave a comment containing a longer explanation for why the issue is being closed along with general information.

While issues that are closed as **not planned** get a more specific resolution label, issues that are closed as **completed** do not always get one, since the linked PRs, commits, automated messages, and the **completed** reason itself are often sufficient to convey all relevant information. For information about using closure reasons in searches, see *Search tips*.

## Backports

Issues in GitHub can only be open or closed, but when it comes to bugs that affect multiple versions of Qubes OS, there are several possible states:

1. Not fixed yet
2. Fix developed but not yet committed (PR open)
3. Fix committed (PR merged), but update not yet pushed to any repo
4. Update pushed to testing repo for the most recent development version
5. Update pushed to stable repo for the most recent development version
6. Update backported to stable version(s) and pushed to the testing repo
7. Update pushed to stable repo of stable version(s)

We close issues at step 3. Then, as updates are released, the issue automatically gets the appropriate `current-testing` (`rX.Y-*cur-test`) and `stable` (`rX.Y-*stable`) labels. Based on these labels, it's possible to select issues waiting for step 6 (see [issues by release](#)).

Therefore, if you see that an issue is closed, but the fix is not yet available to you, be aware that it may be at an intermediate stage of this process between issue closure and the update being available in whichever repos you have enabled in whichever version of Qubes you're using.

In order to assist with this, we have a label called `backport pending`, which means, "The fix has been released for the testing release but is pending backport to the stable release." Our infrastructure will attempt to apply this label automatically, when appropriate, but it is not perfect, and the developers may need to adjust it manually.

## Understanding open and closed issues

Every issue is always in one of two states: open or closed, with open being the default. The **open** and **closed** states mean that, according to our available information at present, the issue in question either **is** or **is not** (respectively) actionable for the Qubes team. The open and closed states do not mean anything more than this, and it's important not to read anything else into them. It's also important to understand that closing an issue is, in effect, nothing more than changing a virtual tag on an issue. Closing an issue is never "final" in any sense, and it does not affect the issue itself in any other way. Issues can be opened and closed instantly with a single button press an unlimited number of times at no cost. In fact, since the open and closed states reflect our available information at present, one should expect these states to change back and forth as new information becomes available. Closed issues are fully searchable, just like open issues, and we explicitly instruct all users of the issue tracker to search *both* open *and* closed issues, which GitHub makes easy.

## 1.7 Help, support, mailing lists, and forum

The Qubes community is here to help! Since Qubes is a security-oriented operating system, we want to make sure you *stay safe* as you get the support you need, and we want to make sure our community remains a friendly and productive place by ensuring we all follow the *Code of Conduct* and *discussion guidelines*.

## 1.7.1 How to get help and support

First, let's see what kind of help you need.

### I have a problem or a question.

No worries! Here's how we recommend proceeding:

1. Check the *documentation*. There may already be a page about it. Specifically, check out the *How-To Guides* and *Troubleshooting* sections.
2. Search the *FAQ*. Your question might already be answered.
3. Try *searching the issue tracker*. There may already be an open **or closed** issue about your problem. The issue tracker is constantly being updated with known bugs and may contain workarounds for problems you're experiencing. If there are any pinned issues at the top, make sure to check them first!
4. Try *searching the Qubes Forum*. There may already be a matching topic.
5. Try *searching the qubes-users archives*. There may have already been a relevant thread.

### I didn't find a solution or an answer!

Sorry to hear that! In that case, we recommend asking for help on the *Qubes Forum* or on the *qubes-users mailing list*. Choose the venue you prefer, but please don't ask on both at the same time! Before you ask, please review our *discussion guidelines* and StackOverflow's advice on *how to ask good questions*. Don't forget to *stay safe*!

### I don't need support, but I think I found a bug.

We'd be grateful if you reported it (but please make sure no one else has already reported it first)! Please see *Issue Tracking* for details.

### I don't need support, but I'd like to request a feature.

No promises, but we'd be happy to consider it! Please see *Issue Tracking* for details.

### Where's the best place to discuss Qubes?

That would be the *Qubes Forum* and the *qubes-users mailing list*. Please have a look at our *discussion guidelines* before diving in. Enjoy!

### How can I get involved and contribute?

Thank you for asking! Please see *How to Contribute* for all the ways you can do so.



## I would like to report a security vulnerability.

That sounds more like you helping us! Thanks! Please see *Reporting Security Issues in Qubes OS*.

### 1.7.2 Staying safe

The Qubes mailing lists and forum are open to the public. The contents are crawled by search engines and archived by third-party services outside of our control. Please do not send or post anything that you are not comfortable seeing discussed in public. If confidentiality is a concern, please use PGP encryption in an off-list email.

The Qubes community includes people from all walks of life and from around the world. Individuals differ in areas of experience and technical expertise. You will come into contact with others whose views and agendas differ from your own. Everyone is free to write what they please, as long as it doesn't violate our *Code of Conduct*. Be friendly and open, but do not believe everything you read. Use good judgment, and be especially careful when following instructions (e.g., copying commands) given by others on the lists.

It's always possible that a bad actor could try to impersonate any member of the *Qubes team* anywhere on the Internet. Please don't assume that someone who claims to be an official Qubes team member really is one without an appropriate form of authentication, such as a *verified PGP-signed message*. (But bear in mind that anyone can generate a key with any name on it and use it to PGP-sign a message, so the mere presence of a PGP signature does not indicate authority. Successful *verification* is what counts.) All official *news* can be authenticated by *verifying the signatures* on the relevant tags or commits in the *qubes-posts* repository.

Given that there may be impostors and others trying to lead you astray, how should you sort the good advice from the bad? This is up to each individual to decide, but it helps to know that many members of our community have proven themselves knowledgeable through their *contributions* to the project. Often, these individuals sign their messages with the same key as (or another key authenticated by) the one they use to *sign their contributions*.

For example, you might find it easier to trust advice from someone who has a proven track record of *contributing software packages* or contributing to the documentation. It's unlikely that individuals who have worked hard to build good reputations for themselves through their contributions over the years would risk giving malicious advice in signed messages to public mailing lists. Since every contribution to the Qubes OS Project is publicly visible and cryptographically signed, anyone would be in a position to *verify* that these came from the same keyholder.

### 1.7.3 Discussion guidelines

Qubes discussions mainly take place on *qubes-users*, *qubes-devel*, and our *forum*, all of which are explained below. Most questions should be directed to *qubes-users* or the *forum*. **Please do not send questions to individual Qubes developers.** By sending a message to the appropriate mailing list, you are not only giving others a chance to help you, but you may also be helping others by starting a public discussion about a shared problem or interest.

These are open venues where people freely come together to discuss Qubes and voluntarily help each other out of mutual interest and good will. They are *not* your personal, paid support service. **No one owes you a reply.** No one here is responsible for solving your problems for you. Nonetheless, there are many things you can do to make it more likely that you will receive a reply. This community is fortunate to have an exceptionally large number of friendly and knowledgeable people who enjoy corresponding on these lists. The vast majority of them will be happy to help you if you follow these simple guidelines.

### Be polite and respectful

Remember, no one here is under any obligation to reply to you. Think about your readers. Most of them are coming home after a long, hard day at work. The last thing they need is someone's temper tantrum. If you are rude and disrespectful, you are very likely to be ignored.

### Be concise

Include only essential information. Most of your readers lead busy lives and have precious little time. We *want* to spend some of that time helping you, if we can. But if you ramble, it will be easier to skip over you and help someone else who gets right to the point.

### Help us help you

Tell us what you've already tried, and which documentation pages you've already read. Put yourself in your readers' shoes. What essential information would they require in order to be able to help you? Make sure to include that information in your message. A great way to provide your hardware details is by *generating and submitting a Hardware Compatibility List (HCL) report*, then linking to it in your message. *Ask questions the smart way*.

### Be patient

Do not "bump" a thread more than once every three days *at most*. If it seems like your messages to the mailing lists are consistently being ignored, make sure you're following the guidelines explained on this page. If you're already doing so but still not getting any replies, then it's likely that no one who knows the answer has had time to reply yet. Remember that the devs are very busy working on Qubes. They usually only have a chance to answer questions on the mailing lists once every several days.

### Be a good community member

As with any social community, members earn different reputations for themselves over time. We want these discussion venues to be friendly, productive places where information and ideas are exchanged for the mutual benefit of all. We understand that the best way to achieve this is to encourage and cultivate other like-minded individuals. Those who have shown themselves to be good community members through their past contributions have earned our good will, and we will be especially eager to help them and collaborate with them. If you are new to the community, you should understand that it may take time for you to earn the good will of others. This does not mean that you will not receive help. On the contrary, we are fortunate to have such a helpful and understanding community that many of them spend hours of their personal time helping complete strangers, including many who post anonymously. (Given the integration of Qubes with *Whonix*, we understand better than most the complexities of privacy and anonymity, and we know that many users have no other choice but to post anonymously.) You can read our project's *Code of Conduct* and *Privacy Policy* for more information.

## Report issues and submit changes in the right places

The mailing lists and *forum* are good places to ask questions and discuss things. However, if you're submitting a more formal report, we'd prefer that you submit it to our *issue tracker* so that it doesn't get overlooked. (However, please remember that *the issue tracker is not a discussion forum*.) Likewise, if you see that something in the documentation should be changed, don't simply point it out in a discussion venue. Instead, submit the change.

## Moderation

The moderation team aims to enforce our *Code of Conduct*. Beyond this, users should not expect any specific action from the moderation team. Specifically, users should not request that posts or messages be deleted or edited by a moderator. Users are reminded that, in most venues, anything posted will be sent out as an email to others, and these emails cannot be deleted from others' inboxes.

## Specific mailing list rules and notes

### Use the correct list

Send your message to the correct list. Read the sections below to determine which list is correct for your message.

### Do not top-post

*Top-posting* is placing your reply above the quoted message to which you're replying. Please refrain from doing this. Instead, either *interleave* your reply by placing parts of your message immediately below each quoted portion to which it is replying, or *bottom-post* by placing your entire reply below the quoted message to which you're replying.

### Use proper subject lines

Include a precise and informative subject line. This will allow others to easily find your thread in the future and use it as a reference. (Bad: "Help! Qubes problems!" Good: "R2B2 Installation problem: Apple keyboard not working in installer.")

### Do not send duplicates

If your message is not successfully sent to the list, it probably got caught in the spam filter. We check the spam filter regularly, so please be patient, and your message should be approved (and your email address added to the whitelist) within a few days.

### Keep the list CCed

Keep the mailing list CCed throughout the conversation unless there's a special need for privacy (in which case, use PGP encryption). This increases the likelihood that a greater quantity of useful information will be available to everyone in the future.

### Quote appropriately

If you're replying to a thread (whether your own or someone else's), you should make sure to quote enough from previous messages in the thread so that people reading your message can understand the context without having to find and read earlier messages from that thread. Each reply should continue the conversation and, ideally, be readable as a conversation in itself. Do not quote advertisements in signatures or inline PGP signature blocks. (Quoting the latter interferes with the ability of programs like Enigmail to properly quote replies thereafter).

### English not required

If you do not speak English, you should feel free to post in your own language. However, bear in mind that most members of the list can only read English. You may wish to include an automated translation in your message out of consideration for those readers. If you choose to write in English, please do not apologize for doing so poorly, as it is unnecessary. We understand and will ask for clarification if needed.

### Suggestions

While we're generally open to hearing suggestions for new features, please note that we already have a pretty well defined [roadmap](#), and it's rather unlikely that we will change our schedule in order to accommodate your request. If there's a particular feature you'd like to see in Qubes, a much more effective way to make it happen is to contribute a patch that implements it. We happily accept such contributions, provided they meet our standards. Please note, however, that it's always a good idea to field a discussion of your idea on the `qubes-devel` list before putting in a lot of hard work on something that we may not be able or willing to accept.

### Google Groups

While the mailing lists are implemented as Google Group web forums, a Google account is in no way required, expected, or encouraged. Many discussants (including most members of the Qubes team) treat these lists as conventional [mailing lists](#), interacting with them solely through plain text email with MUAs like [Thunderbird](#) and [Mutt](#). The Google Groups service is just free infrastructure, and we *distrust the infrastructure*. This is why, for example, we encourage discussants to use [Split GPG](#) to sign all of their messages to the lists, but we do not endorse the use of these Google Groups as web forums. For that, we have a separate, dedicated [forum](#).

## 1.7.4 Mailing lists

This section covers each of our individual [mailing lists](#), with details about the purpose of each list and how to use it. A Google account is **not** required for any of these mailing lists.

### qubes-announce

This is a read-only list for those who wish to receive only very important, infrequent messages. Only the core Qubes team can post to this list. Only [Qubes Security Bulletins \(QSBs\)](#), new stable Qubes OS releases, and Qubes OS release end-of-life notices are announced here.

To subscribe, send a blank email to `qubes-announce+subscribe@googlegroups.com`. (Note: A Google account is **not** required. Any email address will work.) To unsubscribe, send a blank email to `qubes-announce+unsubscribe@googlegroups.com`. This list also has a [traditional mail archive](#) and an optional [Google Groups web interface](#).

## qubes-users

This list is for helping users solve various daily problems with Qubes OS. Examples of topics or questions suitable for this list include:

- HCL reports
- Installation problems
- Hardware compatibility problems
- Questions of the form: “How do I...?”

Please try searching both the Qubes website and the archives of the mailing lists before sending a question. In addition, please make sure that you have read and understood the following basic documentation prior to posting to the list:

- The *Installation Guide*, *System Requirements*, and HCL (for problems related to installing Qubes OS)
- The *User FAQ*
- The *documentation* (for questions about how to use Qubes OS)

You must be subscribed in order to post to this list. To subscribe, send a blank email to [qubes-users+subscribe@googlegroups.com](mailto:qubes-users+subscribe@googlegroups.com). (Note: A Google account is **not** required. Any email address will work.) To post a message to the list, address your email to [qubes-users@googlegroups.com](mailto:qubes-users@googlegroups.com). If your post does not appear immediately, please allow time for moderation to occur. To unsubscribe, send a blank email to [qubes-users+unsubscribe@googlegroups.com](mailto:qubes-users+unsubscribe@googlegroups.com). This list also has a [traditional mail archive](#) and an optional [Google Groups web interface](#).

## qubes-devel

This list is primarily intended for people who are interested in contributing to Qubes or who are willing to learn more about its architecture and implementation. Examples of topics and questions suitable for this list include:

- Questions about why we made certain architecture or implementation decisions.
  - For example: “Why did you implement XYZ this way and not the other way?”
- Questions about code layout and where code is for certain functionality.
- Discussions about proposed new features, patches, etc.
  - For example: “I would like to implement feature XYZ.”
- Contributed code and patches.
- Security discussions which are relevant to Qubes in some way.

You must be subscribed in order to post to this list. To subscribe, send a blank email to [qubes-devel+subscribe@googlegroups.com](mailto:qubes-devel+subscribe@googlegroups.com). (Note: A Google account is **not** required. Any email address will work.) To post a message to the list, address your email to [qubes-devel@googlegroups.com](mailto:qubes-devel@googlegroups.com). If your post does not appear immediately, please allow time for moderation to occur. To unsubscribe, send a blank email to [qubes-devel+unsubscribe@googlegroups.com](mailto:qubes-devel+unsubscribe@googlegroups.com). This list also has a [traditional mail archive](#) and an optional [Google Groups web interface](#).

### qubes-project

This list is for non-technical discussion and coordination around the Qubes OS project.

Examples of topics or question suitable for this list include:

- Participation (talks, workshops, etc.) at upcoming events
- Project funding applications and strategies
- FOSS governance discussions
- Most Github issues tagged [business](#) or [project management](#)

You must be subscribed in order to post to this list. To subscribe, send a blank email to [qubes-project+subscribe@googlegroups.com](mailto:qubes-project+subscribe@googlegroups.com). (Note: A Google account is **not** required. Any email address will work.) To post a message to the list, address your email to [qubes-project@googlegroups.com](mailto:qubes-project@googlegroups.com). If your post does not appear immediately, please allow time for moderation to occur. To unsubscribe, send a blank email to [qubes-project+unsubscribe@googlegroups.com](mailto:qubes-project+unsubscribe@googlegroups.com). This list also has a [traditional mail archive](#) and an optional [Google Groups web interface](#).

### qubes-translation

This list is for discussion around the localization and translation of Qubes OS, its documentation, and the website.

Examples of topics or question suitable for this list include:

- Questions about or issues with [Transifex](#), the translation platform we use
- Who is managing localization for a given language
- Most Github issues tagged [localization](#)

You must be subscribed in order to post to this list. To subscribe, send a blank email to [qubes-translation+subscribe@googlegroups.com](mailto:qubes-translation+subscribe@googlegroups.com). (Note: A Google account is **not** required. Any email address will work.) To post a message to the list, address your email to [qubes-translation@googlegroups.com](mailto:qubes-translation@googlegroups.com). If your post does not appear immediately, please allow time for moderation to occur. To unsubscribe, send a blank email to [qubes-translation+unsubscribe@googlegroups.com](mailto:qubes-translation+unsubscribe@googlegroups.com). This list also has an optional [Google Groups web interface](#).

## 1.7.5 Forum

The official [Qubes Forum](#) is a place where you can ask questions, get help, share tips and experiences, and more! For a long time, members of our community have sought a privacy-respecting forum experience with modern features that traditional mailing lists do not support. The open-source [Discourse](#) platform fills this need for us, as it does for many other open-source projects.

### Why was this forum created?

Previously, the only option for a forum-like experience was to interact with our mailing lists via Google Groups, but we understand all too well that the privacy implications and user experience were unacceptable for many members of our community, especially with the recent addition of a sign-in requirement to view threads. Many of you value the lower barrier to entry, organization, ease-of-use, and modern social features that today's forums support. Moreover, Discourse [features email integration](#) for those who still prefer the traditional mailing list format.

### How is this different from our mailing lists?

To be clear, this is *not* a replacement for the mailing lists. This forum is simply an *additional* place for discussion. Certain types of discussions naturally lend themselves more to mailing lists or to forums, and different types of users prefer different venues. We’ve heard from some users who find the mailing lists to be a bit intimidating or who may feel that their message isn’t important enough to merit creating a new email that lands in thousands of inboxes. Others want more selective control over topic notifications. Some users simply appreciate the ability to add a “reaction” to a message instead of having to add an entirely new reply. Whatever your reasons, it’s up to you to decide where and how you want to join the conversation.

### Does this split the community?

Many open-source projects (such as Fedora and Debian) have both mailing lists and forums (and additional discussion venues). In fact, the Qubes OS Project already had non-mailing-list discussion venues such as [Reddit](#) before this forum was introduced. We believe that this additional venue fosters the continued growth of community participation and improves everyone’s experience. In addition, we fully expect that many community members – especially the most active ones – will choose to participate in both venues. (Again, for those who still prefer interacting via email, [Discourse supports that too!](#))

## 1.7.6 Social media

The Qubes OS Project has a presence on the following social media platforms:

- [Twitter](#)
- [Mastodon](#)
- [Reddit](#)
- [Facebook](#)
- [LinkedIn](#)

Generally speaking, these are not intended to be primary support venues. (Those would be [qubes-users](#) and the [forum](#).) Rather, these are primarily intended to be a way to more widely disseminate items published on the [news](#) page. If you use one of these platforms, you may find it convenient to follow the Qubes OS Project there as a way of receiving Qubes news.

## 1.7.7 Chat

If you’d like to chat, join us on the [#qubes](#) IRC channel (or its Matrix bridge: [#qubes:libera.chat](#)).

## 1.7.8 Unofficial venues

If you find another venue on the Internet that is not listed above, it is **unofficial**, which means that the Qubes team does **not** monitor or moderate it. Please be especially careful in unofficial venues.

(Note: If a Qubes team member discovers the venue and decides to pop in, that should not be taken as a commitment to monitor or moderate the venue. It still remains unofficial. Also, please make sure someone claiming to be a Qubes team member really is one. It could be an impostor!)

## 1.8 How to contribute

Thank you for your interest in contributing to Qubes! Here are some of the many ways in which you can help:

- Audit the *source code*
- *Report security issues*
- *Send patches* to fix bugs or implement features
- *Contribute packages*
- *Report bugs*
- *Test new releases and updates*
- Submit HCL reports for your hardware
- Record *video tours*
- Create *artwork* (plymouth themes, installer themes, wallpapers, etc.)
- Write and edit the documentation
- *Donate* to the project
- If you represent an organization, become a *Qubes partner*
- Add a Qubes download mirror
- Answer questions and discuss Qubes on the *mailing lists* and *forum*
- Engage with us on social media:
  - Follow us on *Twitter*
  - Join us on *Reddit*
  - Like us on *Facebook*
  - Connect with us on *LinkedIn*
- And last but not least, tell your friends and colleagues about how Qubes can help them secure their digital lives!

### 1.8.1 Contributing Code

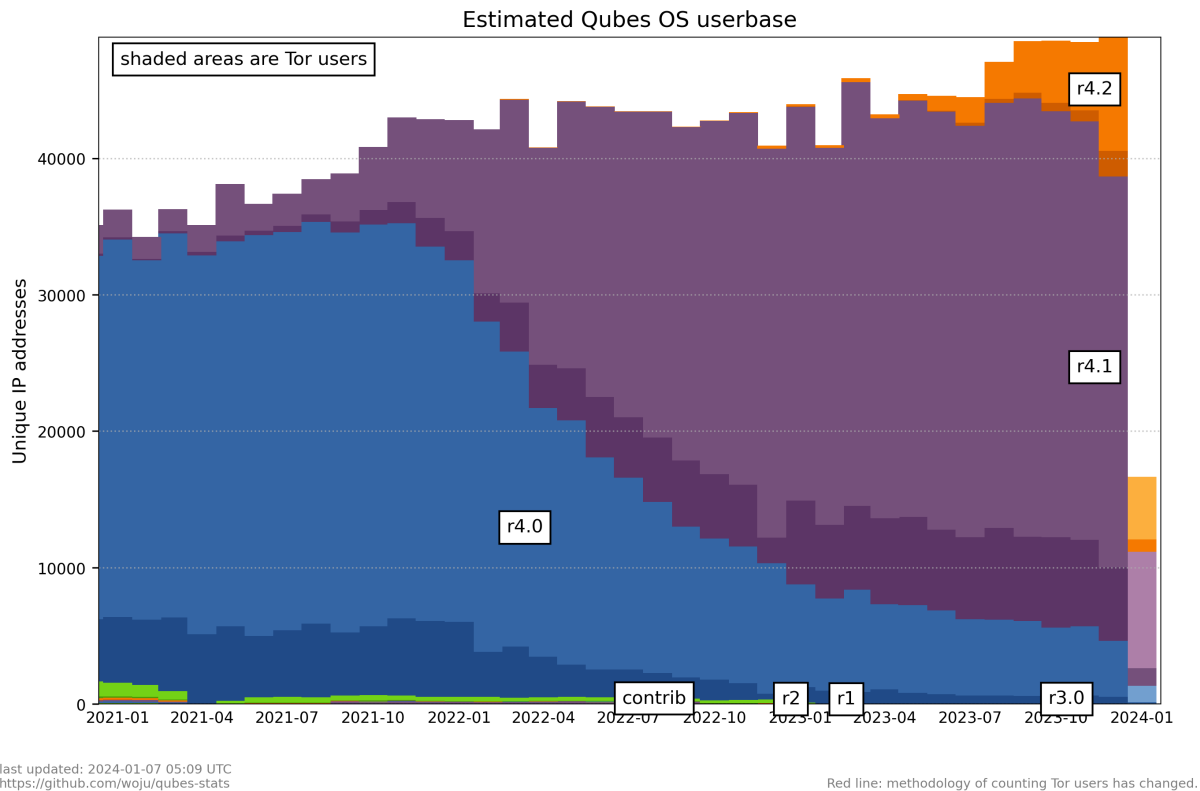
If you're interested in contributing code, the best starting point is to have a look at our *GitHub issues* to see which tasks are the most urgent. You can filter issues depending on your interest and experience. For example, here are some common issue labels:

- *Help wanted*
- *UX and usability*
- *Windows tools*
- *Documentation*
- *Privacy*
- *Debian/Ubuntu*

Before you engage in an activity that will take you a significant amount of time, like implementing a new feature, it's always good to contact us first, preferably via the *qubes-devel* mailing list. Once we've worked out the details, we'll be grateful to *receive your patch*.



## 1.9 Statistics



### 1.9.1 FAQ

#### How often is this graph updated?

Daily.

#### Why is the bar for the current month so low?

Since the graph is updated daily, the bar for the current month will be very low at the start of the month and rise gradually until the end of the month.

#### How is the userbase estimated?

We simply count the number of unique IPv4 addresses that connect to the Qubes update servers each month (except for Tor connections; see [below](#)). (Note: Users who have manually configured their systems to bypass the metalink and connect directly to a mirror are not counted.)

## How are Tor users counted?

We estimate the number of Tor users as a proportion of the total number of *requests* from Tor exit nodes on the assumption that the proportion of users to requests is roughly the same for both clearnet and Tor users. To be precise, the formula is:

$$\text{tor\_users} = \text{tor\_requests} * (\text{plain\_users} / \text{plain\_requests})$$

Where:

- `tor_users` is the estimated number of Qubes users who download updates via Tor each month.
- `tor_requests` is the total number of requests the Qubes update servers receive from Tor exit nodes each month.
- `plain_users` is the number of unique clearnet IPv4 addresses that connect to the Qubes update servers each month.
- `plain_requests` is the total number of requests the Qubes update servers receive from clearnet IPv4 addresses each month.

We cross-reference the list of connecting IP addresses with [TorDNSSEL's exit lists](#) in order to distinguish Tor and clearnet IPs and requests. For this purpose, we count an IP address as belonging to a Tor exit node if there was a Tor exit node active for that address within the 24-hour periods before or after it connected to the Qubes update servers.

## What kinds of data do you collect about Qubes users?

Please see our [Privacy Policy](#).

## Where can I find the raw data and source code?

The raw data is available [here](#). (This does not include any personally-identifying user data.) Please note that the format of this data is not documented and may change any time if the developers feel the need to include something else. The source code is available [here](#).

# 1.10 Code of conduct

## 1.10.1 Introduction

This Code of Conduct is a collaborative, evolving document that attempts to transparently set out a public set of standards regarding appropriate conduct in the Qubes OS Project. It is *not* intended to be a statement or endorsement, whether implicit or explicit, of any particular political or philosophical attitude, belief, or way of living. Rather, it is an attempt to find a reasonable middle ground among the inevitable disagreements regarding free expression that arise in a large, diverse community of people from around the world. It is intended to be a practical means of serving the best interests of our users, contributors, and the project itself. We welcome you to view the [history of changes](#) to this document and the [discussion](#) leading to its creation.

### 1.10.2 Our Pledge

The Qubes OS project creates a reasonably secure OS. In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to make participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, sexual identity and orientation, or other characteristic.

### 1.10.3 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Reinforcing stereotypical models for illustration of non-technical users (e.g. our mothers/grandmothers, etc.)
- Public or private harassment, as defined by the [Citizen Code of Conduct](#)
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

(Please also see our [discussion guidelines](#).)

### 1.10.4 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior. This action can include removing, editing, or rejecting comments, commits, code, wiki edits, issues, and other contributions, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

### 1.10.5 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

In all other cases, this Code of Conduct applies only in the official project venues specified on [this page](#); it does not apply to unofficial venues, community-run venues, or any other public or private place. For example, if a Qubes user decides to create an unofficial discussion space on a third-party platform, and someone appears to violate this Code of Conduct in that space, we are not responsible for taking any action with respect to that behavior; the venue; the venue's creators, owners, leaders, or moderators; or anything else pertaining to the incident.

### 1.10.6 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project CoC team at [mods@qubes-os.org](mailto:mods@qubes-os.org). The project CoC team is the [Marek Marczykowski-Górecki](#), [Andrew David Wong](#), and [Michael Carbone](#). All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident, and will ensure reporter, reported and all others impacted are regularly updated through the process. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

### 1.10.7 A Note on Trust

Expect all contributions to be reviewed with some amount of healthy adversarial skepticism, regardless of your perceived standing in the community. This is a security project, and allowing ourselves to get complacent while reviewing code simply because it comes from a well-known party would not be in the best interest of the project. Please try not to get offended if you perceive your contributions as being met with distrust – we are most definitely thankful and appreciative of your efforts, but must also remain vigilant in order to ensure continued quality and safeguard against potential sabotage.

### 1.10.8 Attribution

The initial published version of this Code of Conduct was adapted from the [Contributor Covenant, version 1.4](#) and the [Rust Code of Conduct](#).

## 1.11 Privacy policy

The short version is that we try to respect your privacy as much as possible. We absolutely do not sell any user data. In fact, we go out of our way to help you keep your data private from everyone, including us. For example, from the moment you *install Qubes OS*, we offer to set up [Whonix](#) so that all of your updates are routed through [Tor](#).

### 1.11.1 Website

For the legally-required boilerplate, see [Website Privacy Policy](#).

This is just a static website generated with Jekyll and hosted from GitHub Pages. We try to use as little JavaScript as possible. We host all resources locally (no third-party CDNs) so that you only have to connect to one domain. This site should be easy to browse using Tor Browser and with scripts blocked. We also have an [onion service](#) (access is not logged). We even go out of our way to make it easy to download [this website's git repo](#), including all the website source code, so that you can host this entire site from your own local machine offline. Better yet, we've specifically written all of the [documentation](#) in Markdown so that the plain text can be enjoyed from the comfort of your terminal. Here's the [repo](#). (By the way, Git tags on our repos are PGP-signed so you can [verify](#) the authenticity of the content.) Obviously, we don't use any ads or trackers, but this is still a public website, so man-in-the-middle attacks and such are always a possibility. Please be careful. See [FAQ: Should I trust this website?](#)

### 1.11.2 Update Servers and Repositories

We provide repositories at <https://yum.qubes-os.org> and <https://deb.qubes-os.org>.

We collect and store standard server access and error logs, which include IP addresses. We use this data for generating *Qubes userbase statistics* and for incident response.

The data is retained for up to three months so that we can re-calculate the previous two months' statistics in case anything goes wrong. After that, the data is deleted. We never sell the data to anyone or share it with any third party.

If you would like to hide your IP address from us, we strongly encourage it and are happy to help you do so! Simply choose the Whonix option to route all of your updates over Tor when *installing Qubes OS*.

### 1.11.3 Onion Services

We provide an *onion service* for the website and onion service mirrors of the repositories. Access to these servers is not logged.

### 1.11.4 Mirrors

There are also other third-party mirrors hosted by volunteers. These are used both for *ISO downloads* and *updates*. We have no control over what data these mirrors collect or with whom they share it. Please see the privacy policy of each respective mirror operator.

### 1.11.5 Qubes OS

We have specifically designed Qubes OS so that it is not possible to collect any data directly from Qubes OS installations. In other words, Qubes OS does not have the ability to “phone home” and is intentionally architected to forbid that from happening. This is mainly because we have ensured that dom0 has no network access.

We don't want the ability to collect any data directly from Qubes OS installations, because if anyone has that power, then the system is not secure. We use Qubes OS ourselves as a daily driver for our work and personal lives, so our interests are aligned with yours. We want privacy too! Thankfully, Qubes OS is free and open-source software, so you don't have to take our word for it.

Of course, third-party software (including other operating systems) running inside of Qubes OS may not be as privacy-respecting, so please be mindful of what you install. We have no control over such third-party software.

For more information, please see *FAQ: How does Qubes OS provide privacy?*

## 1.12 User Documentation

Core documentation for Qubes users.

### 1.12.1 System requirements

**Warning:** Notice: The system requirements on this page are necessary, but not sufficient, for Qubes compatibility at a minimal or recommended level. In other words, just because a computer satisfies these requirements doesn't mean that Qubes will successfully install and run on it. We strongly recommend consulting the resources below when selecting hardware for Qubes.

#### Minimum

- **CPU:** 64-bit Intel or AMD processor (also known as x86\_64, x64, and AMD64)
  - Intel VT-x with EPT or AMD-V with RVI
  - Intel VT-d or AMD-Vi (also known as AMD IOMMU)
- **Memory:** 6 GB RAM
- **Storage:** 32 GB free space

#### Recommended

- **CPU:** 64-bit Intel or AMD processor (also known as x86\_64, x64, and AMD64)
  - Intel VT-x with EPT or AMD-V with RVI
  - Intel VT-d or AMD-Vi (also known as AMD IOMMU)
- **Memory:** 16 GB RAM
- **Storage:** 128 GB free space
  - High-speed solid-state drive strongly recommended
- **Graphics:** Intel integrated graphics processor (IGP) strongly recommended
  - Nvidia GPUs may require significant [troubleshooting](#)
  - AMD GPUs have not been formally tested, but Radeons (especially RX580 and earlier) generally work well
- **Peripherals:** A non-USB keyboard or multiple USB controllers
- **TPM:** Trusted Platform Module (TPM) with proper BIOS support (required for *Anti Evil Maid*)

#### Qubes-certified hardware

The following are *required* for *Qubes-certified hardware devices* but *merely recommended* for *non-certified* hardware (see the *hardware certification requirements* for details).

- Open-source boot firmware (e.g., [coreboot](#))
- Hardware switches for all built-in USB-connected microphones (if any)
- Either support for non-USB input devices (e.g., via PS/2, which most laptops already use internally) or a separate USB controller only for input devices

## Choosing Hardware

We recommend consulting these resources when selecting hardware for Qubes OS:

- *Certified hardware* — Qubes developer certified, officially recommended
- *Community-recommended hardware* — list curated and maintained by the community, unofficially recommended
- Hardware compatibility list (HCL) — community test results, neither recommended nor disrecommended

## Important Notes

- **Installing Qubes in a virtual machine is not recommended, as it uses its own bare-metal hypervisor (Xen).**
- Qubes **can** be installed on many systems that do not meet the recommended requirements. Such systems will still offer significant security improvements over traditional operating systems, since things like GUI isolation and kernel protection do not require special hardware.
- Qubes **can** be installed on a USB flash drive or external disk, and testing has shown that this works very well. A fast USB 3.0 flash drive is recommended for this. (As a reminder, its capacity must be at least 32 GiB.) Simply plug the flash drive into the computer before booting into the Qubes installer from a separate installation medium, choose the flash drive as the target installation disk, and proceed with the installation normally. After Qubes has been installed on the flash drive, it can then be plugged into other computers in order to boot into Qubes. In addition to the convenience of having a portable copy of Qubes, this allows users to test for hardware compatibility on multiple machines (e.g., at a brick-and-mortar computer store) before deciding on which computer to purchase. (See *generating and submitting HCL reports* for advice on hardware compatibility testing.) Remember to change the devices assigned to your NetVM and USB VM if you move between different machines.
- You can check whether an Intel processor has VT-x and VT-d on [ark.intel.com](http://ark.intel.com).

### 1.12.2 Certified hardware

The Qubes OS Project aims to partner with a select few computer vendors to ensure that Qubes users have reliable hardware purchasing options. We aim for these vendors to be as diverse as possible in terms of geography, cost, and availability.

**Danger:** Warning: The Qubes OS Project certifies only that a particular hardware configuration is supported by Qubes OS and is available to purchase with Qubes OS preinstalled. We take no responsibility for any vendor's manufacturing, shipping, payment, or other practices; nor can we control whether physical hardware is modified (whether maliciously or otherwise) en route to the user.

You may also be interested in the [community-recommended hardware](#) list and the hardware compatibility list (HCL).

## Qubes-certified computers

Qubes-certified computers are certified for a *major release* and regularly tested by the Qubes developers to ensure compatibility with all of Qubes' features within that major release. The developers test all new updates within that major release to ensure that no regressions are introduced.

The current Qubes-certified models are listed below in chronological order of certification.

## Insurgo PrivacyBeast X230



The [Insurgo PrivacyBeast X230](#) is a laptop based on the ThinkPad X230. It is certified for Qubes OS 4.X.



## NitroPad X230



The [NitroPad X230](#) is a laptop based on the ThinkPad X230. It is certified for Qubes OS 4.X.

## NitroPad T430



The [NitroPad T430](#) is a laptop based on the ThinkPad T430. It is certified for Qubes OS 4.X.

### Dasharo FidelisGuard Z690



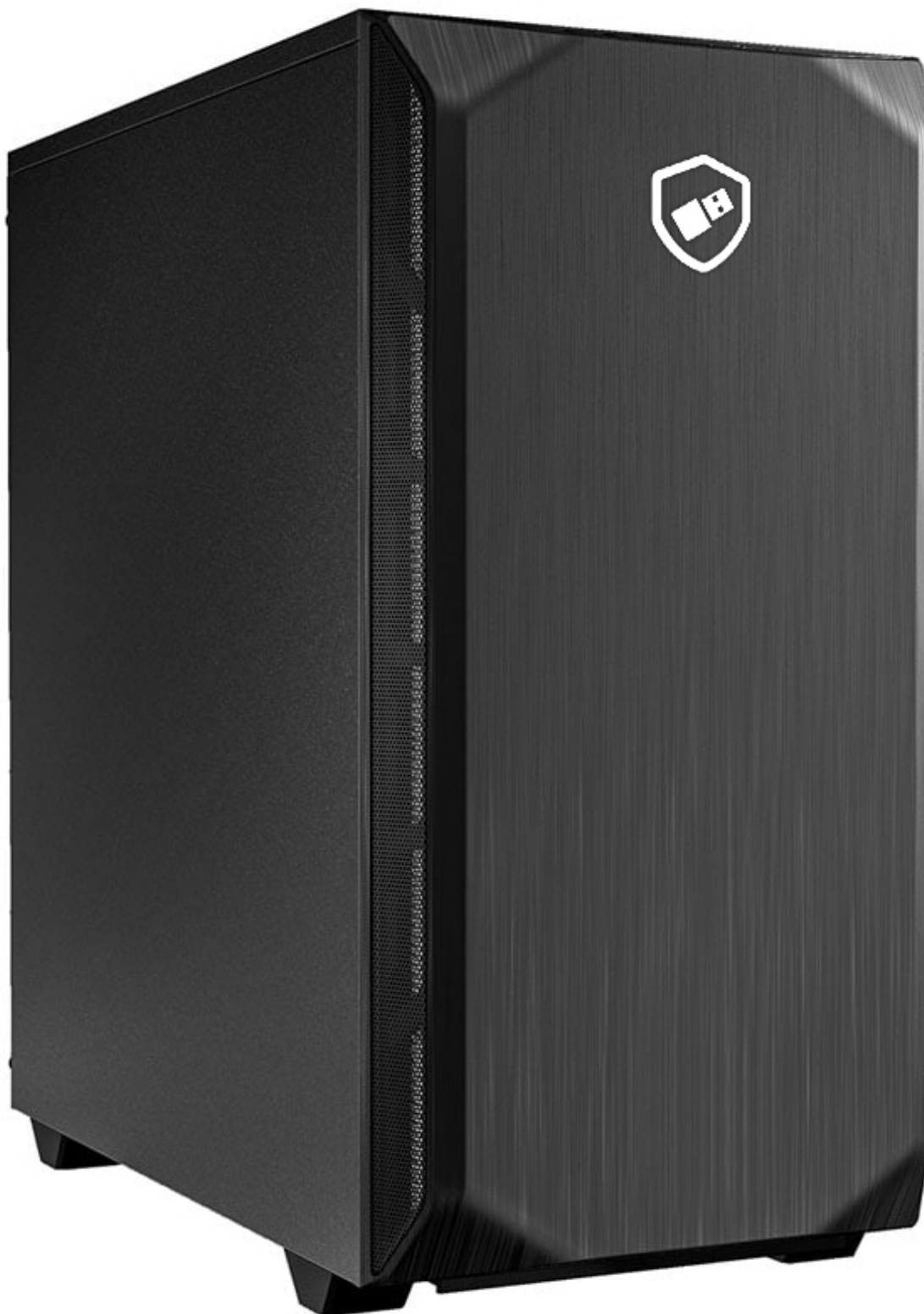
The [Dasharo FidelisGuard Z690](#) is a desktop based on the MSI PRO Z690-A DDR4 motherboard. It is certified for Qubes OS 4.X.

## NovaCustom NV41 Series



The [NovaCustom NV41 Series](#) is a 14-inch custom laptop. It is certified for Qubes OS 4.X.

## NitroPC Pro





The [NitroPC Pro](#) is a desktop based on the MSI PRO Z690-A DDR5 motherboard. It is certified for Qubes OS 4.X.

### Become hardware certified

If you are a hardware vendor, you can have your hardware certified as compatible with Qubes OS. The benefits of hardware certification include:

- Your customers can purchase with confidence, knowing that they can take full advantage of Qubes OS on your hardware for a specific major version.
- We will continue testing your hardware to ensure compatibility with the supported major version. In the course of this testing, we will also test your hardware against upcoming versions, which can help with future planning.
- Your hardware will continue to be compatible with Qubes OS as it further develops within that major version, and we will work with you toward preserving compatibility and certification in future releases.
- You can support the development of Qubes OS.

### Hardware certification requirements

**Note:** This section describes the requirements for hardware *certification*, *not* the requirements for *running* Qubes OS. For the latter, please see the [system requirements](#). A brief list of the requirements described in this section is available [here](#).

A basic requirement is that all Qubes-certified devices must be available for purchase with Qubes OS preinstalled. Customers may be offered the option to select from a list of various operating systems (or no operating system at all) to be preinstalled, but Qubes OS must be on that list in order to maintain Qubes hardware certification.

One of the most important security improvements introduced with the release of Qubes 4.0 was to replace paravirtualization (PV) technology with **hardware-enforced memory virtualization**, which recent processors have made possible thanks to so-called Second Level Address Translation ([SLAT](#)), also known as [EPT](#) in Intel parlance. SLAT (EPT) is an extension to Intel VT-x virtualization, which originally was capable of only CPU virtualization but not memory virtualization and hence required a complex Shadow Page Tables approach. We hope that embracing SLAT-based memory virtualization will allow us to prevent disastrous security bugs, such as the infamous [XSA-148](#), which — unlike many other major Xen bugs — regrettably did [affect](#) Qubes OS. Consequently, we require SLAT support of all certified hardware beginning with Qubes OS 4.0.

Another important requirement is that Qubes-certified hardware should run only **open-source boot firmware** (aka “the BIOS”), such as [coreboot](#). The only exception is the use of (properly authenticated) CPU-vendor-provided blobs for silicon and memory initialization (see [Intel FSP](#)) as well as other internal operations (see [Intel ME](#)). However, we specifically require all code used for and dealing with the System Management Mode (SMM) to be open-source.

While we [recognize](#) the potential problems that proprietary CPU-vendor code can cause, we are also pragmatic enough to realize that we need to take smaller steps first, before we can implement even stronger countermeasures such as a [stateless laptop](#). A switch to open source boot firmware is one such important step. To be compatible with Qubes OS, the BIOS must properly expose all the VT-x, VT-d, and SLAT functionality that the underlying hardware offers (and which we require). Among other things, this implies **proper DMAR ACPI table** construction.

Most laptops use PS/2 connections internally for their input devices (i.e., keyboard and touchpad). On most desktops, however, USB-connected keyboards and mice have become standard. This presents a dilemma when the computer has only one USB controller. If that single USB controller is dedicated solely to the input devices, then no untrusted USB devices can be used. Conversely, if the sole USB controller is completely untrusted, then there is no way for the user to physically control the system in a secure way. In practice, Qubes users on such hardware systems are generally forced to use a single USB controller for both trusted and untrusted purposes — [an unfortunate security trade-off](#). For this reason, we require that every Qubes-certified non-laptop device **either** (1) supports non-USB input devices (e.g., via PS/2) **or** (2) has a separate USB controller that is only for input devices.

Finally, we require that Qubes-certified hardware does not have any built-in *USB-connected* microphones (e.g. as part of a USB-connected built-in camera) that cannot be easily physically disabled by the user, e.g. via a convenient mechanical switch. Thankfully, the majority of laptops on the market that we have seen already satisfy this condition out-of-the-box, because their built-in microphones are typically connected to the internal audio device, which itself is a type of PCIe device. This is important, because such PCIe audio devices are — by default — assigned to Qubes’ (trusted) dom0 and exposed through our carefully designed protocol only to select app qubes when the user explicitly chooses to do so. The rest of the time, they should be outside the reach of malware.

While we also recommend a physical kill switch on the built-in camera (or, if possible, not to have a built-in camera), we also recognize this isn’t a critical requirement, because users who are concerned about it can easily cover it a piece of tape (something that, regrettably, is far less effective on a microphone).

Similarly, we don’t consider physical kill switches on Wi-Fi and Bluetooth devices to be mandatory. Users who plan on using Qubes in an air-gap scenario would do best if they manually remove all such devices persistently (as well as the builtin *speakers!*), rather than rely on easy-to-flip-by-mistake switches, while others should benefit from the Qubes default sandboxing of all networking devices in dedicated VMs.

We hope these hardware requirements will encourage the development of more secure and trustworthy devices.

## Hardware certification process

To have hardware certified, the vendor must:

1. Send the Qubes team two (2) units for testing (non-returnable) for each configuration the vendor wishes to be offering.
2. Offer to customers the very same configuration (same motherboard, same screen, same BIOS version, same Wi-Fi module, etc.) for at least one year.
3. Pay the Qubes team a flat monthly rate, to be agreed upon between the hardware vendor and the Qubes team.

It is the vendor’s responsibility to ensure the hardware they wish to have certified can run Qubes OS, at the very least the latest stable version. This could be done by consulting the Hardware Compatibility List or trying to install it themselves before shipping any units to us. While we are willing to troubleshoot simple issues, we will need to charge a consulting fee for more in-depth work.

If you are interested in having your hardware certified, please [contact us](#).

### 1.12.3 How to use the hardware compatibility list (HCL)

The HCL is a compilation of reports generated and submitted by users across various Qubes versions about their hardware’s compatibility with Qubes.

**Note:** Except in the case of developer-reported entries, the Qubes team has not independently verified the accuracy of these reports. Please first consult the data sheets (CPU, chipset, motherboard) prior to buying new hardware for Qubes. Make sure it meets the *System Requirements* and search in particular for support of:

- HVM (“AMD virtualization (AMD-V)”, “Intel virtualization (VT-x)”, “VIA virtualization (VIA VT)”)
- IOMMU (“AMD I/O Virtualization Technology (AMD-Vi)”, “Intel Virtualization Technology for Directed I/O (VT-d)”)
- TPM (“Trusted Platform Module (TPM)” connected to a “20-pin TPM header” on motherboards.)

If using the list to make a purchasing decision, we recommend that you choose hardware with:

- the best achievable Qubes security level (green columns in HVM, IOMMU, TPM)
- and general machine compatibility (green columns in Qubes version, dom0 kernel, remarks).

Also see *Certified Hardware*.

### Generating and Submitting New Reports

In order to generate an HCL report in Qubes, simply open a terminal in dom0 (Applications Menu > Terminal Emulator) and run `qubes-hcl-report <qube-name>`, where `<qube-name>` is the name of the qube in which the generated HCL files will be saved.

You are encouraged to submit your HCL report for the benefit of further Qubes development and other users. When submitting reports, test the hardware yourself, if possible. If you would like to submit your HCL report, please copy and paste the contents of the **HCL Info** .yaml file into an email to the [qubes-users mailing list](#) with the subject HCL - `<your machine model name>`, or create a post in the [HCL Reports category](#) of the forum. Pasting the contents into the email or post has the advantage that members of the mailing list and the forum can see the report without downloading and opening a file. In addition, new forum members are unable to attach files to posts.

Please include any useful information about any Qubes features you may have tested (see the legend below), as well as general machine compatibility (video, networking, sleep, etc.). Please consider sending the **HCL Support Files** .cpio.gz file as well. To generate these add the `-s` or `--support` command line option.

**Please note:** The **HCL Support Files** may contain numerous hardware details, including serial numbers. If, for privacy or security reasons, you do not wish to make this information public, please **do not** post the .cpio.gz file on a public mailing list or forum.

### 1.12.4 Installation guide

Welcome to the Qubes OS installation guide! This guide will walk you through the process of installing Qubes. Please read it carefully and thoroughly, as it contains important information for ensuring that your Qubes OS installation is functional and secure.

#### Pre-installation

#### Hardware requirements

**Danger:** Warning: Qubes has no control over what happens on your computer before you install it. No software can provide security if it is installed on compromised hardware. Do not install Qubes on a computer you don't trust. See installation security for more information.

Qubes OS has very specific *system requirements*. To ensure compatibility, we strongly recommend using *Qubes-certified hardware*. Other hardware may require you to perform significant troubleshooting. You may also find it helpful to consult the Hardware Compatibility List.

Even on supported hardware, you must ensure that **IOMMU-based virtualization** is activated in the BIOS or UEFI. Without it, Qubes OS won't be able to enforce isolation. For Intel-based boards, this setting is called Intel Virtualization for Directed I/O (**Intel VT-d**) and for AMD-based boards, it is called AMD I/O Virtualization Technology (or simply **AMD-Vi**). This parameter should be activated in your computer's BIOS or UEFI, alongside the standard Virtualization (**Intel VT-x**) and AMD Virtualization (**AMD-V**) extensions. This [external guide](#) made for Intel-based boards can help you figure out how to enter your BIOS or UEFI to locate and activate those settings. If those settings are not nested under the Advanced tab, you might find them under the Security tab.



**Warning:** Note: Qubes OS is not meant to be installed inside a virtual machine as a guest hypervisor. In other words, nested virtualization is not supported. In order for a strict compartmentalization to be enforced, Qubes OS needs to be able to manage the hardware directly.

## Copying the ISO onto the installation medium

Pick the most secure existing computer and OS you have available for downloading and copying the Qubes ISO onto the installation medium. Download a Qubes ISO.

**Danger:** Warning: Any file you download from the internet could be malicious, even if it appears to come from a trustworthy source. Our philosophy is to distrust the infrastructure. Regardless of how you acquire your Qubes ISO, verify its authenticity before continuing.

Once the ISO has been verified as authentic, you should copy it onto the installation medium of your choice, such as a USB drive, dual-layer DVD, or Blu-ray disc. The size of each Qubes ISO is available on the downloads page by hovering over the download button. The instructions below assume you’ve chosen a USB drive as your medium. If you’ve chosen a different medium, please adapt the instructions accordingly.

**Warning:** Note: There are important security considerations to keep in mind when choosing an installation medium. Advanced users may wish to re-verify their installation media after writing.

**Danger:** Warning: Be careful to choose the correct device when copying the ISO, or you may lose data. We strongly recommended making a full backup before modifying any devices.

## Linux ISO to USB

On Linux, if you choose to use a USB drive, copy the ISO onto the USB device, e.g. using dd:

```
$ sudo dd if=Qubes-RX-x86_64.iso of=/dev/sdY status=progress bs=1048576 conv=fsync
```

Change `Qubes-RX-x86_64.iso` to the filename of the version you’re installing, and change `/dev/sdY` to the correct target device e.g., `/dev/sdc`). Make sure to write to the entire device (e.g., `/dev/sdc`) rather than just a single partition (e.g., `/dev/sdc1`).

## Windows ISO to USB

On Windows, you can use the [Rufus](#) tool to write the ISO to a USB key. Be sure to select “Write in DD Image mode” *after* selecting the Qubes ISO and pressing “START” on the Rufus main window.

**Note:** Note: Using Rufus to create the installation medium means that you won’t be able to choose the “Test this media and install Qubes OS” option mentioned in the example below. Instead, choose the “Install Qubes OS” option.

Rufus 3.8.1580 (Portable)

---

## Drive Properties

Device  
NO\_LABEL (Disk 1) [16 GB]

Boot selection  
Qubes-R4.0.1-x86\_64.iso ☒ SELECT

Persistent partition size  
0 (No persistence)

Partition scheme  
MBR

Target system  
BIOS or UEFI

▼ Show advanced drive properties

---

## Format Options

Volume label  
Qubes-R4.0.1-x86\_64

File system  
FAT32 (Default)

Cluster size  
8192 bytes (Default)

^ Hide advanced format options

☒ Quick format





☒ Create extended label and icon files

☐ Check device for bad blocks 1 pass

---

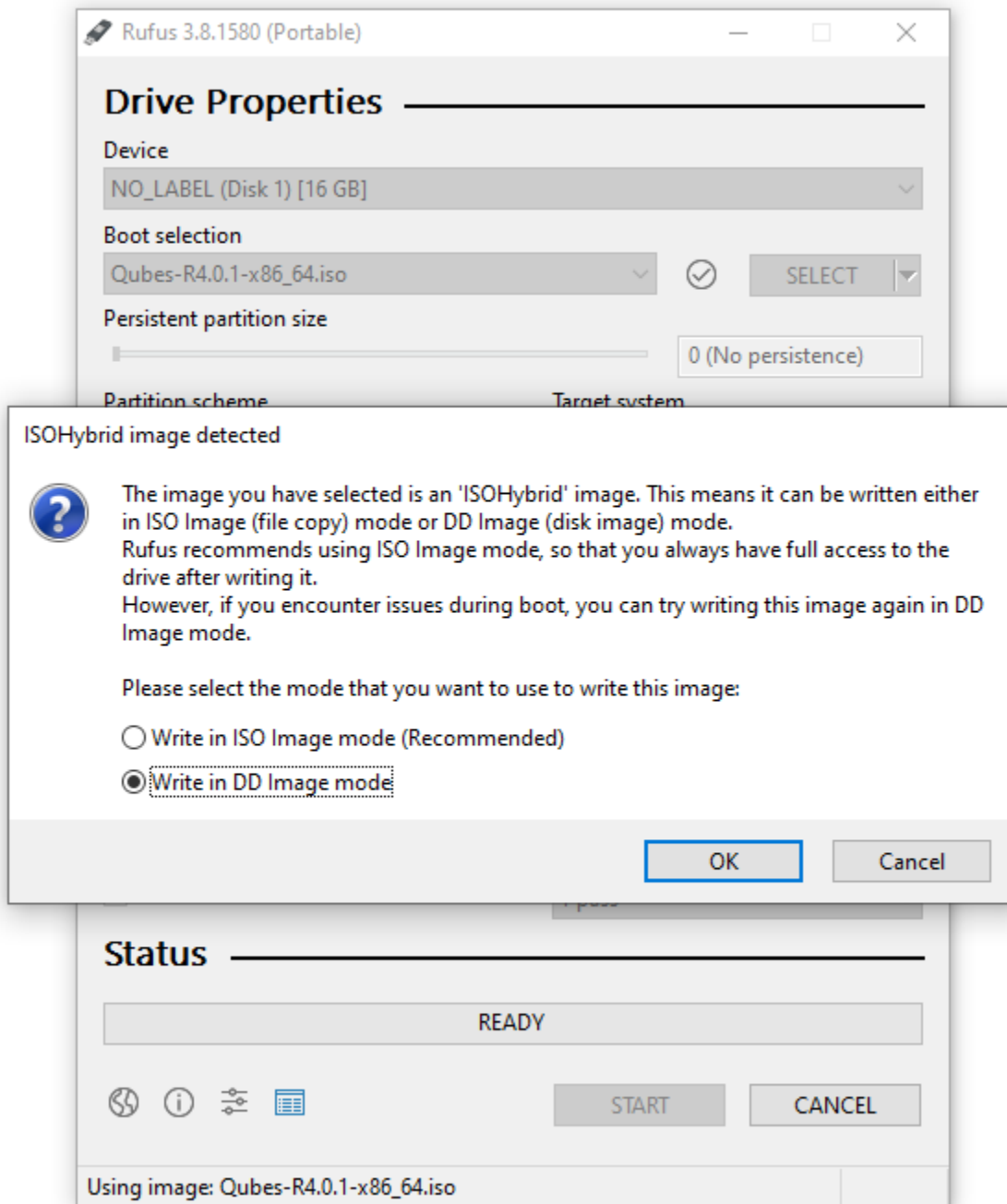
## Status

READY

START CLOSE

Using image: Qubes-R4.0.1-x86\_64.iso



## Installation

This section will demonstrate a simple installation using mostly default settings.

### Getting to the boot screen

“Booting” is the process of starting your computer. When a computer boots up, it first runs low-level software before the main operating system. Depending on the computer, this low-level software is may be called the “BIOS” or “UEFI”.

Since you’re installing Qubes OS, you’ll need to access your computer’s BIOS or UEFI menu so that you can tell it to boot from the USB drive to which you just copied the Qubes installer ISO.

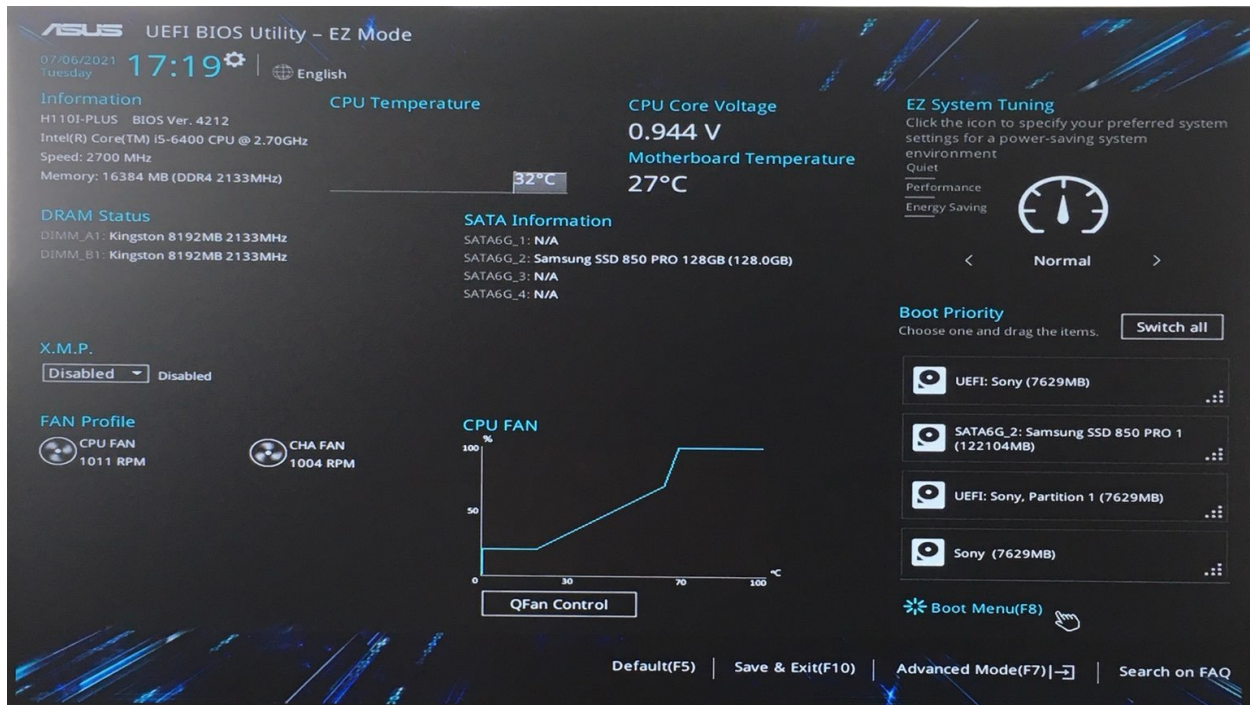
To begin, power off your computer and plug the USB drive into a USB port, but don’t press the power button yet. Right after you press the power button, you’ll have to immediately press a specific key to enter the BIOS or UEFI menu. The key to press varies from brand to brand. Esc, Del, and F10 are common ones. If you’re not sure, you can search the web for <COMPUTER\_MODEL> BIOS key or <COMPUTER\_MODEL> UEFI key (replacing <COMPUTER\_MODEL> with your specific computer model) or look it up in your computer’s manual.

Once you know the key to press, press your computer’s power button, then repeatedly press that key until you’ve entered your computer’s BIOS or UEFI menu. To give you an idea of what you should be looking for, we’ve provided a couple of example photos below.

Here’s an example of what the BIOS menu looks like on a ThinkPad T430:



And here’s an example of what a UEFI menu looks like:



Once you access your computer's BIOS or UEFI menu, you'll want to go to the "boot menu," which is where you tell your computer which devices to boot from. The goal is to tell the computer to boot from your USB drive so that you can run the Qubes installer. If your boot menu lets you select which device to boot from first, simply select your USB drive. (If you have multiple entries that all look similar to your USB drive, and you're not sure which one is correct, one option is just to try each one until it works.) If, on the other hand, your boot menu presents you with a list of boot devices in order, then you'll want to move your USB drive to the top so that the Qubes installer runs before anything else.

Once you're done on the boot menu, save your changes. How you do this depends on your BIOS or UEFI, but the instructions should be displayed right there on the screen or in a nearby tab. (If you're not sure whether you've saved your changes correctly, you can always reboot your computer and go back into the boot menu to check whether it still reflects your changes.) Once your BIOS or UEFI is configured the way you want it, reboot your computer. This time, don't press any special keys. Instead, let the BIOS or UEFI load and let your computer boot from your USB drive. If you're successful in this step, after a few seconds you'll be presented with the Qubes installer screen:



From here, you can navigate the boot screen using the arrow keys on your keyboard. Pressing the “Tab” key will reveal options. You can choose one of three options:

- Install Qubes OS
- Test this media and install Qubes OS
- Troubleshooting

Select the option to test this media and install Qubes OS.

---

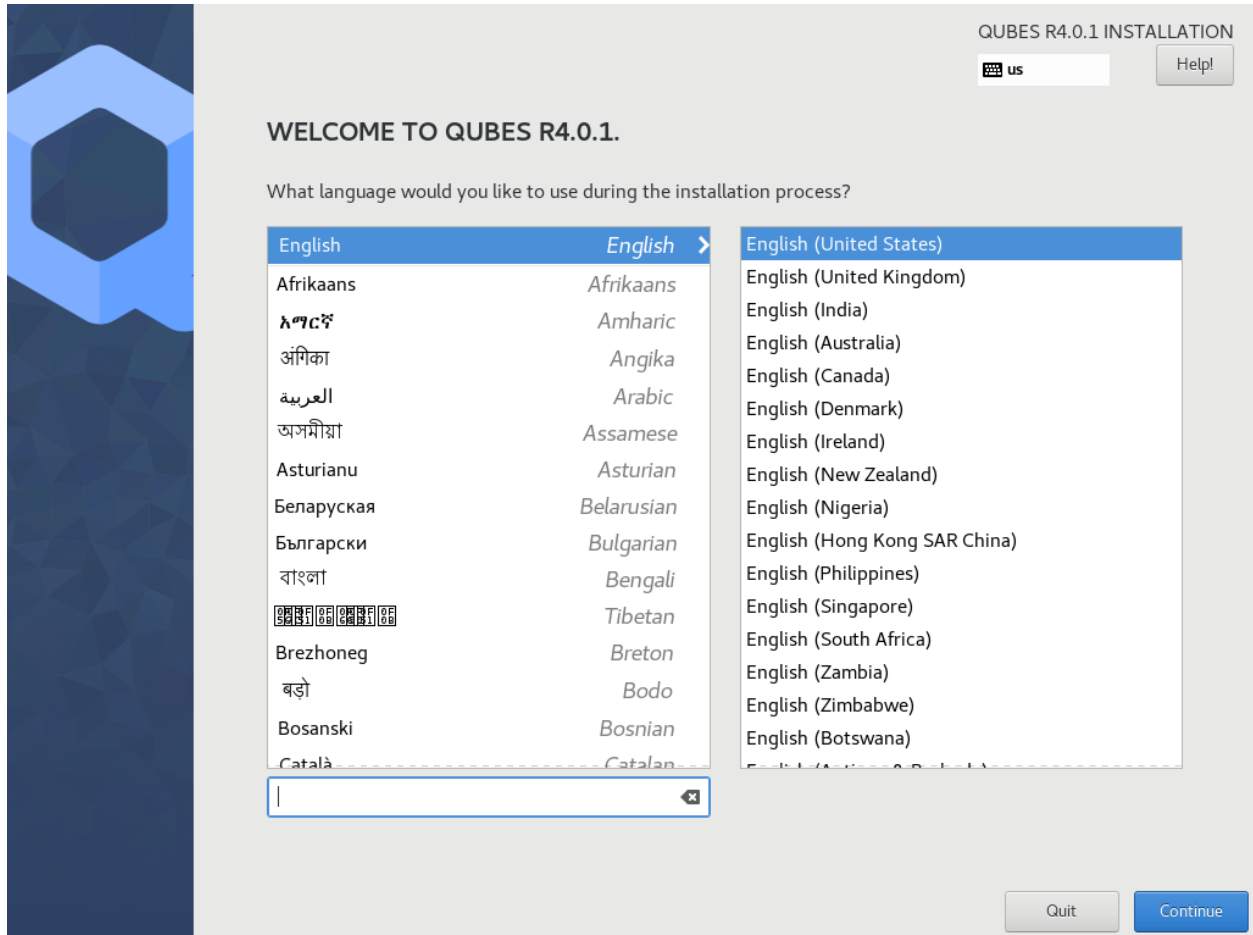
**Note:** Note: If the latest stable release is not compatible with your hardware, you may wish to consider testing a newer release.

---

If the boot screen does not appear, there are several options to troubleshoot. First, try rebooting your computer. If it still loads your currently installed operating system or does not detect your installation medium, make sure the boot order is set up appropriately. The process to change the boot order varies depending on the currently installed system and the motherboard manufacturer. If **Windows 10** is installed on your machine, you may need to follow specific instructions to change the boot order. This may require an [advanced reboot](#).

## The installer home screen

On the first screen, you are asked to select the language that will be used during the installation process. When you are done, select **Continue**.



Prior to the next screen, a compatibility test runs to check whether IOMMU-virtualization is active or not. If the test fails, a window will pop up.



Do not panic. It may simply indicate that IOMMU-virtualization hasn't been activated in the BIOS or UEFI. Return to the [hardware requirements](#) section to learn how to activate it. If the setting is not configured correctly, it means that your hardware won't be able to leverage some Qubes security features, such as a strict isolation of the networking and USB hardware.

If the test passes, you will reach the installation summary screen. The installer loads Xen right at the beginning. If you can see the installer's graphical screen, and you pass the compatibility check that runs immediately afterward, Qubes OS is likely to work on your system!

Like Fedora, Qubes OS uses the Anaconda installer. Those that are familiar with RPM-based distributions should feel at home.

## Installation summary

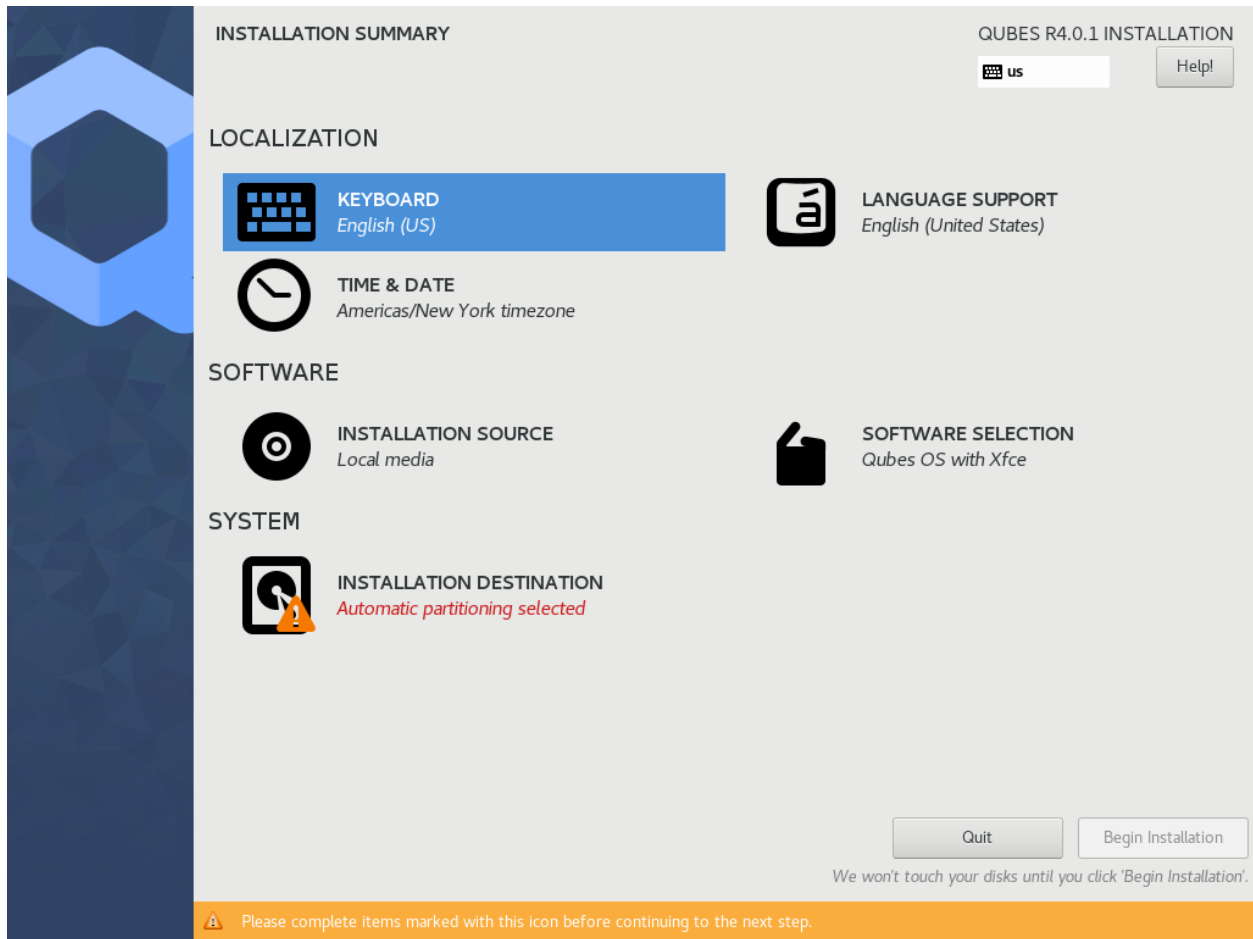
---

**Note:** Did you know? The Qubes OS installer is completely offline. It doesn't even load any networking drivers, so there is no possibility of internet-based data leaks or attacks during the installation process.

---

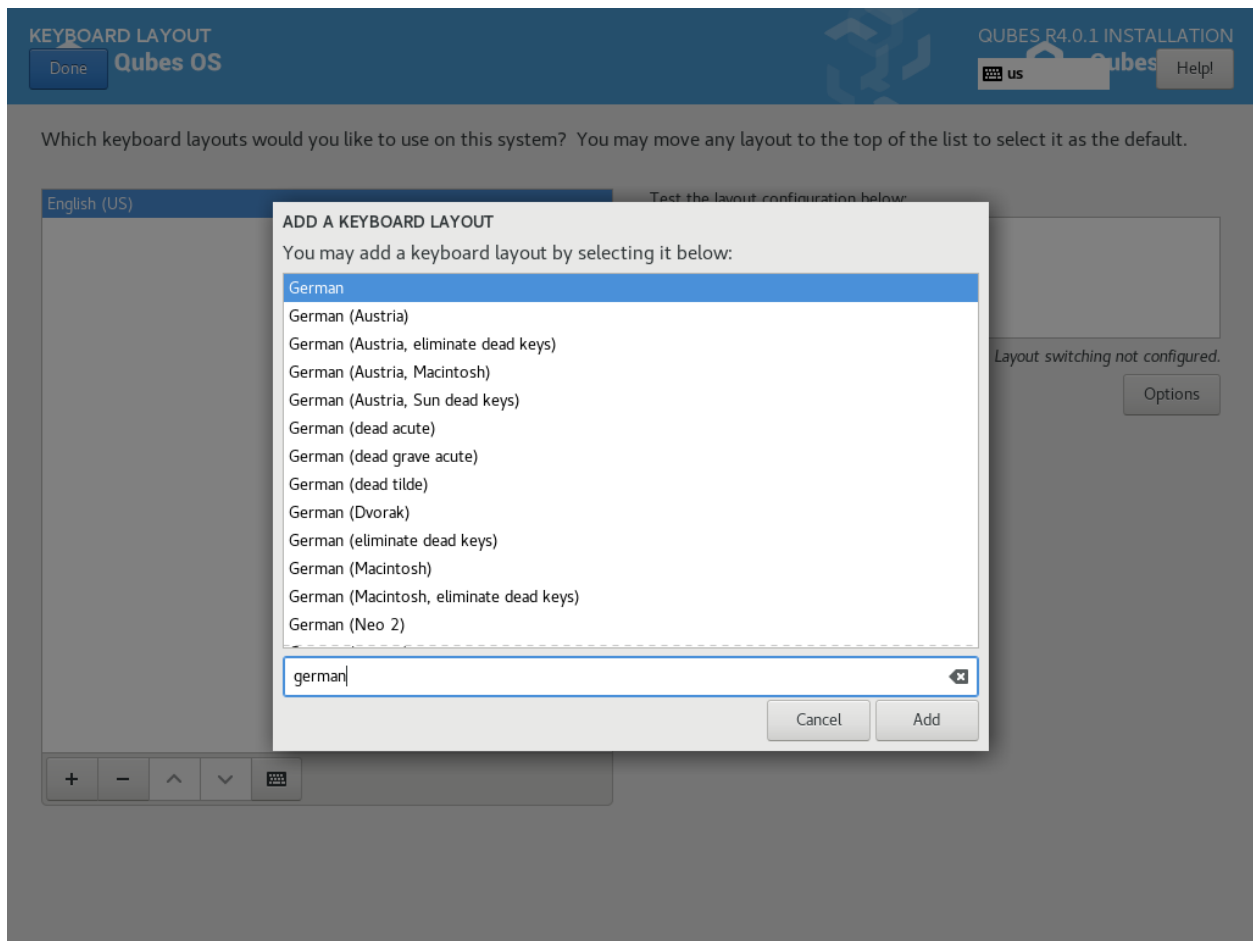
The Installation summary screen allows you to change how the system will be installed and configured, including localization settings. At minimum, you are required to select the storage device on which Qubes OS will be installed.



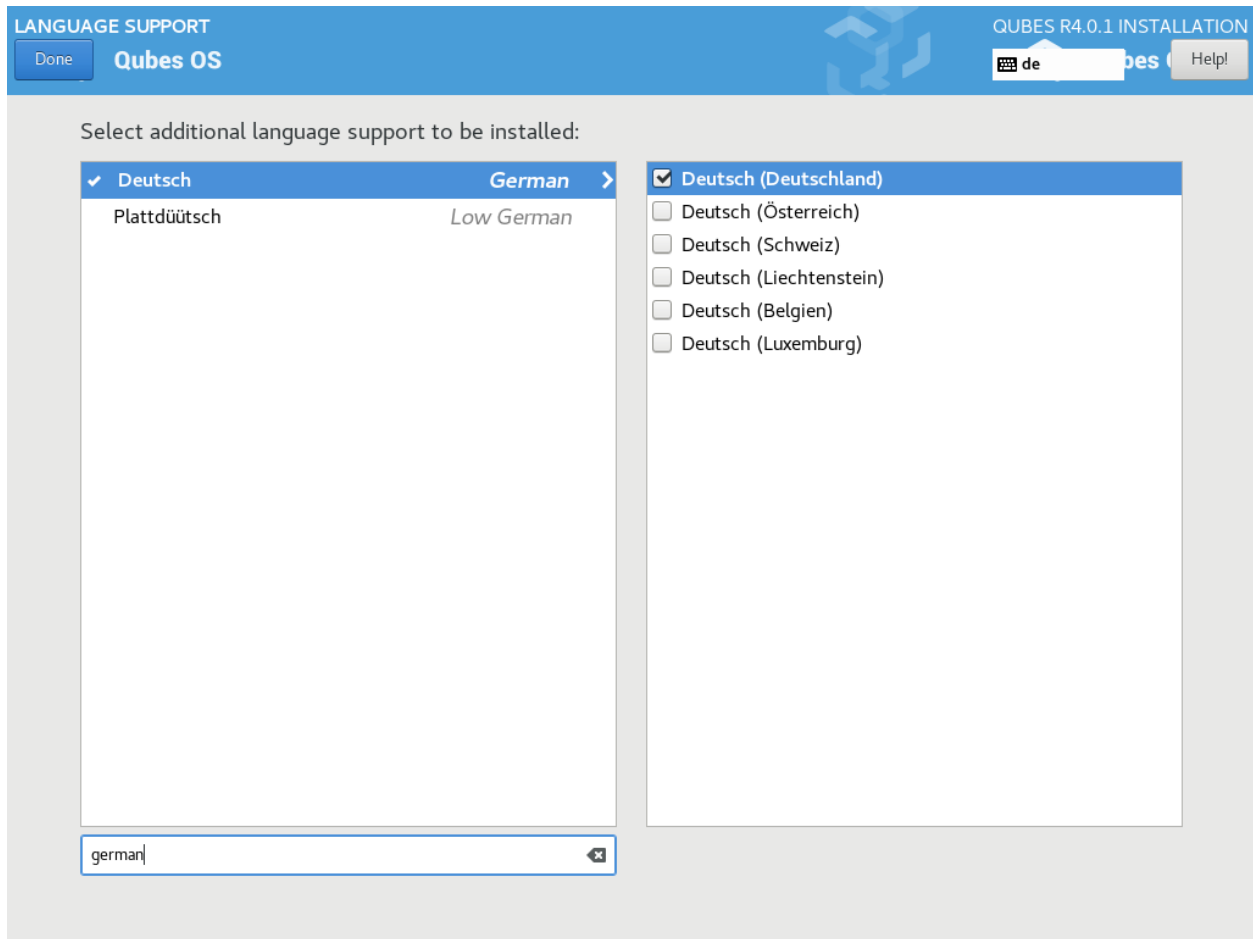


## Localization

Let's assume you wish to add a German keyboard layout. Go to Keyboard Layout, press the "Plus" symbol, search for "German" as indicated in the screenshot and press "Add". If you want it be your default language, select the "German" entry in the list and press the arrow button. Click on "Done" in the upper left corner, and you're ready to go!

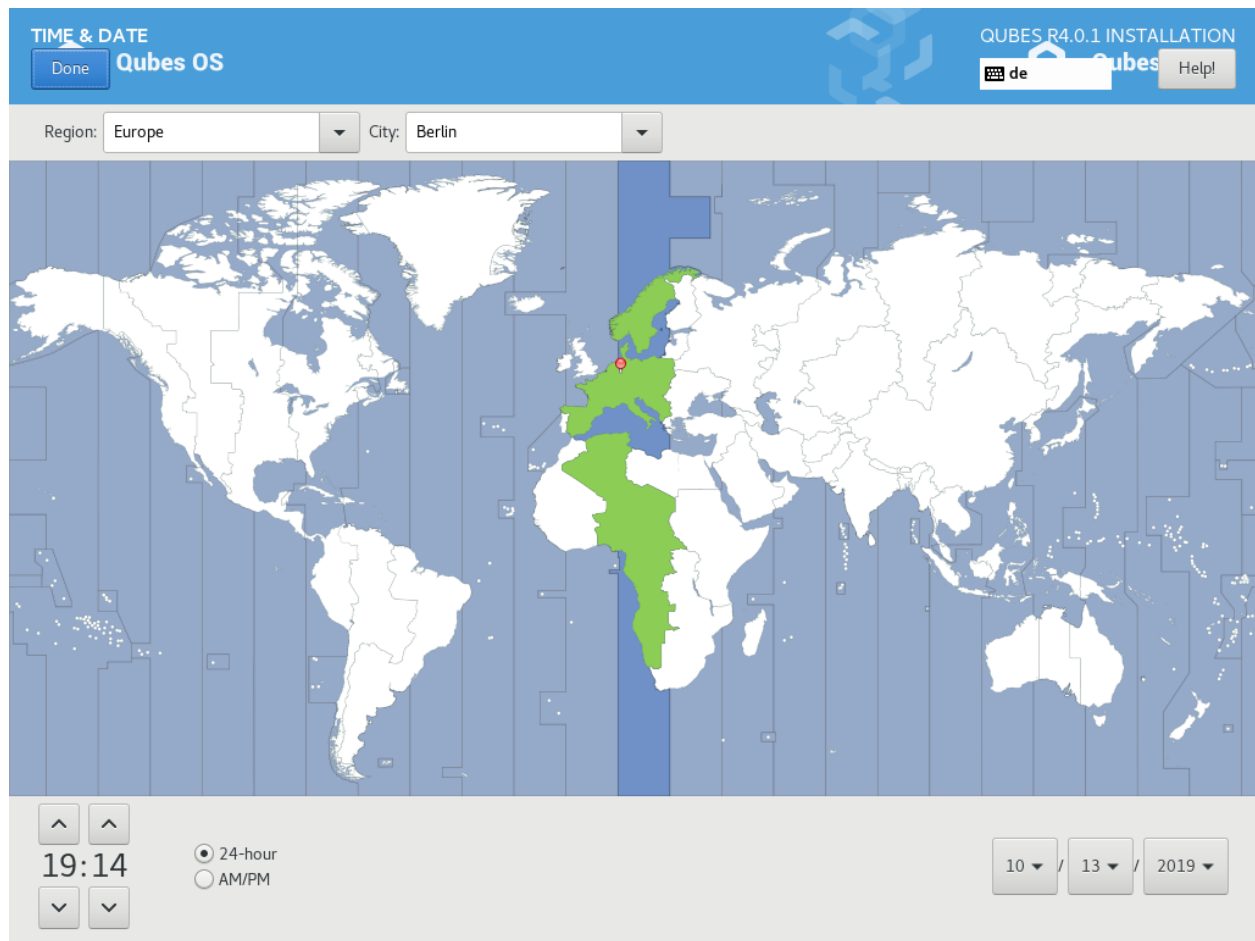


The process to select a new language is similar to the process to select a new keyboard layout. Follow the same process in the “Language Support” entry.

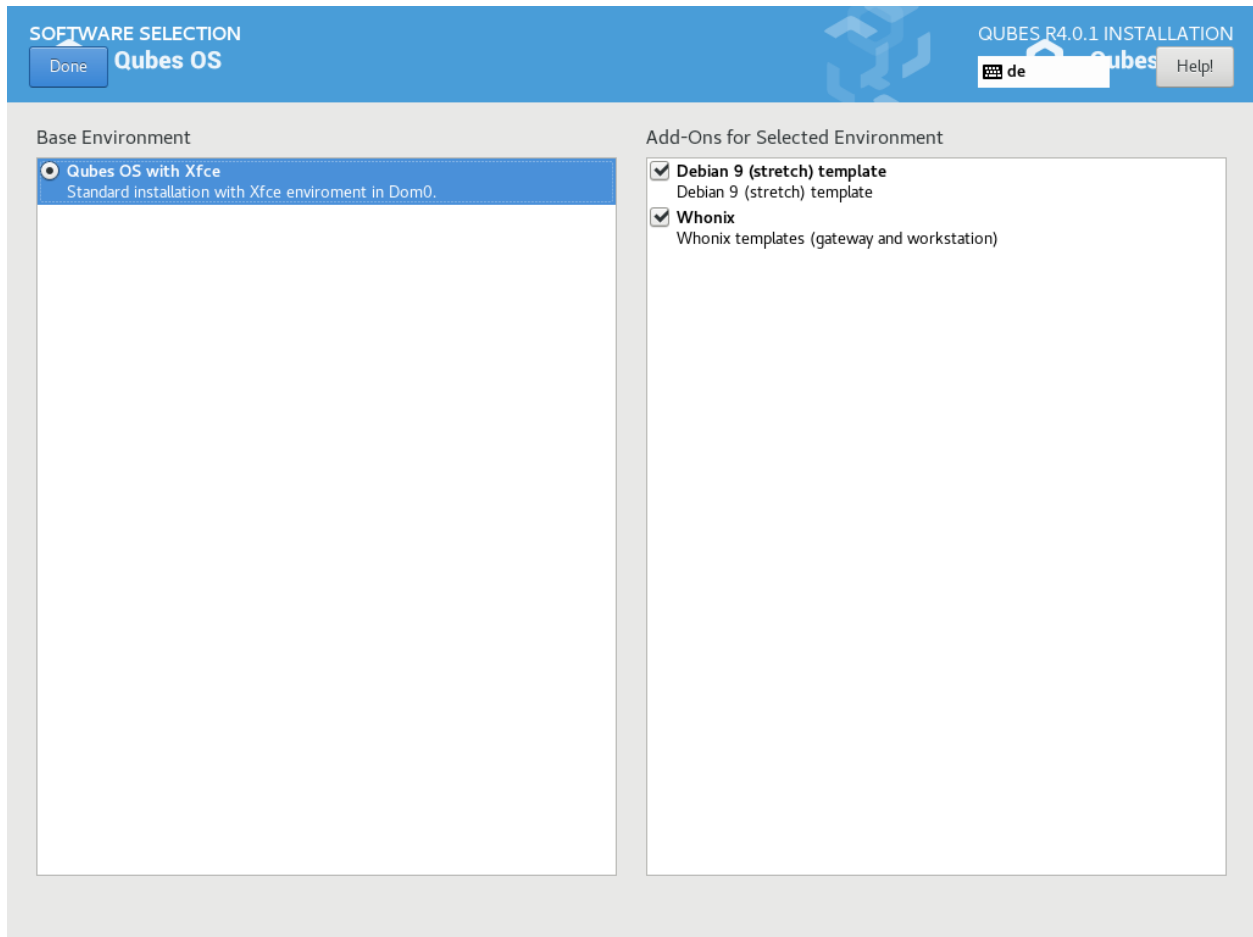


You can have as many keyboard layout and languages as you want. Post-install, you will be able to switch between them and install others.

Don't forget to select your time and date by clicking on the Time & Date entry.



## Software



On the software selection tab, you can choose which software to install in Qubes OS. Two options are available:

- **Debian:** Select this option if you would like to use [Debian](#) qubes in addition to the default Fedora qubes.
- **Whonix:** Select this option if you would like to use [Whonix](#) qubes. Whonix allows you to use [Tor](#) securely within Qubes.

Whonix lets you route some or all of your network traffic through Tor for greater privacy. Depending on your threat model, you may need to install Whonix templates right away.

Regardless of your choices on this screen, you will always be able to install these and other [templates](#) later. If you're short on disk space, you may wish to deselect these options.

By default, Qubes OS comes preinstalled with the lightweight Xfce4 desktop environment. Other desktop environments will be available to you after the installation is completed, though they may not be officially supported (see [advanced topics](#)).

Press **Done** to go back to the installation summary screen.

### Installation destination

Under the System section, you must choose the installation destination. Select the storage device on which you would like to install Qubes OS.

**Danger:** Warning: Be careful to choose the correct installation target, or you may lose data. We strongly recommended making a full backup before proceeding.

Your installation destination can be an internal or external storage drive, such as an SSD, HDD, or USB drive. The installation destination must have a least 32 GiB of free space available.

**Warning:** Note: The installation destination cannot be the same as the installation medium. For example, if you're installing Qubes OS from a USB drive onto a USB drive, they must be two distinct USB drives, and they must both be plugged into your computer at the same time. (Note: This may not apply to advanced users who partition their devices appropriately.)

Installing an operating system onto a USB drive can be a convenient way to try Qubes. However, USB drives are typically much slower than internal SSDs. We recommend a very fast USB 3.0 drive for decent performance. Please note that a minimum storage of 32 GiB is required. If you want to install Qubes OS onto a USB drive, just select the USB device as the target installation device. Bear in mind that the installation process is likely to take longer than it would on an internal storage device.

**INSTALLATION DESTINATION** Qubes OS QUBES R4.0.1 INSTALLATION de Help!

**Device Selection**

Select the device(s) you'd like to install to. They will be left untouched until you click on the main menu's "Begin Installation" button.

**Local Standard Disks**

35 GiB  
ATA QEMU HARDDISK  
sda / 35 GiB free

*Disks left unselected here will not be touched.*

**Specialized & Network Disks**

Add a disk...

*Disks left unselected here will not be touched.*

**Other Storage Options**

**Partitioning**

☒ Automatically configure partitioning. ☐ I will configure partitioning.

☐ I would like to make additional space available.

**Encryption**

☒ Encrypt my data. You'll set a passphrase next.

[Full disk summary and boot loader...](#)

1 disk selected; 35 GiB capacity; 35 GiB free [Refresh...](#)

**Note:** Did you know? By default, Qubes OS uses LUKS/dm-crypt to encrypt everything except the /boot partition.

As soon as you press **Done**, the installer will ask you to enter a passphrase for disk encryption. The passphrase should be complex. Make sure that your keyboard layout reflects what keyboard you are actually using. When you're finished, press **Done**.

**Danger:** Warning: If you forget your encryption passphrase, there is no way to recover it.


**DISK ENCRYPTION PASSPHRASE**

You have chosen to encrypt some of your data. You will need to create a passphrase that you will use to access your data when you start your computer.

Passphrase: [dots] [eye icon]

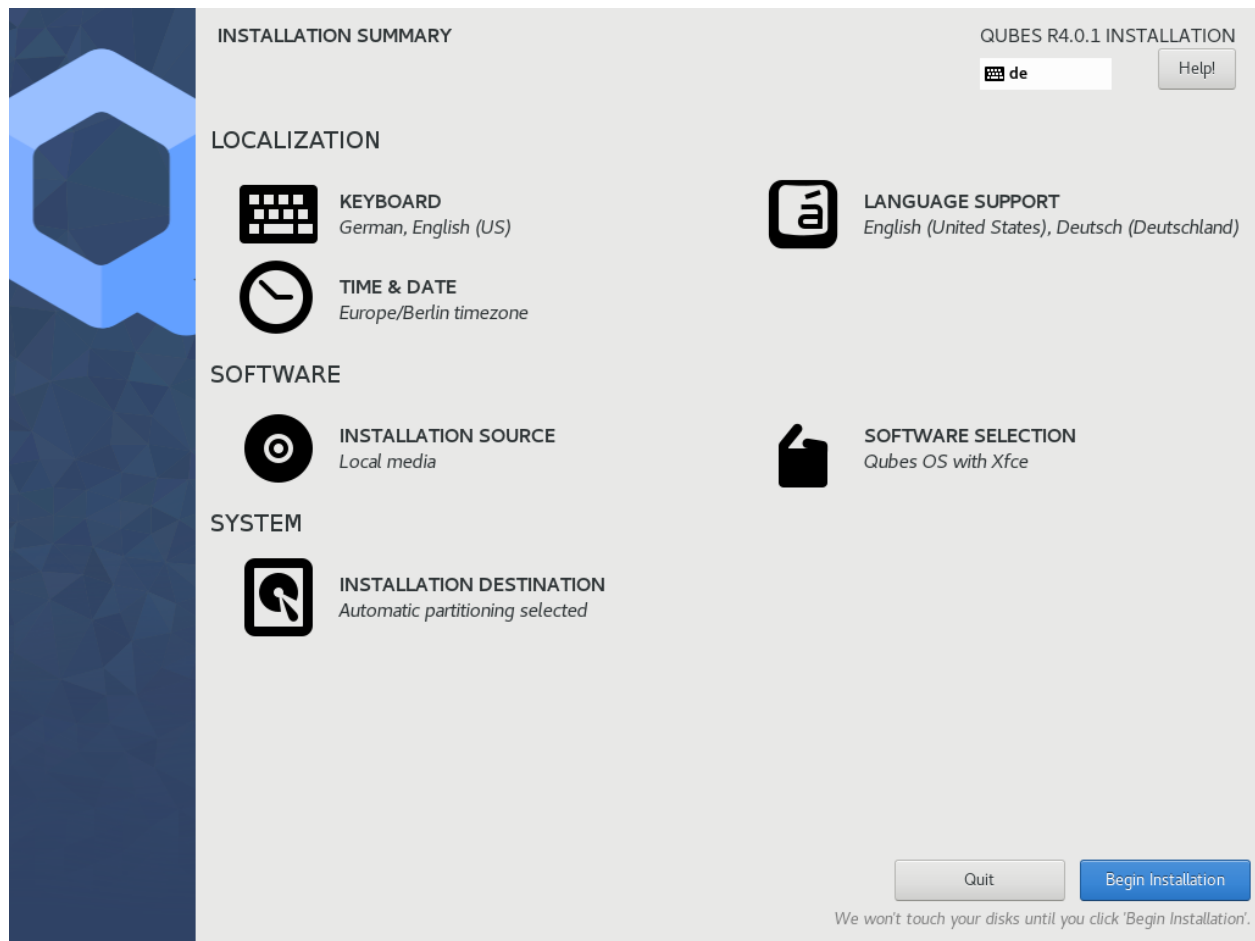
[keyboard icon] de [strength bars] Strong

Confirm: [dots] [eye icon]

 Warning: You won't be able to switch between keyboard layouts (from the default one) when you decrypt your disks after install.

Cancel Save Passphrase

When you're ready, press **Begin Installation**.



## Create your user account

While the installation process is running, you can create your user account. This is what you'll use to log in after disk decryption and when unlocking the screen locker. This is a purely local, offline account in dom0. By design, Qubes OS is a single-user operating system, so this is just for you.

Select **User Creation** to define a new user with administrator privileges and a password. Just as for the disk encryption, this password should be complex. The root account is deactivated and should remain as such.

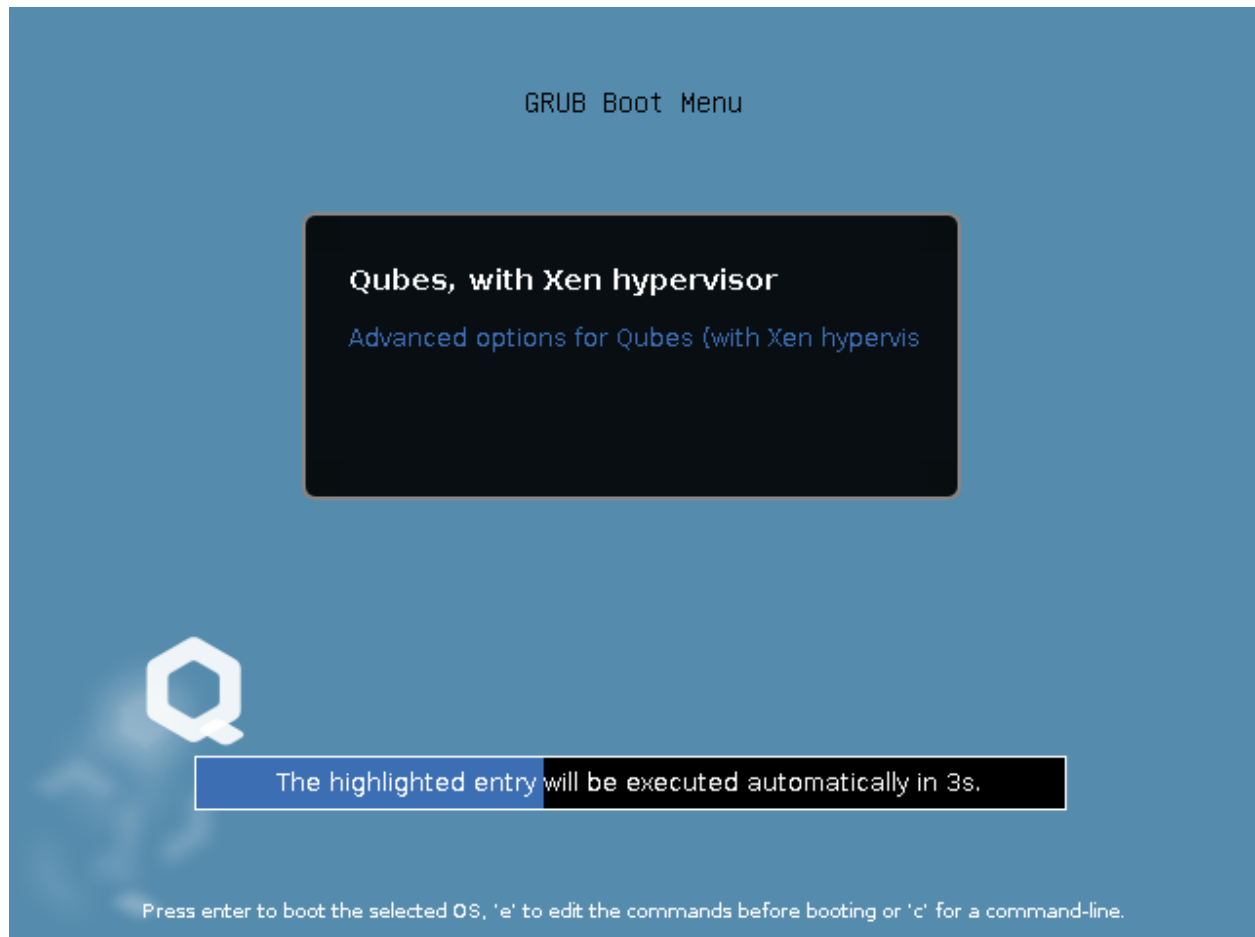


When the installation is complete, press **Reboot**. Don't forget to remove the installation medium, or else you may end up seeing the installer boot screen again.

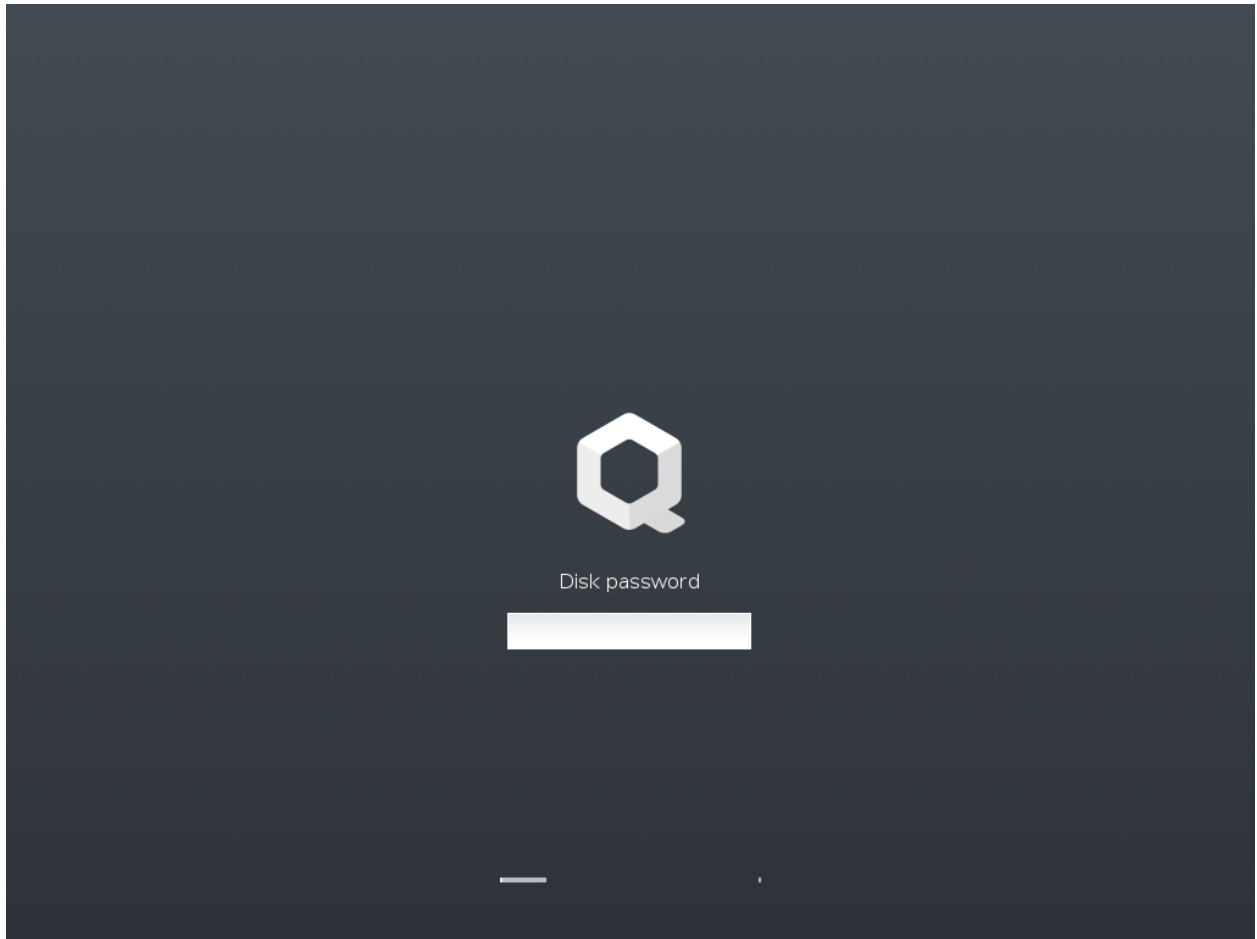
## Post-installation

### First boot

If the installation was successful, you should now see the GRUB menu during the boot process.

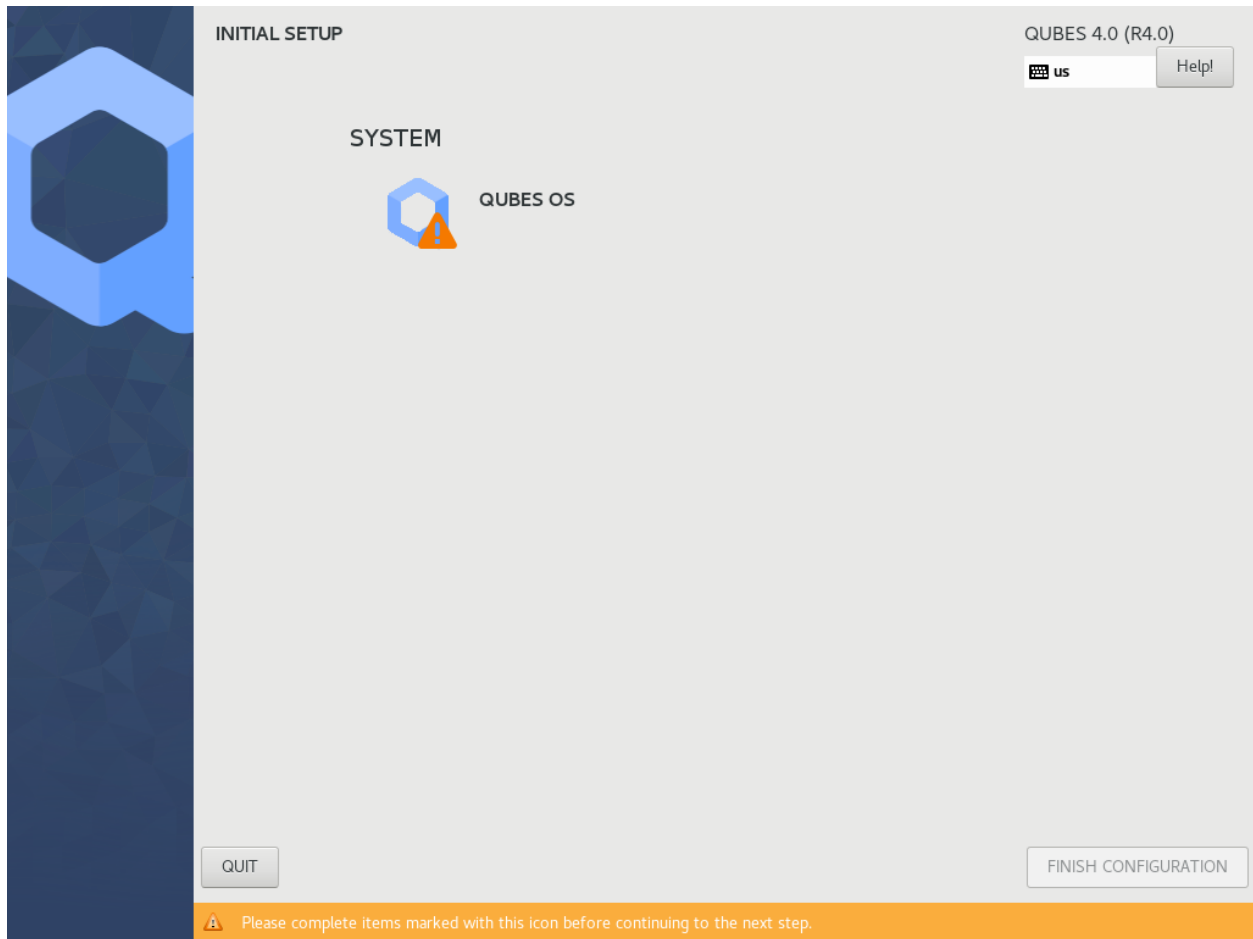


Just after this screen, you will be asked to enter your encryption passphrase.

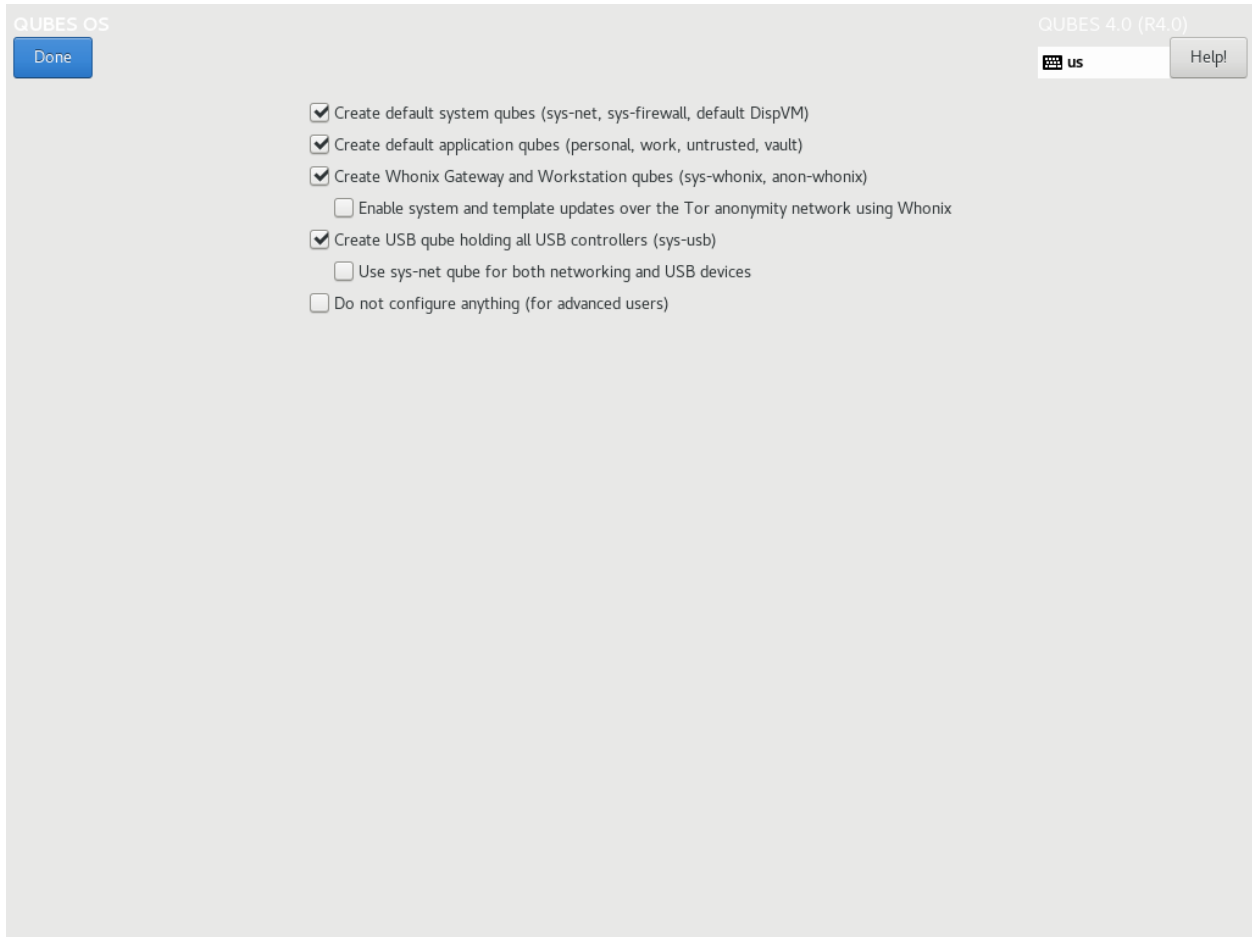


## Initial Setup

You're almost done. Before you can start using Qubes OS, some configuration is needed.



By default, the installer will create a number of qubes (depending on the options you selected during the installation process). These are designed to give you a more ready-to-use environment from the get-go.

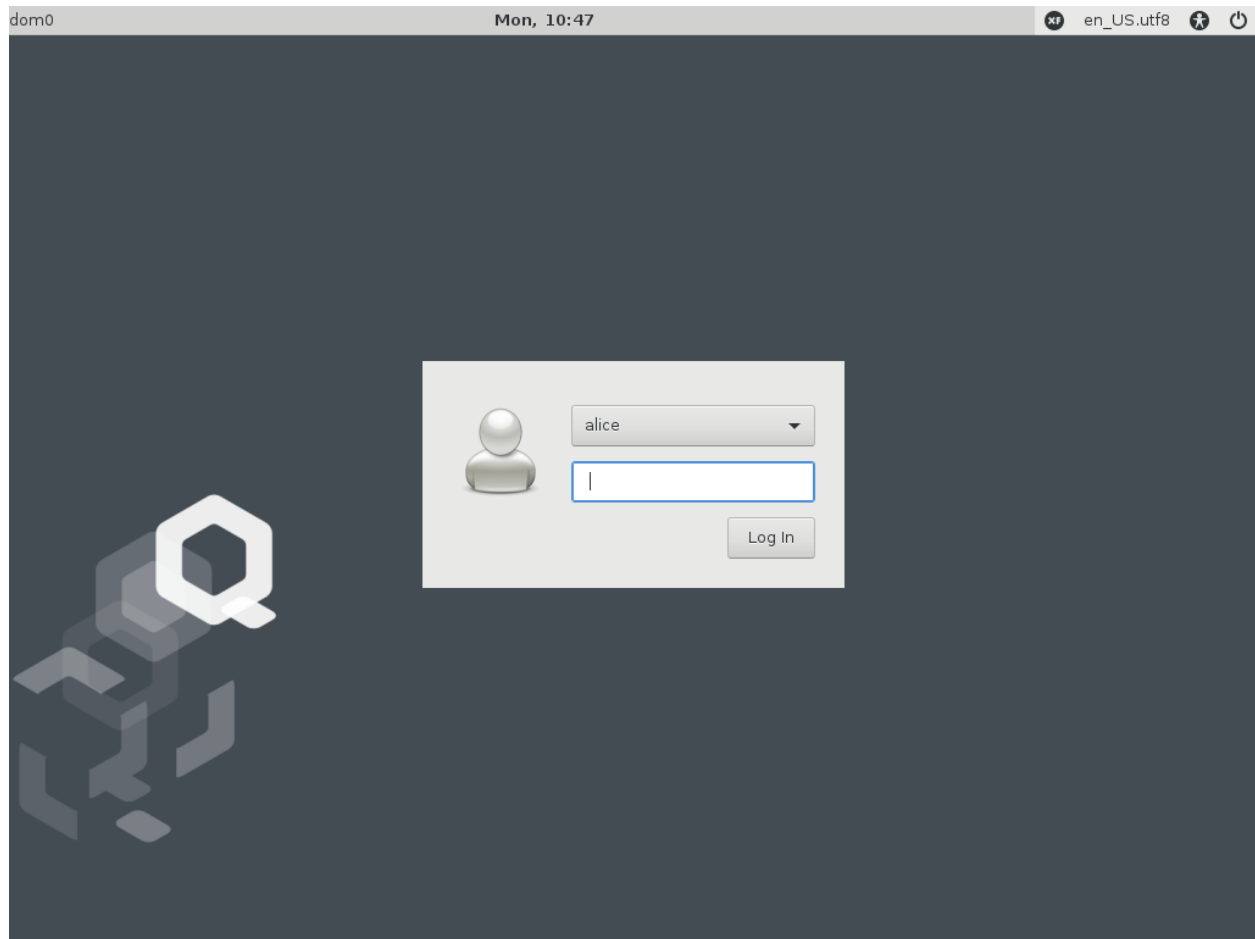


Let's briefly go over the options:

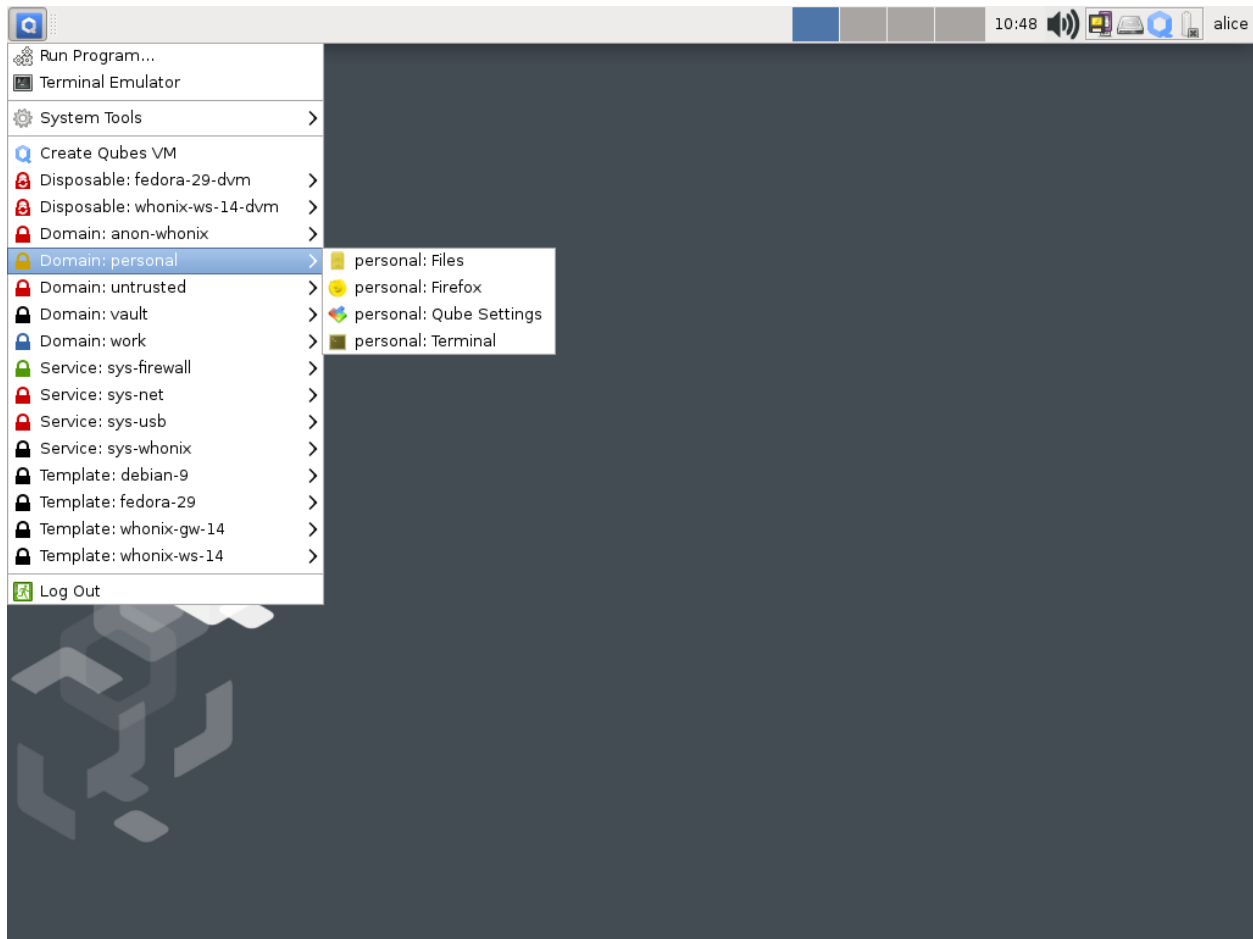
- **Create default system qubes:** These are the core components of the system, required for things like internet access.
- **Create default application qubes:** These are how you compartmentalize your digital life. There's nothing special about the ones the installer creates. They're just suggestions that apply to most people. If you decide you don't want them, you can always delete them later, and you can always create your own.
- **Create Whonix Gateway and Workstation qubes:** If you want to use Whonix, you should select this option.
  - **Enabling system and template updates over the Tor anonymity network using Whonix:** If you select this option, then whenever you install or update software in dom0 or a template, the internet traffic will go through Tor.
- **Create USB qube holding all USB controllers:** Just like the network qube for the network stack, the USB qube isolates the USB controllers.
  - **Use sys-net qube for both networking and USB devices:** You should select this option if you rely on a USB device for network access, such as a USB modem or a USB Wi-Fi adapter.
- **Do not configure anything:** This is for very advanced users only. If you select this option, you'll have to set everything up manually afterward.

When you're satisfied with your choices, press **Done**. This configuration process may take a while, depending on the speed and compatibility of your system.

After the configuration is done, you will be greeted by the login screen. Enter your password and log in.



Congratulations, you are now ready to use Qubes OS!



## Next steps

### Updating

Next, [update](#) your installation to ensure you have the latest security updates. Frequently updating is one of the best ways to remain secure against new threats.

### Security

The Qubes OS Project occasionally issues [Qubes Security Bulletins \(QSBs\)](#) as part of the [Qubes Security Pack \(qubes-secpack\)](#). It is important to make sure that you receive all QSBs in a timely manner so that you can take action to keep your system secure. (While [updating](#) will handle most security needs, there may be cases in which additional action from you is required.) For this reason, we strongly recommend that every Qubes user subscribe to the [qubes-announce](#) mailing list.

In addition to QSBs, the Qubes OS Project also publishes [Canaries](#), XSA summaries, template releases and end-of-life notices, and other items of interest to Qubes users. Since these are not essential for all Qubes users to read, they are not sent to [qubes-announce](#) in order to keep the volume on that list low. However, we expect that most users, especially novice users, will find them helpful. If you are interested in these additional items, we encourage you to subscribe to the [Qubes News RSS feed](#) or join one of our other [venues](#), where these news items are also announced.

For more information about Qubes OS Project security, please see the [security center](#).

### Backups

It is extremely important to make regular backups so that you don't lose your data unexpectedly. The *Qubes backup system* allows you to do this securely and easily.

### Submit your HCL report

Consider giving back to the Qubes community and helping other users by *generating and submitting a Hardware Compatibility List (HCL) report*.

### Get Started

Find out *Getting Started* with Qubes, check out the other *How-To Guides*, and learn about *Templates*.

### Getting help

- We work very hard to make the *documentation* accurate, comprehensive useful and user friendly. We urge you to read it! It may very well contain the answers to your questions. (Since the documentation is a community effort, we'd also greatly appreciate your help in improving it!)
- If issues arise during installation, see the *Installation Troubleshooting* guide.
- If you don't find your answer in the documentation, please see *Help, Support, Mailing Lists, and Forum* for places to ask.
- Please do **not** email individual members of the Qubes team with questions about installation or other problems. Instead, please see *Help, Support, Mailing Lists, and Forum* for appropriate places to ask questions.

### 1.12.5 Installation security

There are several security matters to consider before and during the Qubes installation process.

#### Trusting your hardware

No operating system, not even Qubes, can help you if you're installing it on hardware that is already compromised. This includes CPUs, GPUs, SSDs, HDDs, the motherboard, BIOS/EFI/UEFI, and all relevant firmware. Unfortunately, in today's world of undetectable supply chain attacks, there are no easy solutions. (Tools like *Anti Evil Maid (AEM)* can help with *maintaining* the trustworthiness of your hardware, but not with establishing it in the first place.) Some users have chosen to use tools like *Coreboot*, *Heads*, and *Skulls*.

#### Verifying the Qubes ISO

You should *verify* the PGP signature on your Qubes ISO before you install from it. However, if the machine on which you attempt the verification process is already compromised, it could falsely claim that a malicious ISO has a good signature. Therefore, in order to be certain that your Qubes ISO is trustworthy, you require a trustworthy machine. But how can you be certain *that* machine is trustworthy? Only by using another trusted machine, and so forth. This is a *classic problem*. While various *solutions* have been proposed, the point is that each user must ultimately make a choice about whether to trust that a file is non-malicious.



## Choosing an installation medium

So, after taking some measures to verify its integrity and authenticity, you’ve decided to trust your Qubes ISO. Great! Now you must decide what sort of medium on which to write it so that you can install from it. From a Qubes-specific security perspective, each has certain pros and cons.

### USB drives

Pros:

- Works via USB, including with a *USB qube*.
- Non-fixed capacity. (Easy to find one on which the ISO can fit.)

Cons:

- Rewritable. (If the drive is mounted to a compromised machine, the ISO could be maliciously altered after it has been written to the drive.)
- Untrustworthy firmware. (Firmware can be malicious even if the drive is new. Plugging a drive with rewritable firmware into a compromised machine can also [compromise the drive](#). Installing from a compromised drive could compromise even a brand new Qubes installation.)

### Optical discs

Pros:

- Read-only available. (If you use read-only media, you don’t have to worry about the ISO being maliciously altered after it has been written to the disc. You then have the option of verifying the signature on multiple different machines.)

Cons:

- Fixed capacity. (If the size of the ISO is larger than your disc, it will be inconvenient.)
- Passthrough recording (a.k.a., “burning”) is not supported by Xen. (This mainly applies if you’re upgrading from a previous version of Qubes.) Currently, the only options for recording optical discs (e.g., CDs, DVDs, BRDs) in Qubes are:

1. Use a USB optical drive.
2. Attach a SATA optical drive to a secondary SATA controller, then assign this secondary SATA controller to an app qube.
3. Use a SATA optical drive attached to dom0.

(Option 3 violates the Qubes security model since it entails transferring an untrusted ISO to dom0 in order to burn it to disc, which leaves only the other two options.)

Considering the pros and cons of each, perhaps a USB drive with non-rewritable (or at least cryptographically-signed) firmware and a physical write-protect switch might be the best option.

## 1.12.6 Upgrade guides

These guides are for upgrading from one version of Qubes to another. If you're just looking to update your system while staying on the same version, see [how to update](#).

### Upgrading to R2B1

**Note: Qubes R2 Beta 1 is no longer supported! Please install or upgrade to a newer Qubes R2.**

**Note: This page is kept for historical reasons only! Do not follow the instructions below”**

Existing users of Qubes R1 (but not R1 betas!) can upgrade their systems to the latest R2 beta release by following the procedure below. As usual, it is advisable to backup the system before proceeding with the upgrade

### Upgrade all Template and Standalone VM(s)

By default, in Qubes R1, there is only one template, however users are free to create more templates for special purposes, as well as Standalone VMs. More information on using multiple templates, as well as Standalone VMs, can be found [here](#) and [here](#). The steps described in this section should be repeated in *all* user's Template and Standalone VMs.

1. Open terminal in the template (or standalone VM). E.g. use the Qubes Manager's right-click menu and choose Run Command in VM and type `gnome-terminal` there.
2. Install `qubes-upgrade-vm` package (this package brings in R2 repo definitions and R2 keys)

```
sudo yum install qubes-upgrade-vm
```

3. Proceed with normal update in the template (this should bring in also the R2 packages for the VMs):

```
sudo yum update
```

The installer (yum) will prompt to accept the new Qubes R2 signing key:

```
Importing GPG key 0x0A40E458:
  Userid      : "Qubes OS Release 2 Signing Key"
  Fingerprint: 3f01 def4 9719 158e f862 66f8 0c73 b9d4 0a40 e458
  Package     : qubes-upgrade-vm-1.0-1.fc17.x86_64 (@qubes-vm-current)
  From        : /etc/pki/rpm-gpg/RPM-GPG-KEY-upgrade-qubes-2-primary
  Is this ok [y/N]:
```

If you see (as is the case on the “screenshot” above) that the new key was imported from a local filesystem (`/etc/pki/rpm-gpg/...`) you can safely accept the key, without checking its fingerprint. This is because there were only two ways for such a key to make it to your template's filesystem:

- via a legitimate RPM package previously installed (in our case it was the `qubes-upgrade-vm` RPM). Such an RPM must have been signed by one of the keys you decided to trust previously, by default this would be either via the Qubes R1 signing key, or Fedora 17 signing key.
  - via system compromise or via some illegal RPM package (e.g. Fedora released package pretending to bring new Firefox). In that case, however, your VM is already compromised, and it careful checking of the new R2 key would not change this situation to any better one. The game is lost for this VM anyway (and all VMs based on this template).
1. Shut down the VM.

## Upgrade Dom0

Be sure to do steps described in this section after *all* your template and standalone VMs got updated as described in the section above.

1. Open terminal in Dom0. E.g. Start->System Settings->Konsole.
2. Upgrade the qubes-release package to the latest version which brings in new repo definitions and R2 signing keys:

```
sudo qubes-dom0-update qubes-release
```

This should install qubes-release-1-6 in your Dom0.

3. Install R2 packages:

```
sudo qubes-dom0-update --releasever=2
```

4. Reboot your system. Please note that if you use Anti Evil Maid, then it won't be able to unseal the passphrase this time, because the Xen, kernel, and initramfs binaries have changed. Once the system boots up again, you could reseat your Anti Evil Maid's passphrase to the new configuration. Please consult Anti Evil Maid documentation for explanation on how to do that.

## Upgrading to R2B2

Existing users of Qubes R1 (but not R1 betas!) can upgrade their systems to the latest R2 beta release by following the procedure below. As usual, it is advisable to backup the system before proceeding with the upgrade. While it is possible to upgrade the system **it is strongly recommended to reinstall it**. You will preserve all your data and settings thanks to *backup and restore tools*.

## Upgrade all Template and Standalone VM(s)

**If you have already R2 Beta1 installed, follow standard template update procedure (e.g. "Update VM" button in Qubes Manager) and skip the rest of this section**

By default, in Qubes R1, there is only one template, however users are free to create more templates for special purposes, as well as Standalone VMs. More information on using multiple templates, as well as Standalone VMs, can be found [here](#) and [here](#). The steps described in this section should be repeated in *all* user's Template and Standalone VMs.

1. Open terminal in the template (or standalone VM). E.g. use the Qubes Manager's right-click menu and choose Run Command in VM and type `gnome-terminal` there.
2. Install qubes-upgrade-vm package (this package brings in R2 repo definitions and R2 keys)

```
sudo yum install qubes-upgrade-vm
```

3. Proceed with normal update in the template (this should bring in also the R2 packages for the VMs):

```
sudo yum update
```

The installer (yum) will prompt to accept the new Qubes R2 signing key:

```
Importing GPG key 0x0A40E458:
Userid      : "Qubes OS Release 2 Signing Key"
Fingerprint: 3f01 def4 9719 158e f862 66f8 0c73 b9d4 0a40 e458
```

(continues on next page)

(continued from previous page)

```
Package      : qubes-upgrade-vm-1.0-1.fc17.x86_64 (@qubes-vm-current)
From         : /etc/pki/rpm-gpg/RPM-GPG-KEY-upgrade-qubes-2-primary
Is this ok [y/N]:
```

If you see (as is the case on the “screenshot” above) that the new key was imported from a local filesystem (/etc/pki/rpm-gpg/...) you can safely accept the key, without checking its fingerprint. This is because there were only two ways for such a key to make it to your template’s filesystem:

- via a legitimate RPM package previously installed (in our case it was the `qubes-upgrade-vm` RPM). Such an RPM must have been signed by one of the keys you decided to trust previously, by default this would be either via the Qubes R1 signing key, or Fedora 17 signing key.
- via system compromise or via some illegal RPM package (e.g. Fedora released package pretending to bring new Firefox). In that case, however, your VM is already compromised, and it careful checking of the new R2 key would not change this situation to any better one. The game is lost for this VM anyway (and all VMs based on this template).

1. Shut down the VM.

## Installing new template

Qubes R2 Beta2 brings new fedora-18-x64 template (based on Fedora 18). You can install it from Qubes installation DVD. Insert installation DVD into your drive and issue following commands:

```
$ sudo -s
# mkdir -p /mnt/cdrom
# mount /dev/cdrom /mnt/cdrom # you can also use ISO image instead of /dev/cdrom; then,
→ add -o loop option
# yum install /mnt/cdrom/Packages/q/qubes-template-fedora-18-x64*rpm
# umount /mnt/cdrom
```

If you already have fedora-17-x64, you can also upgrade it to fedora-18-x64 following [standard Fedora upgrade procedure](#) (only “yum” method will work in Qubes VM).

## Upgrade Dom0

Be sure to do steps described in this section after *all* your template and standalone VMs got updated as described in the section above.

1. Open terminal in Dom0. E.g. Start->System Settings->Konsole.
2. Upgrade the `qubes-release` package to the latest version which brings in new repo definitions and R2 signing keys:

```
sudo qubes-dom0-update qubes-release
```

This should install `qubes-release-1-6` in your Dom0.

3. Install R2 upgrade package:

```
sudo qubes-dom0-update --releasever=1 qubes-dist-upgrade
```

4. Start upgrade process:

```
sudo qubes-dist-upgrade
```

5. Follow instructions on screen, first stage of upgrade should end up with reboot request.
6. Reboot your system, ensure that you choose “Qubes Upgrade” boot option.
7. When system starts up, login and start start

```
sudo qubes-dist-upgrade
```

again. This will start second stage of upgrade, here most packages will be upgraded, so this will take a while.

8. You will be prompted to install new bootloader. If you haven’t changed anything in this matter from initial installation, just accept the default.
9. Reboot your system. System shutdown may hung because some running system components no longer match that installed on disk; just wait a few minutes and hard reset the system in such case.
10. This is end of upgrade process, you should now have Qubes R2 system.

Please note that if you use Anti Evil Maid, then it won’t be able to unseal the passphrase this time, because the Xen, kernel, and initramfs binaries have changed. Once the system boots up again, you could reseat your Anti Evil Maid’s passphrase to the new configuration. Please consult Anti Evil Maid documentation for explanation on how to do that.

## Upgrading to R2B3

Current Qubes R2 Beta 2 (R2B2) systems can be upgraded in-place to the latest R2 Beta 3 (R2B3) release by following the procedure below. However, upgrading in-place is riskier than performing a clean installation, since there are more things which can go wrong. For this reason, **we strongly recommended that users perform a [clean installation](#) of Qubes R2 Beta 3.**

**Before attempting either an in-place upgrade or a clean installation, we strongly recommend that users back up the system by using the built-in [backup tool](#) .**

Experienced users may be comfortable accepting the risks of upgrading in-place. Such users may wish to first attempt an in-place upgrade. If nothing goes wrong, then some time and effort will have been saved. If something does go wrong, then the user can simply perform a clean installation, and no significant loss will have occurred (as long as the user [backed up](#) correctly!).

## Upgrade all Template and Standalone VM(s)

By default, in Qubes R2, there is only one template, however users are free to create more templates for special purposes, as well as Standalone VMs. More information on using multiple templates, as well as Standalone VMs, can be found [here](#). The steps described in this section should be repeated in *all* user’s Template and Standalone VMs.

It is critical to complete this step **before** proceeding to dom0 upgrade. Otherwise you will most likely ends with unusable system.

1. Open terminal in the template (or standalone VM). E.g. use the Qubes Manager’s right-click menu and choose Run Command in VM and type `gnome-terminal` there.
2. Proceed with normal update in the template:

```
sudo yum update
```

3. Ensure that you’ve got qubes-core-vm package version 2.1.13-3.fc18:

```
rpm -q qubes-core-vm
```

4. Update the system to R2 beta3 packages:

```
sudo yum --enablerepo=qubes-vm-r2b3-current update
```

5. **Do not** shutdown the VM.

### Upgrading dom0

Be sure to do steps described in this section after *all* your template and standalone VMs got updated as described in the section above. Also make sure you haven't shutdown any of: netvm, firewallvm, fedora-18-x64 (or to be more precise: template which your netvm and firewallvm is based on).

1. Open terminal in Dom0. E.g. Start->System Settings->Konsole.
2. Upgrade the qubes-release package to the latest version which brings in new repo definitions and R2 signing keys:

```
sudo qubes-dom0-update qubes-release
```

This should install qubes-release-2-3.1 in your Dom0.

3. Upgrade dom0 to R2 beta3:

```
sudo qubes-dom0-update --enablerepo=qubes-dom0-r2b3-current
```

4. If above step completed successfully you should have qubes-core-dom0 at least 2.1.34. If not, repeat above step with additional `--clean` option.
5. Now is the time to shutdown all the VMs:

```
qvm-shutdown --all --wait
```

6. Reboot the system.

Please note that if you use Anti Evil Maid, then it won't be able to unseal the passphrase this time, because the Xen, kernel, and initramfs binaries have changed. Once the system boots up again, you could reseal your Anti Evil Maid's passphrase to the new configuration. Please consult Anti Evil Maid documentation for explanation on how to do that.

### Upgrading to R2

Current Qubes R2 Beta 3 (R2B3) systems can be upgraded in-place to the latest R2 (R2) release by following the procedure below.

**Before attempting either an in-place upgrade or a clean installation, we strongly recommend that users back up the system by using the built-in *backup tool*.**

## Upgrade Template and Standalone VM(s)

- Qubes R2 comes with new template based on Fedora 20. You can upgrade existing template according to procedure described [here](#).
- **It also possible to download a new Fedora 20-based template from our repositories.** To do this please first upgrade the Dom0 distro as described in the section below.

While technically it is possible to use old Fedora 18 template on R2, it is strongly recommended to upgrade all the templates and Standalone VMs, because Fedora 18 no longer receive security updates.

By default, in Qubes R2, there is only one template, however users are free to create more templates for special purposes, as well as Standalone VMs. If more than one template and/or Standalone VMs are used, then it is recommended to upgrade/replace all of them. More information on using multiple templates, as well as Standalone VMs, can be found [here](#).

## Upgrading dom0

Note that dom0 in R2 is based on Fedora 20, in contrast to Fedora 18 in previous release, so this operation will upgrade a lot of packages.

1. Open terminal in Dom0. E.g. Start->System Settings->Konsole.
2. Install all the updates for Dom0:

```
sudo qubes-dom0-update
```

After this step you should have `qubes-release-2-5` in your Dom0. Important: if you happen to have `qubes-release-2-6*` then you should downgrade to `qubes-release-2-5`! The `qubes-release-2-6*` packages have been uploaded to the testing repos and were kept there for a few hours, until we realized they bring incorrect repo definitions and so we removed them and also have changed the update procedure a bit (simplifying it).

1. Upgrade dom0 to R2:

Note: be sure that the VM used as a update-downloading-vm (by default its the firewallvm based on the default template) has been updated to the latest Qubes packages, specifically `qubes-core-vm-2.1.33` or later. This doesn't imply that the VM must already be upgraded to fc20 – for Dom0 upgrade we could still use an fc18-based VM (updatevm) it is only important to install the latest Qubes packages there.

```
sudo qubes-dom0-update qubes-dom0-dist-upgrade
sudo qubes-dom0-update
```

1. If above step completed successfully you should have `qubes-release-2-9` or later. If not, repeat above step with additional `--clean` option.
- 4a. If you chose not to upgrade your fc18 templates, but instead to download our new fc20-based template you should now be able to do that by simply typing:

```
sudo qubes-dom0-update qubes-template-fedora-20-x64
```

1. Reboot the system.

Please note that if you use Anti Evil Maid, then it won't be able to unseal the passphrase this time, because the Xen, kernel, and initramfs binaries have changed. Once the system boots up again, you could reseat your Anti Evil Maid's passphrase to the new configuration. Please consult Anti Evil Maid documentation for explanation on how to do that.

## Upgrading to R3.0

**This instruction is highly experimental, the official way to upgrade from R2 is to backup the data and reinstall the system. Use at your own risk!**

Current Qubes R3.0 (R3.0) systems can be upgraded in-place to the latest R3.0 by following the procedure below. However, upgrading in-place is riskier than performing a clean installation, since there are more things which can go wrong. For this reason, **we strongly recommended that users perform a *clean installation* of Qubes R3.0.**

**Before attempting either an in-place upgrade or a clean installation, we strongly recommend that users back up the system by using the built-in *backup tool* .**

Experienced users may be comfortable accepting the risks of upgrading in-place. Such users may wish to first attempt an in-place upgrade. If nothing goes wrong, then some time and effort will have been saved. If something does go wrong, then the user can simply perform a clean installation, and no significant loss will have occurred (as long as the user *backed up* correctly!).

## Upgrade all Template and Standalone VM(s)

By default, in Qubes R2, there is only one template, however users are free to create more templates for special purposes, as well as Standalone VMs. More information on using multiple templates, as well as Standalone VMs, can be found [here](#). The steps described in this section should be repeated in **all** user's Template and Standalone VMs.

It is critical to complete this step **before** proceeding to dom0 upgrade. Otherwise you will most likely end with unusable system.

### Upgrade Fedora template:

1. Open terminal in the template (or standalone VM). E.g. use the Qubes Manager's right-click menu and choose Run Command in VM and type `gnome-terminal` there.
2. Install `qubes-upgrade-vm` package:

```
sudo yum install qubes-upgrade-vm
```

3. Proceed with normal update in the template:

```
sudo yum update
```

You'll need to accept "Qubes Release 3 Signing Key" - it is delivered by signed `qubes-upgrade-vm` package (verify that the message is about local file), so you don't need to manually verify it.

4. Shutdown the template.

### Upgrade Debian template:

1. Open terminal in the template (or standalone VM). E.g. use the Qubes Manager's right-click menu and choose Run Command in VM and type `gnome-terminal` there.
2. Update repository definition:

```
sudo cp /etc/apt/sources.list.d/qubes-r2.list
/etc/apt/sources.list.d/qubes-r3-upgrade.list
sudo sed -i 's/r2/r3.0/' /etc/apt/sources.list.d/qubes-r3-upgrade.list
```



3. Proceed with normal update in the template:

```
sudo apt-get update
sudo apt-get dist-upgrade
```

There will be some error messages during the process, but our tests does not revealed any negative consequences. Update of `qubesdb-vm` package will restart the service, which will fail (after 3min timeout), but you can ignore this problem for now. After completing the whole upgrade the service will be properly restarted.

4. Shutdown the template.

## Upgrading dom0

Be sure to do steps described in this section after *all* your template and standalone VMs got updated as described in the section above. Also make sure you haven't shutdown any of: `netvm`, `firewallvm` - you will not be able to start them again.

1. Open terminal in Dom0. E.g. Start->System Settings->Konsole.
2. Upgrade the `qubes-release` package to the latest version which brings in new repo definitions and R2 signing keys:

```
sudo qubes-dom0-update qubes-release
```

This should install `qubes-release-2-12` in your Dom0.

3. Upgrade dom0 to R3.0:

```
sudo qubes-dom0-update --releasever=3.0
```

After this step, until you reboot the system, most of the `qvm-*` tools will not work.

4. If above step completed successfully you should have `qubes-core-dom0` at least 3.0.8. If not, repeat above step with additional `--clean` option.
5. Enable Xen services:

```
sudo systemctl enable xenconsole.service xenstored.service
```

6. Reboot the system. It may happen that the system hang during the reboot. Hard reset the system in such case, all the filesystems are unmounted at this stage.

Please note that if you use Anti Evil Maid, then it won't be able to unseal the passphrase this time, because the Xen, kernel, and `initramfs` binaries have changed. Once the system boots up again, you could reseat your Anti Evil Maid's passphrase to the new configuration. Please consult Anti Evil Maid documentation for explanation on how to do that.

Now, when you have dom0 upgraded, you can install new templates from Qubes R3.0 repositories. Especially Fedora 21 - default Qubes R3.0 template:

```
sudo qubes-dom0-update qubes-template-fedora-21
```

## Upgrading template on already upgraded dom0

If for some reason you did not upgrade all the templates and standalone VMs before upgrading dom0, you can still do this, but it will be more complicated. This can be the case when you restore backup done on Qubes R2.

When you start R2 template/standalone VM on R3.0, there will be some limitations:

1. qrexec will not connect (you will see an error message during VM startup)
2. GUI will not connect - you will not see any VM window
3. VM will not be configured - especially it will not have network access

Because of above limitations, you will need to configure some of those manually. The instruction assumes the VM name is `custom-template`, but the same instructions can be applied to a standalone VM.

1. Check the VM network parameters, you will need them later:

```
[user@dom0 ~]$ qvm-ls -n custom-template
-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪+-----+-----+
      name | on | state | updbl | type |      netvm | label |      ip
↪| ip back | gateway/DNS |
-----+-----+-----+-----+-----+-----+-----+-----+-----+
↪+-----+-----+
[custom-template] |   | Halted | Yes | Tpl | *firewallvm | black | 10.137.1.53
↪|      n/a | 10.137.1.1 |
```

2. Start the VM from command line:

```
[user@dom0 ~]$ qvm-start custom-template
--> Loading the VM (type = template)...
--> Starting Qubes DB...
--> Setting Qubes DB info for the VM...
--> Updating firewall rules...
--> Starting the VM...
--> Starting the qrexec daemon...
Waiting for VM's qrexec agent.....
↪.....Cannot connect to 'custom-template' qrexec agent for 60 seconds, giving up
ERROR: Cannot execute qrexec-daemon!
```

You can interrupt with Ctrl-C that qrexec waiting process.

3. Access VM console:

```
[user@dom0 ~]$ virsh -c xen:/// console custom-template
```

4. Configure network according to parameters retrieved in first step:

```
ip addr add 10.137.1.53/32 dev eth0
ip route add 10.137.1.1/32 dev eth0
ip route add via 10.137.1.1
echo nameserver 10.137.1.1 > /etc/resolv.conf
```

5. Proceed with normal upgrade instruction described on this page.

## Upgrading to R3.1

Before attempting either an in-place upgrade or a clean installation, we strongly recommend that users *back up their systems* .

Current Qubes R3.0 systems can be upgraded in-place to the latest R3.1 by following the procedure below.

### Upgrade all Template and Standalone VM(s)

By default, in Qubes R3.0, there is only one template. However, users are free to create more templates for special purposes, as well as standalones. More information on using multiple templates, as well as standalones, can be found [here](#). The steps described in this section should be repeated in **all** the user's Template and Standalone VMs.

#### Upgrade Fedora templates:

1. Open a terminal in the template (or standalone). (E.g., use Qubes VM Manager's right-click menu, choose "Run Command in VM," and type `gnome-terminal` there.)
2. Install the `qubes-upgrade-vm` package:

```
sudo yum install qubes-upgrade-vm
```

3. Proceed with a normal upgrade in the template:

```
sudo yum upgrade
```

4. Shut down the template.

#### Upgrade Debian (and Whonix) templates:

1. Open a terminal in the template (or standalone). (E.g., use Qubes VM Manager's right-click menu, choose "Run Command in VM," and type `gnome-terminal` there.)
2. Update repository definition:

```
sudo cp /etc/apt/sources.list.d/qubes-r3.list /etc/apt/sources.list.d/qubes-r3-
↪upgrade.list
sudo sed -i 's/r3.0/r3.1/' /etc/apt/sources.list.d/qubes-r3-upgrade.list
```

3. Proceed with a normal update in the template:

```
sudo apt-get update
sudo apt-get dist-upgrade
```

4. Remove unnecessary now file:

```
sudo rm -f /etc/apt/sources.list.d/qubes-r3-upgrade.list
```

5. Shut down the template.

## Upgrading dom0

**Important:** Do not perform the steps described in this section until **all** your Template and Standalone VMs have been upgraded as described in the previous section. Also, do not shut down `sys-net` or `sys-firewall`, since you will not be able to start them again until after the entire in-place upgrade procedure is complete.

1. Open a terminal in Dom0. (E.g., Start -> System Settings -> Konsole.)
2. Upgrade dom0 to R3.1:

```
sudo qubes-dom0-update --releasever=3.1
```

At this point, most of the `qvm-*` tools will stop working until after you reboot the system.

3. If the previous step completed successfully, your `qubes-core-dom0` version should be 3.1.4 or higher. If it's not, repeat the previous step with the `--clean` option.
4. Reboot dom0. The system may hang during the reboot. If that happens, do not panic. All the filesystems will have already been unmounted at this stage, so you can simply perform a hard reboot (e.g., hold the physical power button down until the machine shuts off, wait a moment, then press it again to start it back up).

Please note that if you use *Anti Evil Maid*, it won't be able to unseal the passphrase the first time the system boots after performing this in-place upgrade procedure since the Xen, kernel, and `initramfs` binaries will have changed. Once the system boots up again, you can reseat your Anti Evil Maid passphrase to the new configuration. Please consult the Anti Evil Maid [documentation](#) for instructions on how to do that.

If you use USB VM, you may encounter problem with starting it on updated Xen version (because of strict default settings). Take a look at [User FAQ](#) for details.

Once you have upgraded dom0, you can install new templates from Qubes R3.1 repositories, in particular the new default Fedora 23 template:

```
sudo qubes-dom0-update qubes-template-fedora-23
```

## Upgrading to R3.2

**Before attempting either an in-place upgrade or a clean installation, we strongly recommend that users [back up their systems](#).**

Current Qubes R3.1 systems can be upgraded in-place to the latest R3.2 by following the procedure below.

## Upgrading dom0

1. Close Qubes Manager (right click on its tray icon -> Exit)
2. Open a terminal in Dom0. (E.g., Start -> System Settings -> Konsole.)
3. Install `qubes-release` package carrying R3.2 repository information.

```
sudo qubes-dom0-update --releasever=3.2 qubes-release
```

If you made any manual changes to repository definitions, new definitions will be installed as `/etc/yum.repos.d/qubes-dom0.repo.rpmnew` (you'll see a message about it during package installation). In such a case, you need to manually apply the changes to `/etc/yum.repos.d/qubes-dom0.repo` or simply replace it with `.rpmnew` file. If you are using Debian-based VM as UpdateVM (`sys-firewall` by default), you need to download few more packages manually, but **do not install them** yet:

```

sudo qubes-dom0-update systemd-compat-libs perl-libwww-perl perl-Term-ANSIColor
↳perl-Term-Cap gdk-pixbuf2-xlib speexdsp qubes-mgmt-salt-admin-tools lvm2
(...)
Transaction Summary
=====
Install      16 Packages (+ 31 Dependent packages)
Upgrade      4 Packages (+200 Dependent packages)

Total download size: 173 M
Is this ok [y/d/N]: n
Exiting on user command
Your transaction was saved, rerun it with:
yum load-transaction /tmp/yum_save_tx.....

```

#### 4. Upgrade dom0 to R3.2:

```
sudo qubes-dom0-update
```

You may wish to disable the screensaver “Lock screen” feature for this step, as during the update XScreenSaver may encounter an “Authentication failed” issue, requiring a hard reboot. Alternatively, you may simply move the mouse regularly.

5. If the previous step completed successfully, your `qubes-core-dom0` version should be 3.2.3 or higher. This can be verified with the command `yum info qubes-core-dom0`. If it's not, repeat the previous step with the `--clean` option.
6. Update configuration files. Some of configuration files were saved with `.rpmnew` extension as the actual files were modified. During upgrade, you'll see information about such cases, like:

```

warning: /etc/salt/minion.d/f_defaults.conf created as /etc/salt/minion.d/f_
↳defaults.conf.rpmnew

```

This will happen for every configuration you have modified manually and for a few that has been modified by Qubes scripts. If you are not sure what to do about them, below is a list of commands to deal with few common cases (either keep the old one, or replace with the new one):

```

rm -f /etc/group.rpmnew
rm -f /etc/shadow.rpmnew
rm -f /etc/qubes/guid.conf.rpmnew
mv -f /etc/nsswitch.conf{.rpmnew,}
mv -f /etc/pam.d/postlogin{.rpmnew,}
mv -f /etc/salt/minion.d/f_defaults.conf{.rpmnew,}
mv -f /etc/dracut.conf{.rpmnew,}

```

#### 7. Reboot dom0.

Please note that if you use *Anti Evil Maid*, it won't be able to unseal the passphrase the first time the system boots after performing this in-place upgrade procedure since the Xen, kernel, and `initramfs` binaries will have changed. Once the system boots up again, you can reseat your Anti Evil Maid passphrase to the new configuration. Please consult the Anti Evil Maid [documentation](#) for instructions on how to do that.

At first login after upgrade you may get a message like this:

```
Your saved session type 'kde-plasma' is not valid any more. Please select a new one,
otherwise 'default' will be used.
```

This is result of upgrade KDE4 (kde-plasma) to KDE5 (plasma). Simply choose your favorite desktop environment and continue.

### Upgrade all Template and Standalone VM(s)

By default, in Qubes R3.1, there are few templates and no standalones. However, users are free to create standalones. More information on using multiple templates, as well as standalones, can be found [here](#). The steps described in this section should be repeated in **all** the user's Template and Standalone VMs.

### Upgrade Fedora templates:

**Note:** This will only upgrade your Fedora template from Qubes 3.1 to Qubes 3.2. This will *not* upgrade your Fedora template from Fedora 23 to Fedora 24. In order to do that, please see the [Fedora 23 template upgrade instructions](#).

1. Open a terminal in the template (or standalone). (E.g., use Qubes VM Manager's right-click menu, choose "Run Command in VM," and type `gnome-terminal` there.)
2. Install the `qubes-upgrade-vm` package:

```
sudo dnf install --refresh qubes-upgrade-vm
```

3. Proceed with a normal upgrade in the template:

```
sudo dnf upgrade --refresh
```

4. Add new packages (only needed in default template):

```
sudo dnf install qubes-mgmt-salt-vm-connector
```

5. Shut down the template.

### Upgrade Debian (and Whonix) templates:

1. Open a terminal in the template (or standalone). (E.g., use Qubes VM Manager's right-click menu, choose "Run Command in VM," and type `gnome-terminal` there.)
2. Update repository definition:

```
sudo cp /etc/apt/sources.list.d/qubes-r3.list /etc/apt/sources.list.d/qubes-r3-  
↪upgrade.list  
sudo sed -i 's/r3.1/r3.2/' /etc/apt/sources.list.d/qubes-r3-upgrade.list
```

3. Proceed with a normal update in the template:

```
sudo apt-get update  
sudo apt-get dist-upgrade
```

4. Add new packages (only needed in default template):

```
sudo apt-get install qubes-mgmt-salt-vm-connector
```

5. Remove unnecessary now file:

```
sudo rm -f /etc/apt/sources.list.d/qubes-r3-upgrade.list
```

6. Shut down the template.

## Upgrading to R4.0

**Before attempting either an in-place upgrade or a clean installation, we strongly recommend that users *back up their systems*.**

Current Qubes R3.2 systems cannot be upgraded in-place to R4.0. A full backup, clean 4.0 install, and restore is required. This can be done by following the procedure below.

### Preparation

1. Go to downloads and prepare a USB drive or DVD with the R4.0 installer.
2. If this is your only computer, and you do not have a R3.2 installer, you should also create a separate R3.2 USB drive or DVD installer at this time.

### Backup R3.2

1. Attach the backup drive you will be using. These steps assume it is a USB drive.
2. Shutdown all non-essential VMs. Typically, `sys-usb` will be the only VM you need to leave running.
3. Follow the **Creating a Backup** section in the *Backup, Restoration, and Migration* guide to back up **all VMs** except `sys-usb`.
4. Verify the integrity of your backup by following the **Restoring from a Backup** section in the *Backup, Restoration, and Migration* guide and:
  - If you're using Qubes Manager, check the box under "Restore options" that says, "Verify backup integrity, do not restore the data."
  - If you're using `qvm-backup-restore` from the command-line, use the `--verify-only` option.
5. If your backup verifies successfully, proceed to the next section. If it does not, **stop**. Go back and repeat the backup steps, review the documentation, and ask for *help*.

### Install R4.0

This section provides general guidance on installing R4.0 as part of migrating from R3.2. For further details, please see the *installation guide*.

1. Shut down R3.2 and boot the R4.0 installer.
2. Follow the installation prompts until you get to the drive selection screen. Choose **I want to make additional space available**. Select the box at the top of the list in order to select all partitions. **This will erase the entire drive**, so do this only if Qubes is the only OS installed on your computer. If you did not successfully verify your backup in the previous section, cancel the installation, and go back to do that now.
3. Complete the R4.0 installation. Ask for *help* if you run into trouble.
4. If you are unable to successfully install R4.0 on your system, all is not lost. Use the R3.2 installer to reinstall R3.2, then restore from your backup.

## Restore from your backup

1. Welcome to Qubes R4.0! The first thing you might notice is that **Qubes Manager** is not started by default. We won't need it for the next step, but we will be starting it later.
2. Since patches may have been released since your installation image was created, update Qubes R4.0 by going to the dom0 command line (**Qubes menu -> Terminal Emulator**) then running:

```
sudo qubes-dom0-update
```

3. Reboot dom0.
4. Go to **Qubes menu -> System Tools -> Qubes Manager** to start it.
5. Follow the **Restoring from a Backup** section in the *Backup, Restoration, and Migration* guide. We recommend that you restore only your *app qubes* and *standalones* from R3.2. Using *templates* and *service qubes* from R3.2 is not fully supported (see #3514). Instead, we recommend using the templates that were created specifically for R4.0, which you can *customize* according to your needs. For the template OS versions supported in R4.0, see *supported releases*. If the restore tool complains about missing templates, you can select the option to restore the app qubes anyway, then change them afterward to use one of the default R4.0 templates.

## Note about additional disp-\* qubes created during restore

One of differences between R3.2 and R4.0 is the handling of disposables. In R3.2, a disposable inherited its network settings (NetVM and firewall rules) from the calling qube. In R4.0, this is no longer the case. Instead, in R4.0 it's possible to create multiple disposable templates and choose which one should be used by each qube. It's even possible to use different disposable templates for different operations from the same qube. This allows much more flexibility, since it allows you to differentiate not only network settings, but all of a qube's properties (including its template, memory settings, etc.).

Restoring a backup from R3.2 preserves the old behavior by creating separate disposable template for each network-providing qube (and also `disp-no-netvm` for network-isolated qubes). Then, each restored qube is configured to use the appropriate disposable template according to its `netvm` or `dispvm_netvm` property from R3.2. This way, disposables started on R4.0 by qubes restored from a R3.2 backup have the same NetVM settings as they had on R3.2.

If you find this behavior undesirable and want to configure it differently, you can remove those `disp-*` disposable templates. But, to do so, you must first make sure they are not set as the value for the `default_dispvm` property on any other qube. Both Qubes Manager and the `qvm-remove` tool will show you where a disposable template is being used, so you can go there and change the setting.

## Upgrade all Template and Standalone VM(s)

We strongly recommend that you update **all** templates and standalones before use so that you have the latest security patches from upstream distributions. In addition, if the default templates have reached EOL (end-of-life) by the time you install R4.0, we strongly recommend that you upgrade them before use. Please see *supported releases* for information on supported OS versions and consult the guides below for specific upgrade instructions:

- *Upgrading Fedora templates*
- *Upgrading Debian templates*
- *Updating Whonix templates*



## How to upgrade to Qubes 4.1

This page explains how to upgrade from Qubes 4.0 to Qubes 4.1. There are two ways to upgrade: a clean installation or an in-place upgrade. In general, a clean installation is simpler and less error-prone, but an in-place upgrade allows you to preserve your customizations.

### Back up

Before attempting either an in-place upgrade or a clean installation, we strongly recommend that you first *back up your system* so that you don't lose any data.

### Clean installation

If you would prefer to perform a clean installation rather than upgrading in-place:

1. Create a *backup* of your current installation.
2. Download the latest 4.1 release.
3. Follow the *installation guide* to install Qubes 4.1.
4. *Restore from your backup* on your new 4.1 installation.

### In-place upgrade

**Warning:** It is not possible to upgrade directly from releases earlier than 4.0. If you're still on an earlier release, please either perform a *clean installation of 4.1* or *upgrade to 4.0* first.

The upgrade may take several hours, and will download several gigabytes of data.

In place upgrade is a complex operation. For this reason, we provide a `qubes-dist-upgrade` tool to handle all the necessary steps automatically. You can install it with the following command in the `dom0` terminal:

```
sudo qubes-dom0-update -y qubes-dist-upgrade
```

The upgrade consists of seven stages — six before restarting the system — labeled “STAGE 0” through “STAGE 5” in the options list below. The seventh stage is rebuilding the application and features lists, which you can start with the `--resync-appmenus-features` option.

Full list of options can be obtained with `qubes-dist-upgrade --help`:

```
Usage: qubes-dist-upgrade [OPTIONS]...
```

This script is used **for** updating current QubesOS R4.0 to R4.1.

Options:

```
--double-metadata-size, -d      (STAGE 0) Double current LVM thin pool metadata.
↪ size.
--update, -t                    (STAGE 1) Update of dom0, TemplatesVM and
↪ StandaloneVM.
--template-standalone-upgrade, -l (STAGE 2) Upgrade templates and standalone VMs to
↪ R4.1 repository.
--release-upgrade, -r          (STAGE 3) Update 'qubes-release' for Qubes R4.1.
--dist-upgrade, -s             (STAGE 4) Upgrade to Qubes R4.1 and Fedora 32
↪ repositories.
```

(continues on next page)

(continued from previous page)

<code>--setup-efi-grub, -g</code>	(STAGE 5) Setup EFI Grub.
<code>--all, -a</code>	Execute all the above stages <b>in</b> one call.
<code>--assumeeyes, -y</code>	Automatically answer yes <b>for</b> all questions.
<code>--usbvm, -u</code>	Current UsbVM defined (default <code>'sys-usb'</code> ).
<code>--netvm, -n</code>	Current NetVM defined (default <code>'sys-net'</code> ).
<code>--updatevm, -f</code>	Current UpdateVM defined (default <code>'sys-firewall'</code> ).
<code>--skip-template-upgrade, -j</code>	Don't upgrade TemplateVM to R4.1 repositories.
<code>--skip-standalone-upgrade, -k</code>	Don't upgrade StandaloneVM to R4.1 repositories.
<code>--only-update</code>	Apply STAGE 0, 2 and resync appmenus only to selected qubes (coma separated list).
<code>--keep-running</code>	List of extra VMs to keep running during update.
<code>↪(coma separated list).</code>	Can be useful <b>if</b> multiple updates proxy VMs are.
<code>↪configured.</code>	
<code>--max-concurrency</code>	How many TemplateVM/StandaloneVM to update <b>in</b> .
<code>↪parallel <b>in</b> STAGE 1</code>	(default 4).
<code>--resync-appmenus-features</code>	Resync applications and features. To be ran.
<code>↪individually</code>	after reboot.

After installing the tool, upgrade can be performed all at once with:

```
sudo qubes-dist-upgrade --all
```

Optionally, an `--assumeeyes` (or `-y`) option can be used to automatically accept all the actions without confirmation.

Alternatively, each upgrade stage can be started separately (see the list of options above).

After completing “STAGE 0” through “STAGE 5”, restart the system. Then perform the final step:

```
sudo qubes-dist-upgrade --resync-appmenus-features
```

When this completes, you can start using Qubes OS 4.1.

## Known issues

1. The script does not convert LUKS1 to LUKS2 disk encryption format (fresh Qubes 4.1 install uses LUKS2 for disk encryption, while earlier versions use LUKS1).
2. Early Qubes 4.0 pre-releases (before R4.0-rc2) made `/boot/efi` partition only 200MB, which is too small for R4.1. In case of such partition layout, clean installation is necessary.
3. If user has created some custom qrexec policy entries, they may not be correctly handled in R4.1, resulting in denying all the calls. It is advised to verify if there are not qrexec policy errors in the log after the system restart - using `journalctl -b` command.

If any early upgrade stage fails, the `qubes-dist-upgrade` tool will try to restore previous system state. After fixing an issue, the tool can be started again, to retry the operation. If a later stage (number 3 or later) fails, the tool may not be able to rollback the changes. But it may still be possible to retry the upgrade.

## Update

After upgrading or performing a clean installation, we strongly recommend *updating your system*.

## How to upgrade to Qubes 4.2

This page explains how to upgrade from Qubes 4.1 to Qubes 4.2. There are two ways to upgrade: a clean installation or an in-place upgrade. In general, a clean installation is simpler and less error-prone, but an in-place upgrade allows you to preserve your customizations.

## Back up

Before attempting either an in-place upgrade or a clean installation, we strongly recommend that you first *back up your system* so that you don't lose any data.

## Clean installation

If you would prefer to perform a clean installation rather than upgrading in-place:

1. Create a *backup* of your current installation.
2. Download the latest 4.2 release.
3. Follow the *installation guide* to install Qubes 4.2.
4. *Restore from your backup* on your new 4.2 installation.

## In-place upgrade

**Warning:** It is not possible to upgrade directly from releases earlier than 4.1. If you're still on an earlier release, please either perform a *clean installation of 4.2* or *upgrade to 4.1* first.

The upgrade may take several hours, and will download several gigabytes of data.

In place upgrade is a complex operation. For this reason, we provide a `qubes-dist-upgrade` tool to handle all the necessary steps automatically. You can install it with the following command in the `dom0` terminal:

```
sudo qubes-dom0-update -y qubes-dist-upgrade
```

The upgrade consists of six stages — three before restarting the system — labeled “STAGE 1” through “STAGE 3” in the options list below, and three after restarting the system — labeled as “STAGE 4” through “STAGE 6” below.

Full list of options can be obtained with `qubes-dist-upgrade --help`:

```
Usage: qubes-dist-upgrade [OPTIONS]...
```

```
This script is used for updating current QubesOS R4.1 to R4.2.
```

```
Options:
```

```
--update, -t                (STAGE 1) Update of dom0, TemplatesVM and
↳ StandaloneVM.
--release-upgrade, -r        (STAGE 2) Update 'qubes-release' for Qubes R4.2.
--dist-upgrade, -s           (STAGE 3) Upgrade to Qubes R4.2 and Fedora 37
↳ repositories.
```

(continues on next page)

(continued from previous page)

<code>--template-standalone-upgrade, -l</code>	(STAGE 4) Upgrade templates and standalone VMs to R4.2 repository.
<code>--finalize, -x</code>	(STAGE 5) Finalize upgrade. It does: <ul style="list-style-type: none"> <li>- resync applications and features</li> <li>- cleanup salt states</li> </ul>
<code>--convert-policy, -p</code>	(STAGE 6) Convert qrexec policy in /etc/qubes-rpc/
<code>policy</code>	to the new format in /etc/qubes/policy.d.
<code>--all-pre-reboot</code>	Execute stages 1 to 3
<code>--all-post-reboot</code>	Execute stages 4 to 6
<code>--assumeeyes, -y</code>	Automatically answer yes for all questions.
<code>--usbvm, -u</code>	Current UsbVM defined (default 'sys-usb').
<code>--netvm, -n</code>	Current NetVM defined (default 'sys-net').
<code>--updatevm, -f</code>	Current UpdateVM defined (default 'sys-firewall').
<code>--skip-template-upgrade, -j</code>	Don't upgrade TemplateVM to R4.2 repositories.
<code>--skip-standalone-upgrade, -k</code>	Don't upgrade StandaloneVM to R4.2 repositories.
<code>--only-update</code>	Apply STAGE 4 and resync appmenus only to selected qubes (comma separated list).
<code>--keep-running</code>	List of extra VMs to keep running during update.
<code>(comma separated list).</code>	Can be useful if multiple updates proxy VMs are
<code>configured.</code>	
<code>--max-concurrency</code>	How many TemplateVM/StandaloneVM to update in
<code>parallel in STAGE 1</code>	(default 4).

After installing the tool, before-reboot stages can be performed at once with:

```
sudo qubes-dist-upgrade --all-pre-reboot
```

Optionally, an `--assumeeyes` (or `-y`) option can be used to automatically accept all the actions without confirmation.

Alternatively, each upgrade stage can be started separately (see the list of options above).

After completing “STAGE 1” through “STAGE 3”, restart the system. Then perform the final steps:

```
sudo qubes-dist-upgrade --all-post-reboot
```

After performing those steps, it's recommended to restart the system one last time.

When this completes, you can start using Qubes OS 4.2.

## Update

After upgrading or performing a clean installation, we strongly recommend *updating your system*.

### 1.12.7 Supported releases

This page details the level and period of support for releases of operating systems in the Qubes ecosystem.

#### Qubes OS

Qubes OS releases are supported for **six months** after each subsequent major or minor release (see [Version Scheme](#)). The current release and past major releases are always available on the Downloads page, while all ISOs, including past minor releases, are available from our [download mirrors](#).

Qubes OS	Start Date	End Date	Status
Release 1	2012-09-03	2015-03-26	Unsupported
Release 2	2014-09-26	2016-04-01	Unsupported
Release 3.0	2015-10-01	2016-09-09	Unsupported
Release 3.1	2016-03-09	2017-03-29	Unsupported
Release 3.2	2016-09-29	2019-03-28	Unsupported
Release 4.0	2018-03-28	2022-08-04	Unsupported
Release 4.1	2022-02-04	2024-06-18	Supported
Release 4.2	2023-12-18	TBA	Supported
Release 4.3	TBA	TBA	In development

#### Note on patch releases

Please note that patch releases, such as 3.2.1 and 4.0.1, do not designate separate, new major or minor releases of Qubes OS. Rather, they designate their respective major or minor releases, such as 3.2 and 4.0, inclusive of all package updates up to a certain point. For example, installing Release 4.0 and fully updating it results in the same system as installing Release 4.0.1. Therefore, patch releases are not displayed as separate rows on any of the tables on this page.

#### Dom0

The table below shows the OS used for dom0 in each Qubes OS release.

Qubes OS	Dom0 OS
Release 1	Fedora 13
Release 2	Fedora 18
Release 3.0	Fedora 20
Release 3.1	Fedora 20
Release 3.2	Fedora 23
Release 4.0	Fedora 25
Release 4.1	Fedora 32
Release 4.2	Fedora 37

## Note on dom0 and EOL

Dom0 is isolated from domUs. DomUs can access only a few interfaces, such as Xen, device backends (in the dom0 kernel and in other VMs, such as the NetVM), and Qubes tools (gui-daemon, qrexec-daemon, etc.). These components are *security-critical*, and we provide updates for all of them (when necessary), regardless of the support status of the base distribution. For this reason, we consider it safe to continue using a given base distribution in dom0 even after it has reached end-of-life (EOL).

## Templates

The following table shows select *template* (and *standalone*) releases that are currently supported. Currently, only *Fedora* and *Debian* templates are officially supported by the Qubes OS Project. *Whonix* templates are supported by our partner, the *Whonix Project*. Qubes support for each template ends when that upstream release reaches end-of-life (EOL), even if that release is included in the table below. Please see below for distribution-specific notes.

It is the responsibility of each distribution to clearly notify its users in advance of its own EOL dates, and it is users' responsibility to heed these notices by upgrading to supported releases. As a courtesy to Qubes users, we attempt to pass along upstream EOL notices we receive for select distributions, but our ability to do this reliably is dependent on the upstream distribution's practices. For example, if a distribution provides a mailing list similar to *qubes-announce*, which allows us to receive only very important, infrequent messages, including EOL announcements, we are much more likely to be able to pass along EOL notices to Qubes users reliably. Qubes users can always check the EOL status of an upstream release on the upstream distribution's website (see *Fedora EOL* and *Debian Releases*).

Qubes OS	Fedora	Debian
Release 4.1	38	11, 12
Release 4.2	38	12

## Note on Debian support

Debian releases have two EOL dates: regular and *long-term support (LTS)*. See *Debian Production Releases* for a chart that illustrates this. Qubes support ends at the *regular* EOL date, *not* the LTS EOL date, unless a specific exception has been made.

## Note on Whonix support

*Whonix* templates are supported by our partner, the *Whonix Project*. The Whonix Project has set its own support policy for Whonix templates in Qubes. Please see the *Qubes-Whonix version support policy* for details.

## 1.12.8 Testing new releases and updates

Testing new Qubes OS releases and updates is one of the most helpful ways in which you can *contribute* to the Qubes OS Project. If you're interested in helping with this, please *join the testing team*. There are several different types of testing, which we'll cover below.

**Warning:** Software testing is intended for advanced users and developers. You should only attempt to do this if you know what you're doing. Never rely on code that is in testing for critical work!

## Releases

How to test upcoming Qubes OS releases:

- Test the latest release candidate (RC) on the downloads page, if one is currently available. (Or try an older RC from our [FTP server](#).)
- Try the [signed weekly builds](#). ([Learn more](#) and [track their status](#).)
- Use *qubes-builder* to build the latest release yourself.
- (No support) Experiment with developer alpha ISOs found from time to time at [Qubes OpenQA](#).

Please make sure to [report any bugs you encounter](#).

See [Version scheme](#) for details about release versions and schedules. See [Release checklist](#) for details about the RC process.

## Updates

How to test updates:

- Enable *dom0 testing repositories*.
- Enable *template testing repositories*.

Every new update is first uploaded to the `security-testing` repository if it is a security update or `current-testing` if it is a normal update. The update remains in `security-testing` or `current-testing` for a minimum of one week. On occasion, an exception is made for a particularly critical security update, which is immediately pushed to the `current` stable repository. In general, however, security updates remain in `security-testing` for two weeks before migrating to `current`. Normal updates generally remain in `current-testing` until they have been sufficiently tested by the community, which can last weeks or even months, depending on the amount of feedback received (see [Providing feedback](#)).

“Sufficient testing” is, in practice, a fluid term that is up the developers’ judgment. In general, it means either that no negative feedback and at least one piece of positive feedback has been received or that the package has been in `current-testing` for long enough, depending on the component and the complexity of the changes.

A limitation of the current testing setup is that it is only possible to migrate the *most recent version* of a package from `current-testing` to `current`. This means that, if a newer version of a package is uploaded to `current-testing`, it will no longer be possible to migrate any older versions of that same package from `current-testing` to `current`, even if one of those older versions has been deemed stable enough. While this limitation can be inconvenient, the benefits outweigh the costs, since it greatly simplifies the testing and reporting process.

## Templates

How to test *templates*:

- For official templates, enable the `qubes-templates-itl-testing` repository, then *install* the desired template.
- For community templates, enable the `qubes-templates-community-testing` repository, then *install* the desired template.

To temporarily enable any of these repos, use the `--enablerepo=<repo-name>` option. Example commands:

```
qvm-template --enablerepo=qubes-templates-itl-testing list --available
qvm-template --enablerepo=qubes-templates-itl-testing install <template_name>
```

To enable any of these repos permanently, change the corresponding enabled value to 1 in `/etc/qubes/repo-templates`. To disable any of these repos permanently, change the corresponding enabled value to 0.

## Providing feedback

Since the whole point of testing software is to discover and fix bugs, your feedback is an essential part of this process. We use an [automated build process](#). For every package that is uploaded to a testing repository, a GitHub issue is created in the [updates-status](#) repository for tracking purposes. We welcome any kind of feedback on any package in any testing repository. Even a simple “thumbs up” or “thumbs down” reaction on the package’s associated issue would help us to decide whether the package is ready to be migrated to a stable repository. If you [report a bug](#) in a package that is in a testing repository, please reference the appropriate issue in [updates-status](#).

### 1.12.9 How to organize your qubes

When people first learn about Qubes OS, their initial reaction is often, “Wow, this looks really cool! But... what can I actually *do* with it?” It’s not always obvious which qubes you should create, what you should do in each one, and whether your organizational ideas makes sense from a security or usage perspective.

Each qube is essentially a secure compartment, and you can create as many of them as you like and connect them to each other in various ways. They’re sort of like Lego blocks in the sense that you can build whatever you want. But if you’re not sure what to build, then this open-ended freedom can be daunting. It’s a bit like staring at a blank document when you first sit down to write something. The possibilities are endless, and you may not know where to begin!

The truth is that no one else can tell you *exactly* how you should organize your qubes, as there is no single correct answer to that question. It depends on your needs, desires, and preferences. Every user’s optimal setup will be different. However, what we *can* do is provide you with some illustrative examples based on questionnaires and interviews with Qubes users and developers, as well as our own personal experience and insight from using Qubes over the years. You may be able to adapt some of these examples to fit your own unique situation. More importantly, walking you through the rationale behind various decisions will teach you how to apply the same thought process to your own organizational decisions. Let’s begin!

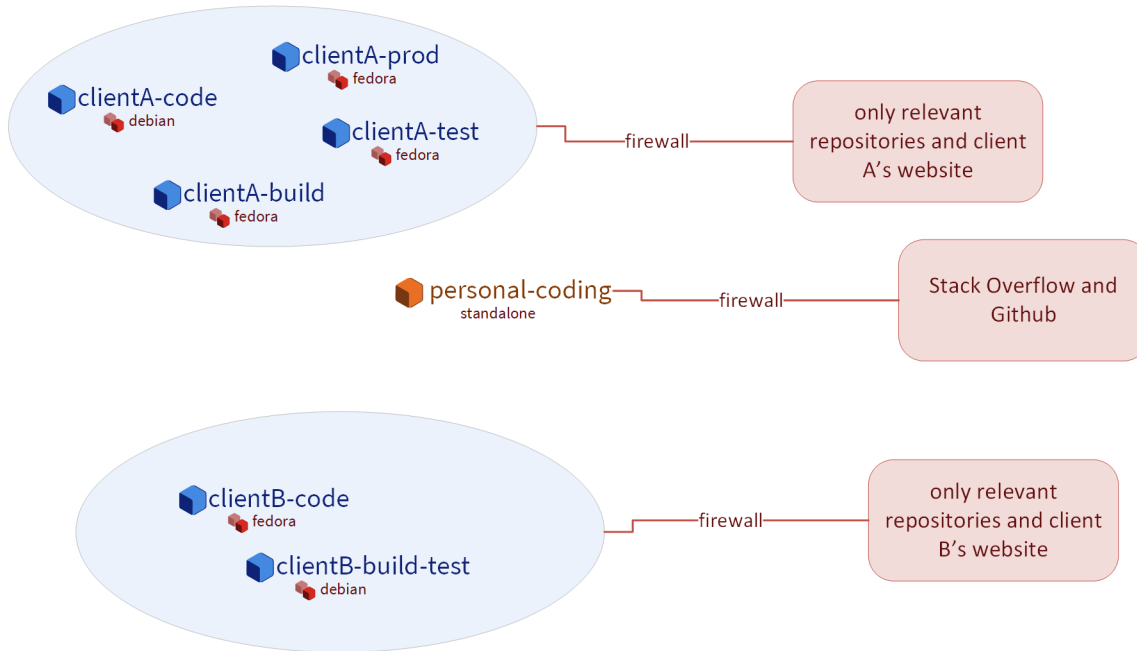
#### Alice, the software developer

Alice is a freelance dev who works on several projects for different clients simultaneously. The projects have varying requirements and often different build environments. She has a separate set of qubes for each project. She keeps them organized by coming up with a naming scheme, such as:

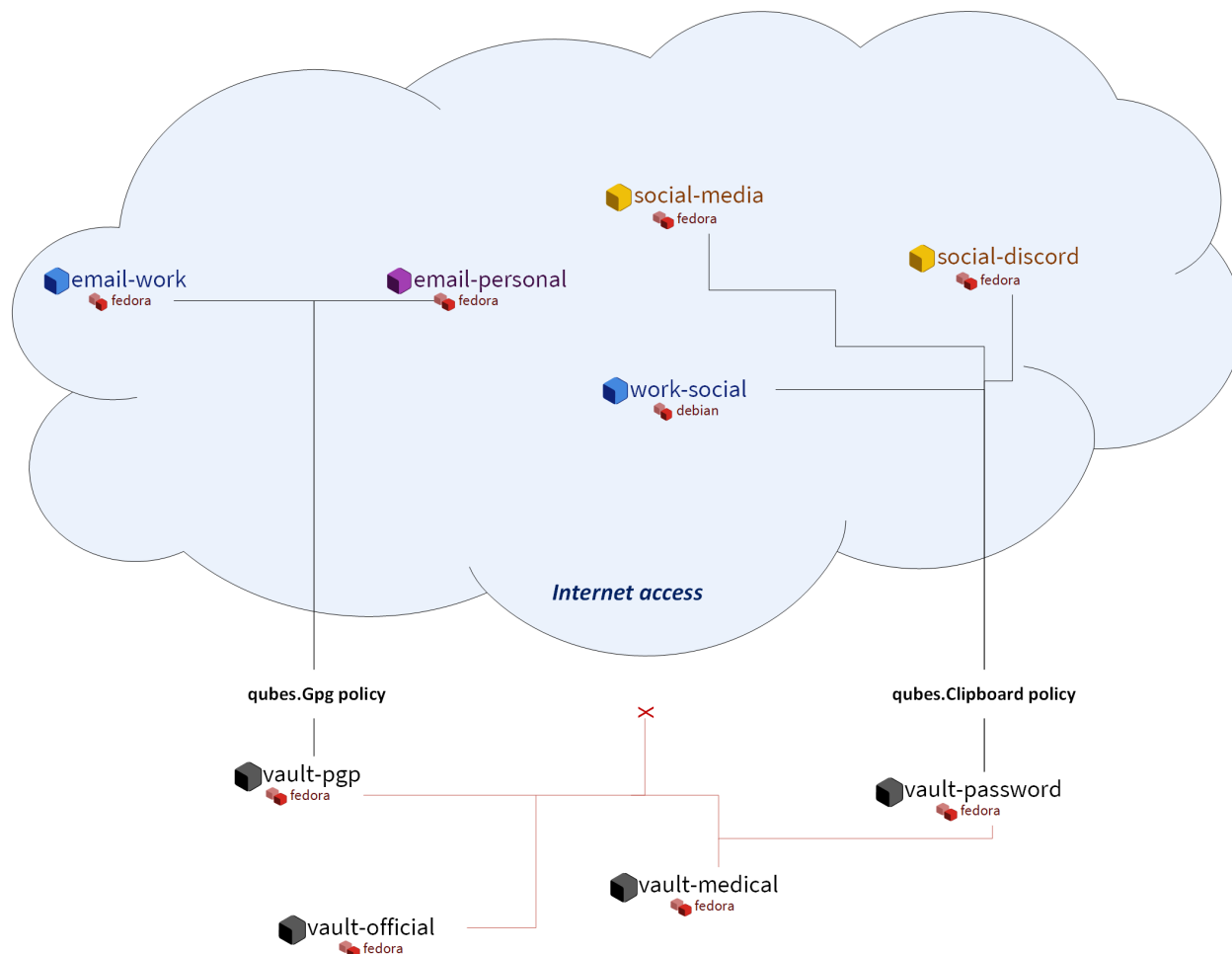
```
clientA-code
clientA-build
clientA-test
clientA-prod
projectB-code
projectB-build-test
projectB-prod
...
```

This helps her keep groups of qubes organized in a set. Some of her qubes are based on [Debian templates](#), while others are based on [Fedora templates](#). The reason for this is that some software packages are more readily available in one distribution as opposed to the other. Alice’s setup looks like this:





- Several qubes for writing code.** Here's where she runs her IDE, commits code, and signs her commits. These qubes are based on different templates depending on which tools and which development environment she needs. In general, Alice likes to have a separate qube of this type for each client or each project. This allows her to keep everything organized and avoid accidentally mixing up any access credentials or client code, which could be disastrous. This also allows her to truthfully tell her clients that their code is always securely isolated from all her other clients. She likes to use the *Qubes firewall* to restrict these qubes' network access to only the code repositories she needs in that qube in order to avoid accidentally interacting with anything else on her local network or on the internet. Alice also has some qubes of this type for personal programming projects that she works on just for fun when she has "free time" (whatever that is).
- Several qubes for building and testing.** Again, Alice usually likes to have one of these for each client or project in order to keep things organized. However, this can become rather cumbersome and memory-intensive when many such qubes are running at the same time, so Alice will sometimes use the same qube for building and testing, or for multiple projects that require the same environment, when she decides that the marginal benefits of extra compartmentalization aren't worth the trouble. Here's where she pulls any dependencies she needs, compiles her code, runs her build toolchain, and tests her deliverables. In some cases, she finds it useful to use *standalones* for these so that it's easier to quickly *install different pieces of software* without having to juggle rebooting both the template and an app qube. She also sometimes finds it necessary (or just convenient) to make edits to config files in the root filesystem, and she'd rather not have to worry about losing those changes during an app qube reboot. She knows that she could use *bind-dirs* to make those changes persistent, but sometimes she doesn't want to get bogged down doing with all that and figures it wouldn't be worth it just for this one qube. She's secretly glad that Qubes OS doesn't judge her this and just gives her the freedom to do things however she likes while keeping everything securely compartmentalized. At times like these, she takes comfort in knowing that things can be messy and disorganized *within* a qube while her overall digital life remains well-organized.



- **Several email qubes.** Since Alice is a command-line aficionado, she likes to use a terminal-based email client, so both her work and personal email qubes are based on a template with *Mutt* installed. The email qubes where she sends and receives PGP-signed and encrypted email securely accesses the private keys in her PGP backend qube (more on that below). To guard against malicious attachments, she configured Mutt to open all attachment files in *disposable qubes*.
- **Several qubes for communication tools**, like Signal, Slack, Zoom, Telegram, IRC, and Discord. This is where she teleconferences and chats with clients. She uses *USB passthrough* to attach her webcam to each qube as needed and detaches it afterward. Likewise, she gives each qube access to her microphone while it's needed, then removes access afterward. This way, she doesn't have to trust any given video chat program's mute button and doesn't have to worry about being spied on when she's not on a call. She also has a qube for social media platforms like Twitter, Reddit, and Hacker News for networking and keeping up with new developments (or so she claims; in reality, it's mostly for feuds over programming language superiority, Vim vs. Emacs wars, and tabs vs. spaces crusades).
- **A GPG backend vault.** Vaults are completely offline qubes that are isolated from the network. This particular vault holds Alice's private keys (e.g., for code signing and email) and is securely accessed by several other "frontend" qubes via the *Split GPG* system. Split GPG allows only the frontend qubes that Alice explicitly authorizes to have the ability to request PGP operations (e.g., signing and encryption) in the backend vault. Even then, no qube ever has direct access to Alice's private keys except the backend vault itself.
- **A password manager vault.** This is another completely offline, network-isolated qube where Alice uses her offline password manager, KeePassXC, to store all of her usernames and passwords. She uses the *secure copy and paste* system to quickly copy credentials into other qubes whenever she needs to log into anything.

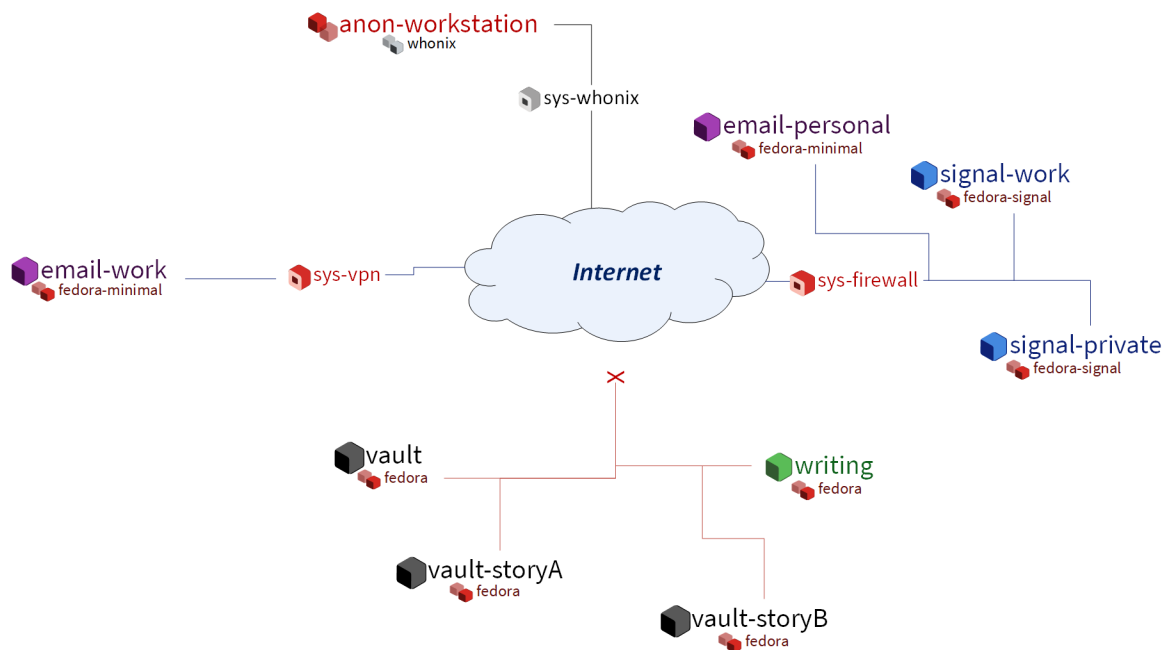
- **Personal qubes.** One of the things Alice loves the most about Qubes is that she can use it for both work *and* personal stuff without having to worry about cross-contamination. Accordingly, she has several qubes that pertain to her personal life. For example, she has an offline vault that holds her medical documents, test results, and vaccination records. She has another offline vault for her government documents, birth certificate, scans of her passport, and so on. She also has some personal social media accounts in a separate qube for keeping up with family members and friends from school.

When she finishes her work for a given client, Alice sends off her deliverables, *backs up* the qubes containing the work for that client, and deletes them from her system. If she ever needs those qubes again or just wants to reference them, she can easily restore them from her backup, and the internal state of each one will be exactly as it was when she finished that project.

### Bob, the investigative journalist

As part of his research and reporting, Bob is frequently forced to interact with suspicious files, often from anonymous sources. For example, he may receive an email with an attachment that claims to be a tip about a story he's working on. Of course, he knows that it could just as easily be malware intended to infect his computer. Qubes OS is essential for Bob, since it allows him to handle all this suspicious data securely, keeping it compartmentalized so that it doesn't risk infecting the rest of his machine.

Bob isn't a super technical guy. He prefers to keep his tools simple so he can focus on what's important to him: uncovering the truth, exposing the guilty, exonerating the innocent, and shining light on the dark corners of society. His mind doesn't naturally gravitate to the technical details of how his computer works, but he's aware that people are getting hacked all the time and that the nature of his work might make him a target. He wants to protect his sources, his colleagues, his family, and himself; and he understands that computer security is an important part of that. He has a Qubes laptop that he uses only for work, which contains:



- **One offline qube for writing.** It runs only LibreOffice Writer. This is where Bob does all of his writing. This window is usually open side-by-side with another window containing research or material from a source.
- **Multiple email qubes.** One is for receiving emails from the general public. Another is for emailing his editor and colleagues. Both are based on a *minimal template* with Thunderbird installed. He's configured both to open all attachments in *disposables* that are offline in case an attachment contains a beacon that tries to phone home.

- **Whonix qubes.** He has the standard `sys-whonix` service qube for providing Torified network access, and he uses disposable `anon-workstation` app qubes for using Tor Browser to do research on stories he's writing. Since the topic is often of a sensitive nature and might implicate powerful individuals, it's important that he be able to conduct this research with a degree of anonymity. He doesn't want the subjects of his investigation to know that he's looking into them. He also doesn't want his network requests being traced back to his work or home IP addresses. Whonix helps with both of these concerns. He also has another Whonix-based disposable template for receiving tips anonymously via Tor, since some high-risk whistleblowers he's interacted with have said that they can't take a chance with any other form of communication.
- **Two qubes for Signal .** Bob has two Signal app qubes (both on the same template in which the Signal desktop app is installed). One is linked to his own mobile number for communicating with co-workers and other known, trusted contacts. The other is a public number that serves as an additional way for sources to reach him confidentially. This is especially useful for individuals who don't use Tor but for whom unencrypted communication could be dangerous.
- **Several data vaults.** When someone sends Bob material that turns out to be useful, or when he comes across useful material while doing his own research, he stores a copy in a completely offline, network-isolated vault qube. Most of these files are PDFs and images, though some are audio files, videos, and text files. Since most of them are from unknown or untrusted sources, Bob isn't sure if it would be safe to put them all in the same vault, so he makes different vaults (usually one for each story or topic) just in case. This has the side benefit of helping to keep things organized.
- **A VPN qube and associated qubes for accessing work resources.** The servers at work can only be accessed from the organization's network, so Bob has certain qubes that are connected to a VPN qube so that he can upload his work and access anything he needs on the local network when he's not physically there.
- **A password manager vault.** Bob stores all of his login credentials in the default password manager that came with his offline vault qube. He *securely copies and pastes* them into other qubes as needed.

A colleague helped Bob set up his Qubes system initially and showed him how to use it. Since Bob's workflow is pretty consistent and straightforward, the way his qubes are organized doesn't change much, and this is just fine by him. His colleague told him to remember a few simple rules: Don't copy or move *text* or *files* from less trusted to more trusted qubes; *update* your system when prompted; and make regular *backups*. Bob doesn't have the need to try out new software or tweak any settings, so he can do everything he needs to do on a daily basis without having to interact with the command line.

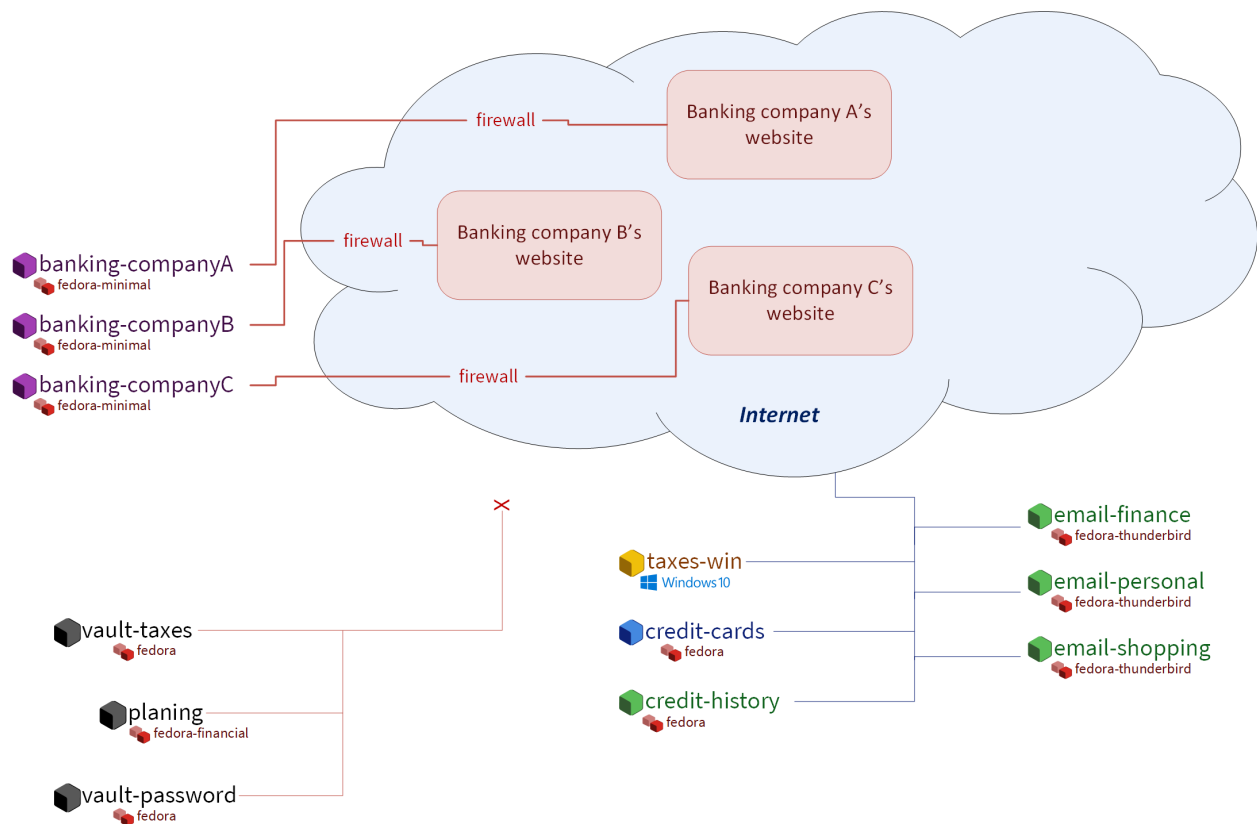
## Carol, the investor

Carol works hard and lives below her means so that she can save money and invest it for her future. She hopes to become financially independent and maybe even retire early someday, and she's decided that her best bet for achieving this is by investing for the long term and allow compounding to do its work. However, after doing some research into her country's consumer financial protection laws, she learned that there's no legal guarantee that customers will be made whole in the event of theft or fraud. The various insurance and protection organizations only guarantee recovery in the case of a financial institution *failing*, which is quite different from an individual customer being hacked. Moreover, even though many financial institutions have their own cybercrime policies, rarely, if ever, do they explicitly guarantee reimbursement in the event that a *customer* gets hacked (rather than the institution itself).

**Warning:** Carol looked into how thieves might actually try to steal her hard-earned wealth and was surprised to learn that they have all sorts of ploys that she had never even considered. For example, she had assumed that any theft would, at the bare minimum, have to involve transferring money out of her account. That seems like a safe assumption. But then she read about “pump and dump” attacks, where thieves buy up some penny stock, hack into innocent people's brokerage accounts, then use the victims' funds to buy that same penny stock, “pumping” up its price so that the thieves can “dump” their shares on the market, leaving the victims with worthless shares. No money is ever transferred into or out of the victims' account; it's just used to buy and sell securities. So, all the

safeguards preventing new bank accounts from being added or requiring extra approval for outbound transfers do nothing to protect victims' funds in cases like these. And this is just one example! Carol realized that she couldn't assume that existing safeguards against specific, known attacks were enough. She had to think about security at a more fundamental level and design it into her digital life from the ground up.

After learning about all this, Carol decided that it was ultimately up to her to take care of her own cybersecurity. She couldn't rely on anyone else to do it for her. Sure, most people just use regular consumer tech and will probably end up fine, but, she reminded herself, most people also don't have as much to lose. It's not a risk that she was willing to take with her future, especially knowing that there's probably no government bailout waiting for her and that all the brokerage firms' vaguely reassuring marketing language about cybersecurity isn't legally binding. So, Carol started reading more about computer security and eventually stumbled upon Qubes OS after searching the web for "most secure operating system." She read about how it's designed and why. Although she didn't immediately understand all of the technical details, the fundamental principle of *security-by-compartmentalization* made intuitive sense to her, and the more she learned about the technical aspects, the more she realized that this is what she'd been looking for. Today, her setup looks like this:



- **One qube for each investment firm and bank.** Carol has a few different retirement accounts, brokerage accounts, and bank accounts. She treats each qube like a "secure terminal" for accessing only that one institution's website. She makes her transactions and saves any statements and confirmations she downloads in that qube. She uses the *Qubes firewall* to enable access only to that institution's website in that qube so that she doesn't accidentally visit any others. Since most of what she does involves using websites and PDFs, most of Carol's app qubes are based on a *minimal template* with just a web browser (which doubles as a PDF viewer) and a file manager installed.
- **One qube for all her credit card accounts.** Carol started to make a separate qube for each credit card account but ultimately decided against it. For one thing, the consumer protections for credit card fraud in her country are much better than for losing assets to theft or fraud in a bank or brokerage account, so the security risk isn't as high.

Second, there's actually not a whole lot that an attacker could do with access to her credit cards' online accounts or her old credit card statements, since online access to these generally doesn't allow spending or withdrawing any money. So, even the worst case scenario here wouldn't be catastrophic, unlike with her bank and brokerage accounts. Third, she's not too worried about any of her credit card company websites being used to attack each other or her qube. (As long as it's contained to a single qube, she's fine with that level of risk.) Last, but not least: She has way too many credit cards! While Carol is very frugal, she likes to collect the sign-up bonuses that are offered for opening new cards, so she's accumulated quite a few of them. (However, she's always careful to pay off her balance each month, so she never pays interest. She's also pretty disciplined about only spending what she would have spent *anyway* and not being tempted to spend more just to meet a spending requirement or because she can.) At any rate, Carol has decided that the tiny benefit she stands to gain from having a separate qube for every credit card website wouldn't be worth the hassle of having to manage so many extra qubes.

- **A qube for credit monitoring, credit reports, and credit history services.** Carol has worked hard to build up a good credit score, and she's concerned about identity theft, so she has one qube dedicated to managing her free credit monitoring services and downloading her free annual credit reports.
- **Two qubes for taxes.** Carol has a [Windows qube](#) for running her Windows-only tax software. She also has an offline vault where she stores all of her tax-related forms and documents, organized by year.
- **A qube for financial planning and tracking.** Carol loves spreadsheets, so this offline qube is where she maintains a master spreadsheet to track all of her investments and her savings rate. She also keeps her budgeting spreadsheet, insurance spreadsheet, and written investment policy statement here. This qube is based on a template with some additional productivity software, like LibreOffice and Gnumeric (so that Carol can run her own Monte Carlo simulations).
- **Various email qubes.** Carol likes to have one email qube for her most important financial accounts; a separate one for her credit cards accounts, online shopping accounts, and insurance companies; and another one for personal email. They're all based on the same template with Thunderbird installed.
- **A password manager vault.** A network-isolated qube where Carol stores all of her account usernames and passwords in KeePassXC. She uses the [Qubes global clipboard](#) to copy and paste them into her other qubes when she needs to log into her accounts.

### Bonus: Carol explores new financial technology

The vast majority of Carol's assets are in broad-based, low-cost, passively-managed indexed funds. Lately, however, she's started getting interested in cryptocurrency. She's still committed to staying the course with her tried-and-true investments, and she's always been skeptical of new asset classes, especially those that don't generate cash flows or that often seem to be associated with scams or wild speculation. However, she finds the ability to self-custody a portion of her assets appealing from a long-term risk management perspective, particularly as a hedge against certain types of political risk.

**Danger:** Some of Carol's friends warned her that cryptocurrency is extremely volatile and that hacking and theft are common occurrences. Carol agreed and reassured them that she's educated herself about the risks and will make sure she never invests more than she can afford to lose.

Carol has added the following to her Qubes setup:

- **A standalone qube for running Bitcoin Core and an offline wallet vault.** Carol finds the design and security properties of Bitcoin very interesting, so she's experimenting with running a full node. She also created a network-isolated vault in order to try running a copy of Bitcoin Core completely offline as a "cold storage" wallet. She's still trying to figure out how this compares to an actual hardware wallet, paper wallet, or physically air-gapped machine, but she's figures they all have different security properties. She also recently heard about using [Electrum as a "split" wallet in Qubes](#) and is interested in exploring that further.



- **Whonix qubes.** Carol read somewhere that Bitcoin nodes should be run over Tor for privacy and security. She found it very convenient that Whonix is already integrated into Qubes, so she simply set her Bitcoin Core “full node” qube to use `sys-whonix` as its networking qube.
- **Various qubes for DeFi and web3.** Carol has also started getting into DeFi (decentralized finance) and web3 on Ethereum and other smart contract blockchains, so a friend recommended that she get a Ledger hardware wallet. She downloaded the Ledger Live software in an app qube and [set up her system to recognize the Ledger](#). She can now start her [USB qube](#), plug her Ledger into it into a USB port, [use the Qubes Devices widget to attach it](#) to her Ledger Live qube, and from there she can interact with the software. She has a separate qube with the Metamask extension installed in a web browser. She can also use the Qubes Devices widget to attach her Ledger to this qube so she can use Metamask in conjunction with her Ledger to interact with smart contracts and decentralized exchanges.
- **Various qubes for research and centralized exchanges.** Carol uses these when she wants to check block explorer websites, coin listing and market cap sites, aggregation tools, or just to see what the latest buzz is on Crypto Twitter.

Carol makes sure to back up all of her qubes that contain important account statements, confirmations, spreadsheets, cryptocurrency wallets, and her password manager vault. If she has extra storage space, she’ll also back up her templates and even her Bitcoin full node qube, but she’ll skip them if she doesn’t have time or space, since she knows she can always recreate them again later and download what she needs from the Internet.

## Conclusion

The characters we’ve met today may be fictional, but they represent the needs of real users like you. You may find that your own needs overlap with more than one of them, in which case you may find it useful to model certain subsets of your overall Qubes system on different examples. You probably also noticed that there are commonalities among them. Most people need to use email, for example, so most people will need at least one email qube and a suitable template to base it on. But not everyone will need [Split GPG](#), and not everyone will want to use the same email client. On the other hand, almost everyone will need a password manager, and it pretty much always makes sense to keep it in an offline, network-isolated vault.

---

**Note:** As you gain experience with Qubes, you may find yourself disagreeing with some of the decisions our fictional friends made. That’s okay! There are many different ways to organize a Qubes system, and the most important criterion is that it serves the needs of its owner. Since everyone’s needs are different, it’s perfectly normal to find yourself doing things a bit differently. Nonetheless, there are some general principles that almost all users find helpful, especially when they’re first starting out.

---

As you’re designing your own Qubes system, keep in mind some of the following lessons from our case studies:

- **You’ll probably change your mind as you go.** You’ll realize that one qube should really be split into two, or you’ll realize that it doesn’t really make sense for two qubes to be separate and that they should instead be merged into one. That’s okay. Qubes OS supports your ability to adapt and make changes as you go. Try to maintain a flexible mindset. Things will eventually settle down, and you’ll find your groove. Changes to the way you organize your qubes will become less drastic and less frequent over time.
- **Make frequent backups.** Losing data is never fun, whether it’s from an accidental deletion, a system crash, buggy software, or a hardware failure. By getting into the habit of making frequent backups now, you’ll save yourself from a lot of pain in the future. Many people never take backups seriously until they suffer catastrophic data loss. That’s human nature. If you’ve experienced that before, then you know the pain. Resolve now never to let it happen again. If you’ve never experienced it, count yourself lucky and try to learn from the hard-won experience of others. Keeping good backups also allows you to be a bit more free with reorganizations. You can delete qubes that you think you won’t need anymore without having to worry that you might need them again someday, since you know you can always restore them from a backup.

- **Think about which programs you want to run and where you want to store data.** In some cases, it makes sense to run programs and store data in the same qube, for example, if the data is generated by that program. In other cases, it makes sense to have qubes that are exclusively for storing data (e.g., offline data storage vaults) and other qubes that are exclusively for running programs (e.g., web browser-only qubes). Remember that when you make backups, it's only essential to back up data that can't be replaced. This can allow you to achieve minimal backups that are quite small compared to the total size of your installation. Templates, service qubes, and qubes that are used exclusively for running programs and that contain no data don't necessarily have to be backed up as long as you're confident that you can recreate them if needed. This is why it's a good practice to keep notes on which packages you installed in which templates and which customizations and configurations you made. Then you can refer to your notes the next time you need to recreate those qubes. Of course, backing up everything is not a bad idea either. It may require a bit more time and disk space upfront, but for some people, it can be just as important as backing up their irreplaceable data. If your system is mission-critical, and you can't afford more than a certain amount of downtime, then by all means, back everything up!
- **Introspect on your own behavior.** For example, if you find yourself wanting to find some way to get two qubes to share the same storage space, then this is probably a sign that those two qubes shouldn't be separate in the first place. Sharing storage with each other largely breaks down the secure wall between them, making the separation somewhat pointless. But you probably had a good reason for wanting to make them two separate qubes instead of one to begin with. What exactly was that reason? If it has to do with security, then why are you okay with them freely sharing data that could allow one to infect the other? If you're sure sharing the data wouldn't cause one to infect the other, then what's the security rationale for keeping them separate? By critically examining your own thought process in this way, you can uncover inconsistencies and contradictions that allow you to better refine your system, resulting in a more logical organization that serves your needs better and better over time.
- **Don't assume that just because you can't find a way to attack your system, an adversary wouldn't be able to.** When you're thinking about whether it's a good idea to combine different activities or data in a single qube, for example, you might think, "Well, I can't really see how these pose a risk to each other." The problem is that we often miss attack vectors that sophisticated adversaries spot and can use against us. After all, most people don't think that using a conventional monolithic operating system is risky, when in reality their entire digital life can be taken down in one fell swoop. That's why a good rule of thumb is: When in doubt, compartmentalize.
- **But remember that compartmentalization — like everything else — can be taken to an extreme.** The appropriate amount depends on your temperament, time, patience, experience, risk tolerance, and expertise. In short, there can be such a thing as *too much* compartmentalization! You also have to be able to actually *use* your computer efficiently to do the things you need to do. For example, if you immediately try to jump into doing everything in *disposables* and find yourself constantly losing work (e.g., because you forget to transfer it out before the disposable self-destructs), then that's a big problem! Your extra self-imposed security measures are interfering with the very thing they're designed to protect. At times like these, take a deep breath and remember that you've already reaped the vast majority of the security benefit simply by using Qubes OS in the first place and performing basic compartmentalization (e.g., no random web browsing in templates). Each further step of hardening and compartmentalization beyond that represents an incremental gain with diminishing marginal utility. Try not to allow the perfect to be the enemy of the good!

### 1.12.10 How to update

*This page is about updating your system while staying on the same [supported version of Qubes OS](#) . If you're instead looking to upgrade from your current version of Qubes OS to a newer version, see [Upgrade guides](#) .*

It is important to keep your Qubes OS system up-to-date to ensure you have the latest security updates, as well as the latest non-security enhancements and bug fixes.

Fully updating your Qubes OS system means updating:

- *dom0*
- *templates*



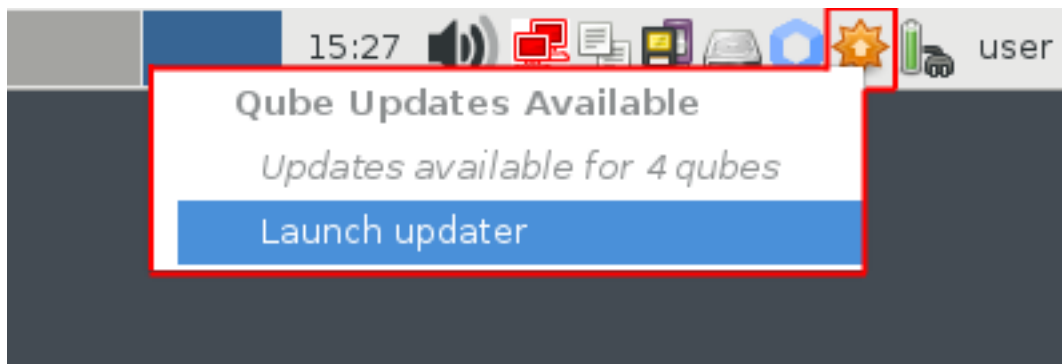
- *standalones* (if you have any)

## Security updates

Security updates are an extremely important part of keeping your Qubes installation secure. When there is an important security incident, we will issue a [Qubes Security Bulletin \(QSB\)](#) via the [Qubes Security Pack \(qubes-secpack\)](#). It is very important to read each new QSB and follow any user instructions it contains. Most of the time, simply updating your system normally, as described below, will be sufficient to obtain security updates. However, in some cases, special action may be required on your part, which will be explained in the QSB.

## Checking for updates

By default, the **Qubes Update** tool will appear as an icon in the Notification Area when updates are available.



However, you can also start the tool manually by selecting it in the Applications Menu under “Qubes Tools.” Even if no updates have been detected, you can use this tool to check for updates manually at any time by selecting “Enable updates for qubes without known available updates,” then selecting all desired items from the list and clicking “Next.”

---

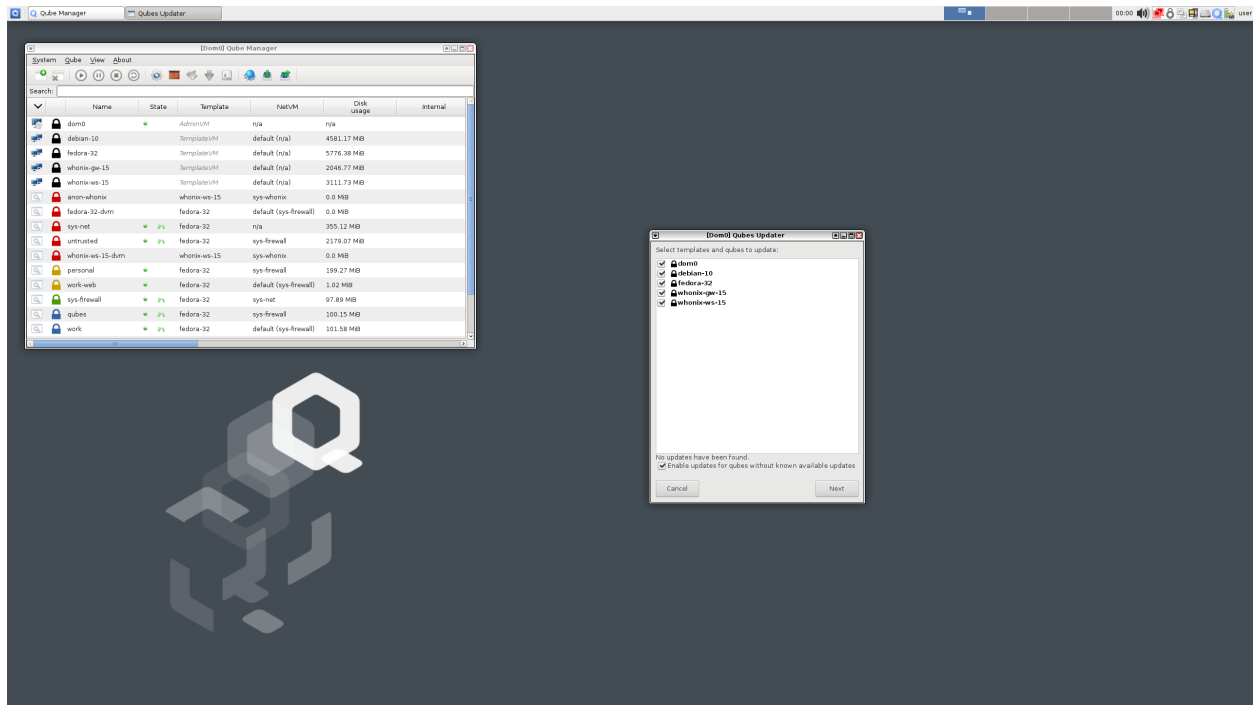
**Note:** For information about how templates download updates, please see [Why don't templates have network access?](#) and the Updates proxy.

---

By default, most qubes that are connected to the internet will periodically check for updates for their parent templates. If updates are available, you will receive a notification as described above. However, if you have any templates that do *not* have any online child qubes, you will *not* receive update notifications for them. Therefore, you should regularly update such templates manually instead.

## Installing updates

The standard way to install updates is with the **Qubes Update** tool. (However, you can also perform the same action via the *command-line interface*.)



Simply follow the on-screen instructions, and the tool will download and install all available updates for you. Note that if you are downloading updates over Tor (`sys-whonix`), this can take a very long time, especially if there are a lot of updates available.

## Restarting after updating

Certain updates require certain components to be restarted in order for the updates to take effect:

- QSBs may instruct you to restart certain components after installing updates.
- Dom0 should be restarted after all **Xen** and **kernel** updates.
- On Intel systems, dom0 should be restarted after all `microcode_ctl` updates.
- On AMD systems, dom0 should be restarted after all `linux-firmware` updates.
- After updating a template, first shut down the template, then restart all running qubes based on that template.

## AEM resealing after updating

If you use *Anti Evil Maid (AEM)*, you'll have to "reseal" after certain updates. It's common for QSBs to contain instructions to this effect. See the relevant QSB and the [AEM README](#) for details.

## Command-line interface

**Danger:** Warning: Updating with direct commands such as `qubes-dom0-update`, `dnf update`, and `apt update` is not recommended, since these bypass built-in Qubes OS update security measures. Instead, we strongly recommend using the Qubes Update tool or its command-line equivalents, as described below. (By contrast, installing packages using direct package manager commands is fine.)

Advanced users may wish to perform updates via the command-line interface. The recommended way to do this is by applying the following two Salt states. **Applying these two Salt states is the same as updating via the Qubes Update tool.**

- `update.qubes-dom0`
- `update.qubes-vm`

In your update qube, a terminal window opens that displays the progress of operations and output as it is logged. At the end of the process, logs are sent back to dom0. You answer any yes/no prompts in your dom0 terminal window.

Advanced users may also be interested in learning [how to enable the testing repos](#).

## Upgrading to avoid EOL

The above covers updating *within* a given operating system (OS) release. Eventually, however, most OS releases will reach **end-of-life (EOL)**, after which point they will no longer be supported. This applies to Qubes OS itself as well as OSes used in [templates](#) (and [standalones](#), if you have any).

**It's very important that you use only supported releases so that you continue to receive security updates.** This means that you *must* periodically upgrade Qubes OS and your templates before they reach EOL. You can always see which versions of Qubes OS and select templates are supported on [Supported releases](#).

In the case of Qubes OS itself, we will make an [announcement](#) when a supported Qubes OS release is approaching EOL and another when it has actually reached EOL, and we will provide [instructions for upgrading to the next stable supported Qubes OS release](#).

Periodic upgrades are also important for templates. For example, you might be using a [Fedora template](#). The [Fedora Project](#) is independent of the Qubes OS Project. They set their own [schedule](#) for when each Fedora release reaches EOL. You can always find out when an OS reaches EOL from the upstream project that maintains it. We also pass along any EOL notices we receive for official template OSes as a convenience to Qubes users (see the [supported template releases](#)).

The one exception to all this is the specific release used for dom0 (not to be confused with Qubes OS as a whole), which *doesn't have to be upgraded*.

### 1.12.11 How to back up, restore, and migrate

With Qubes, it's easy and secure to back up and restore your whole system, as well as to migrate between two physical machines.

These functions are integrated into the Qube Manager. There are also two command-line tools available that perform the same functions: `qvm-backup` and `qvm-backup-restore`.

It's extremely important to make regular backups of all the data you care about. This is true of all computing, not just the use of Qubes. Data loss can and does occur in myriad and unexpected ways. A standard recommendation is to make backups at least weekly: three copies in two different formats, one off-site.

## Backing up changes to dom0

When backing up dom0 using the Qubes backup tool (explained below), only the home directory is backed up. Therefore, if there are files outside of the home directory you wish to save, you should copy them into the home directory prior to creating a backup. Here is an example of how to back up Qubes config files and RPC policies:

```
$ mkdir -p ~/backup/etc/qubes/  
$ cp -a /etc/qubes/* ~/backup/etc/qubes/  
$ mkdir ~/backup/etc/qubes-rpc/  
$ cp -a /etc/qubes-rpc/* ~/backup/etc/qubes-rpc/
```

To restore these files, move them from the restored directory in dom0's home back to their appropriate locations in /etc/. Please note that any packages installed via the package manager in dom0 will not be backed up. Such packages will have to be reinstalled through the package manager when restoring on a fresh installation.

## Creating a backup

1. Go to **Applications menu -> System Tools -> Backup Qubes**. This brings up the **Qubes Backup VMs** window.
2. Move the VMs that you want to back up to the right-hand **Selected** column. VMs in the left-hand **Available** column will not be backed up. You may choose whether to compress backups by checking or unchecking the **Compress the backup** box. Normally this should be left on unless you have a specific reason otherwise. Once you have selected all desired VMs, click **Next**.
3. Select the destination for the backup: If you wish to send your backup to a (currently running) VM, select the VM in the drop-down box next to **Target app qube**. If you wish to send your backup to a *USB mass storage device*, you can use the directory selection widget to mount a connected device (under “Other locations” item on the left); or first mount the device in a VM, then select the mount point inside that VM as the backup destination. You must also specify a directory on the device or in the VM, or a command to be executed in the VM as a destination for your backup. For example, if you wish to send your backup to the ~/backups folder in the target VM, you would simply browse to it using the convenient directory selection dialog (...) at the right. This destination directory must already exist. If it does not exist, you must create it manually prior to backing up. By specifying the appropriate directory as the destination in a VM, it is possible to send the backup directly to, e.g., a USB mass storage device attached to the VM. Likewise, it is possible to enter any command as a backup target by specifying the command as the destination in the VM. This can be used to send your backup directly to, e.g., a remote server using SSH. **Note:** The supplied passphrase is used for **both** encryption/decryption and integrity verification. At this point, you may also choose whether to save your settings by checking or unchecking the **Save settings as default backup profile** box. **Warning: Saving the settings will result in your backup passphrase being saved in plaintext in dom0, so consider your threat model before checking this box.**
4. You will now see the summary of VMs to be backed up. If there are any issues preventing the backup, they will be listed here and the **Next** button grayed out.
5. When you are ready, click **Next**. Qubes will proceed to create your backup. Once the progress bar has completed, you may click **Finish**.
6. Test restore your backup. Follow the *restore procedure*, selecting **Verify backup integrity, do not restore the data**. This step is optional but strongly recommended. A backup is useless if you can't restore your data from it, and you can't be sure that your backup is good until you try to restore.

## Restoring from a backup

1. Go to **Applications menu -> System Tools -> Restore Backup**. This brings up the **Qubes Restore VMs** window.
2. Select the source location of the backup to be restored:
  - If your backup is located on a *USB mass storage device*, attach it first to another VM or select `sys-usb` in the next item.
  - If your backup is located in a (currently running) VM, select the VM in the drop-down box next to **app qube**.

You must also specify the directory and filename of the backup (or a command to be executed in a VM) in the **Backup file** field. If you followed the instructions in the previous section, “Creating a Backup,” then your backup is most likely in the location you chose as the destination in step 3. For example, if you had chosen the `~/backups` directory of a VM as your destination in step 3, you would now select the same VM and again browse to (using `...`) the `backups` folder. Once you’ve located the backup file, double-click it or select it and hit **OK**.

3. There are three options you may select when restoring from a backup:
  1. **ignore missing templates and net VMs**: If any of the VMs in your backup depended upon a NetVM or template that is not present in (i.e., “missing from”) the current system, checking this box will ignore the fact that they are missing and restore the VMs anyway and set them to use the default NetVM and system default template.
  2. **ignore username mismatch**: This option applies only to the restoration of dom0’s home directory. If your backup was created on a Qubes system which had a different dom0 username than the dom0 username of the current system, then checking this box will ignore the mismatch between the two usernames and proceed to restore the home directory anyway.
  3. **Verify backup integrity, do not restore the data**: This will scan the backup file for corrupted data. However, it does not currently detect if it is missing data as long as it is a correctly structured, non-corrupted backup file. See [issue #3498](#) for more details.
4. If your backup is encrypted, you must check the **Encrypted backup** box. If a passphrase was supplied during the creation of your backup (regardless of whether it is encrypted), then you must supply it here. **Note:** The passphrase which was supplied when the backup was created is used for **both** encryption/decryption and integrity verification. If the backup was not encrypted, the supplied passphrase is used only for integrity verification. All backups made from a Qubes R4.0 system will be encrypted.
5. You will now see the summary of VMs to be restored. If there are any issues preventing the restore, they will be listed here and the **Next** button grayed out.
6. When you are ready, click **Next**. Qubes will proceed to restore from your backup. Once the progress bar has completed, you may click **Finish**.

**Note:** When restoring from a dom0 backup, a new directory will be created in the current dom0 home directory, and the contents from the backup will be placed inside this new directory. This is intentional, as it allows users to have explicit control over which files and settings get applied in dom0. If the contents from the dom0 backup were instead to overwrite the existing files in dom0’s home directory, unexpected and undesired configuration changes could occur. However, if you do wish to move all files from the dom0 backup out of the subdirectory into your current dom0 home directory (overwriting any existing files in the process), you may do so by following the instructions [here](#). Just remember that this can cause unexpected and undesired configuration changes in dom0, depending on exactly which files you’re adding and replacing.

## Emergency backup recovery without qubes

The Qubes backup system has been designed with emergency disaster recovery in mind. No special Qubes-specific tools are required to access data backed up by Qubes. In the event a Qubes system is unavailable, you can access your data on any GNU/Linux system with the following procedure.

Refer to the following for emergency restore of a backup created on:

## Emergency backup recovery (v4)

This page describes how to perform an emergency restore of a backup created on Qubes R4.X (which uses backup format version 4).

The Qubes backup system has been designed with emergency disaster recovery in mind. No special Qubes-specific tools are required to access data backed up by Qubes. In the event a Qubes system is unavailable, you can access your data on any GNU/Linux system with the following procedure.

## Required `scrypt` Utility

In Qubes 4.X, backups are encrypted and integrity-protected with `scrypt`. You will need a copy of this utility in order to access your data. Since `scrypt` is not pre-installed on every GNU/Linux system, it is strongly recommended that you store a copy of it with your backups. If your distribution has `scrypt` packaged (e.g., Debian), you can install the package in the standard way using your distribution's package manager. Otherwise, you'll need to obtain a compiled binary (instructions below) or compile the program from source yourself. (Don't forget to *verify signatures* first!) Note that versions of `scrypt` up to 1.2.0 (inclusive) do not support the `-P` option for easier scripting, which means you'll need to enter the passphrase for each file separately, instead of using `echo ... | scrypt`.

Here are instructions for obtaining a compiled `scrypt` binary. This example uses an RPM-based system (Fedora), but the same general procedure should work on any GNU/Linux system.

1. If you're not on Qubes 4.X, *import and authenticate the Release 4 Signing Key*.

```
[user@restore ~]$ sudo rpm --import qubes-release-4-signing-key.asc
```

2. Download the `scrypt` RPM.

```
[user@restore ~]$ dnf download scrypt
```

or, if that doesn't work:

```
[user@restore ~]$ curl -O https://yum.qubes-os.org/r4.0/current/vm/fc28/rpm/scrypt-  
→ 1.2.1-1.fc28.x86_64.rpm
```

3. Verify the signature on the `scrypt` RPM.

```
[user@restore ~]$ rpm -K scrypt-*.rpm  
scrypt-*.rpm: digests signatures OK
```

The message `digests signatures OK` means that both the digest (i.e., the output of a hash function) and PGP signature verification were successful.

4. Install `rpmdevtools`.

```
[user@restore ~]$ sudo dnf install rpmdevtools
```

5. Extract the `scrypt` binary from the RPM.

```
[user@restore ~]$ rpmdev-extract scrypt-*.rpm
```

6. (Optional) Create an alias for the new binary.

```
[user@restore ~]$ alias scrypt="scrypt-*/usr/bin/scrypt"
```

## Emergency Recovery Instructions

**Note:** In the following example, the backup file is both *encrypted* and *compressed*.

1. Untar the main backup file.

```
[user@restore ~]$ tar -i -xvf qubes-backup-2015-06-05T123456
backup-header
backup-header.hmac
qubes.xml.000.enc
vm1/private.img.000.enc
vm1/private.img.001.enc
vm1/private.img.002.enc
vm1/icon.png.000.enc
vm1/firewall.xml.000.enc
vm1/whitelisted-appmenus.list.000.enc
dom0-home/dom0user.000.enc
```

**To extract only specific VMs:** Each VM in the backup file has its path listed in `qubes.xml.000.enc`. Decrypt it. (In this example, the password is `password`.)

```
[user@restore ~]$ cat backup-header | grep backup-id
backup-id=20190128T123456-1234
[user@restore ~]$ scrypt dec -P qubes.xml.000.enc qubes.xml.000
Please enter passphrase: 20190128T123456-1234!qubes.xml.000!password
[user@restore ~]$ tar -i -xvf qubes.xml.000
```

Now that you have the decrypted `qubes.xml.000` file, search for the `backup-path` property inside of it. With the `backup-path`, extract only the files necessary for your VM (`vmX`).

```
[user@restore ~]$ tar -i -xvf qubes-backup-2015-06-05T123456 \
    backup-header backup-header.hmac vmX/
```

2. Set the backup passphrase environment variable. While this isn't strictly required, it will be handy later and will avoid saving the passphrase in the shell's history.

```
[user@restore ~]$ read -r backup_pass
```

3. Verify the integrity of `backup-header`. For compatibility reasons, `backup-header.hmac` is an encrypted *and integrity protected* version of `backup-header`.

```
[user@restore ~]$ set +H
[user@restore ~]$ echo "backup-header!$backup_pass" |\
    scrypt dec -P backup-header.hmac backup-header.verified && \
    diff -qs backup-header backup-header.verified
Files backup-header and backup-header.verified are identical
```

**Note:** If this command fails, it may be that the backup was tampered with or is in a different format. In the latter case, look inside `backup-header` at the `version` field. If it contains a value other than `version=4`, go to the instructions for that format version:

- *Emergency Backup Recovery without Qubes (v2)*
- *Emergency Backup Recovery without Qubes (v3)*

4. Read `backup-header`:

```
[user@restore ~]$ cat backup-header
version=4
encrypted=True
compressed=True
compression-filter=gzip
backup_id=20161020T123455-1234
```

5. Set `backup_id` to the value in the last line of `backup-header`:

```
[user@restore ~]$ backup_id=20161020T123455-1234
```

6. Verify the integrity of your data, decrypt, decompress, and extract `private.img`:

```
[user@restore ~]$ find vm1 -name 'private.img.*.enc' | sort -V | while read f_enc; \
do \
    f_dec=${f_enc%.enc}; \
    echo "$backup_id!$f_dec!$backup_pass" | sscript dec -P $f_enc || break; \
done | gzip -d | tar -xv
vm1/private.img
```

If this pipeline fails, it is likely that the backup is corrupted or has been tampered with. **Note:** If your backup was compressed with a program other than `gzip`, you must substitute the correct compression program in the command above. This information is contained in `backup-header` (see step 4). For example, if your backup is compressed with `bzip2`, use `bzip2 -d` instead in the command above.

7. Mount `private.img` and access your data.

```
[user@restore vm1]$ sudo mkdir /mnt/img
[user@restore vm1]$ sudo mount -o loop vm1/private.img /mnt/img/
[user@restore vm1]$ cat /mnt/img/home/user/your_data.txt
This data has been successfully recovered!
```

8. Success! If you wish to recover data from more than one VM in your backup, simply repeat steps 6 and 7 for each additional VM. **Note:** You may wish to store a copy of these instructions with your Qubes backups in the event that you fail to recall the above procedure while this web page is inaccessible. All Qubes documentation, including this page, is available in plain text format in the following Git repository:

```
https://github.com/QubesOS/qubes-doc.git
```



## Emergency backup recovery (v3)

This page describes how to perform an emergency restore of a backup created on Qubes R2 or later (which uses backup format version 3).

The Qubes backup system has been designed with emergency disaster recovery in mind. No special Qubes-specific tools are required to access data backed up by Qubes. In the event a Qubes system is unavailable, you can access your data on any GNU/Linux system with the following procedure.

**Note:** In the following example, the backup file is both *encrypted* and *compressed*.

1. Untar the main backup file.

```
[user@restore ~]$ tar -i -xvf qubes-backup-2015-06-05T123456
backup-header
backup-header.hmac
qubes.xml.000
qubes.xml.000.hmac
vm1/private.img.000
vm1/private.img.000.hmac
vm1/icon.png.000
vm1/icon.png.000.hmac
vm1/firewall.xml.000
vm1/firewall.xml.000.hmac
vm1/whitelisted-appmenus.list.000
vm1/whitelisted-appmenus.list.000.hmac
dom0-home/dom0user.000
dom0-home/dom0user.000.hmac
```

2. Set the backup passphrase environment variable. While this isn't strictly required, it will be handy later and will avoid saving the passphrase in the shell's history.

```
[user@restore ~]$ read -r backup_pass
```

3. Verify the integrity of the backup-header file, which contains basic information about your backup.

```
[user@restore ~]$ openssl dgst -sha512 -hmac "$backup_pass" backup-header
HMAC-SHA512(backup-header)=
5b266783e116fe3b2601a54c249ca5f5f96d421dfe6828eeaeb2dcd014e9e945c27b3d7b0f952f5d55c927318906d9c30
[user@restore ~]$ cat backup-header.hmac
(stdin)=
5b266783e116fe3b2601a54c249ca5f5f96d421dfe6828eeaeb2dcd014e9e945c27b3d7b0f952f5d55c927318906d9c30
```

**Note:** The hash values should match. If they do not match, then the backup file may have been tampered with, or there may have been a storage error. **Note:** If your backup was hashed with a message digest algorithm other than sha512, you must substitute the correct message digest command. This information is contained in the backup-header file (see step 4), however it is not recommended to open this file until its integrity and authenticity has been verified (the current step). A complete list of supported message digest algorithms can be found with `openssl list-message-digest-algorithms`.

4. Read the backup-header. You'll need some of this information later. The file will look similar to this:

```
[user@restore ~]$ cat backup-header
version=3
hmac-algorithm=SHA512
crypto-algorithm=aes-256-cbc
```

(continues on next page)

(continued from previous page)

```
encrypted=True
compressed=True
compression-filter=gzip
```

**Note:** If you see `version=2` here, go to *Emergency Backup Recovery - format version 2* instead.

5. Verify the integrity of the `private.img` file which houses your data.

```
[user@restore ~]$ cd vm1/
[user@restore vm1]$ openssl dgst -sha512 -hmac "$backup_pass" private.img.000
HMAC-SHA512(private.img.000)=
cf83e1357eeb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85f2b0ff8318d2877eec2f0
[user@restore vm1]$ cat private.img.000.hmac
(stdin)=
cf83e1357eeb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85f2b0ff8318d2877eec2f0
```

**Note:** The hash values should match. If they do not match, then the backup file may have been tampered with, or there may have been a storage error. **Note:** If your backup was hashed with a message digest algorithm other than `sha512`, you must substitute the correct message digest command. This information is contained in the `backup-header` file (see step 4). A complete list of supported message digest algorithms can be found with `openssl list-message-digest-algorithms`.

6. Decrypt the `private.img` file.

```
[user@restore vm1]$ find -name 'private.img.*[0-9]' | sort -V | xargs cat | openssl
enc -d -md MD5 -pass pass:"$backup_pass" -aes-256-cbc -out private.img.dec
```

**Note:** If your backup was encrypted with a cipher algorithm other than `aes-256-cbc`, you must substitute the correct cipher command. This information is contained in the `backup-header` file (see step 4). A complete list of supported cipher algorithms can be found with `openssl list-cipher-algorithms`.

7. Decompress the decrypted `private.img` file.

```
[user@restore vm1]$ zforce private.img.dec
private.img.dec -- replaced with private.img.dec.gz
[user@restore vm1]$ gunzip private.img.dec.gz
```

**Note:** If your backup was compressed with a program other than `gzip`, you must substitute the correct compression program. This information is contained in the `backup-header` file (see step 4). For example, if you used `bzip2`, then you should do this:

```
[user@restore vm1]$ mv private.img.dec private.img.dec.bz2
[user@restore vm1]$ bunzip2 private.img.dec.bz2
```

8. Untar the decrypted and decompressed `private.img` file.

```
[user@restore vm1]$ tar -xvf private.img.dec
vm1/private.img
```

9. Mount the `private.img` file and access your data.

```
[user@restore vm1]$ sudo mkdir /mnt/img
[user@restore vm1]$ sudo mount -o loop vm1/private.img /mnt/img/
[user@restore vm1]$ cat /mnt/img/home/user/your_data.txt
This data has been successfully recovered!
```

10. Success! If you wish to recover data from more than one VM in your backup, simply repeat steps 5–9 for each additional VM. **Note:** You may wish to store a copy of these instructions with your Qubes backups in the event that you fail to recall the above procedure while this web page is inaccessible. All Qubes documentation, including this page, is available in plain text format in the following Git repository:

```
https://github.com/QubesOS/qubes-doc.git
```

## Emergency backup recovery (v2)

This page describes how to perform emergency restore of backup created on Qubes R2 Beta3 or earlier (which uses backup format 2).

The Qubes backup system has been designed with emergency disaster recovery in mind. No special Qubes-specific tools are required to access data backed up by Qubes. In the event a Qubes system is unavailable, you can access your data on any GNU/Linux system with the following procedure.

**Note:** In the following example, the backup file is assumed to be both encrypted and compressed.

1. Untar the main backup file.

```
[user@restore ~]$ tar -i -xvf qubes-backup-2013-12-26-123456
backup-header
backup-header.hmac
qubes.xml.000
qubes.xml.000.hmac
vm1/private.img.000
vm1/private.img.000.hmac
vm1/icon.png.000
vm1/icon.png.000.hmac
vm1/firewall.xml.000
vm1/firewall.xml.000.hmac
vm1/whitelisted-appmenus.list.000
vm1/whitelisted-appmenus.list.000.hmac
dom0-home/dom0user.000
dom0-home/dom0user.000.hmac
```

2. Set the backup passphrase environment variable. While this isn't strictly required, it will be handy later and will avoid saving the passphrase in the shell's history.

```
[user@restore ~]$ read -r backup_pass
```

3. Verify the integrity of the `private.img` file which houses your data.

```
[user@restore ~]$ cd vm1/
[user@restore vm1]$ openssl dgst -sha512 -hmac "$backup_pass" private.img.000
HMAC-SHA512(private.img.000)=
cf83e1357eeb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85f2b0ff8318d2877eec2f
[user@restore vm1]$ cat private.img.000.hmac
(stdin)=
cf83e1357eeb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85f2b0ff8318d2877eec2f
```

**Note:** The hash values should match. If they do not match, then the backup file may have been tampered with, or there may have been a storage error.

**Note:** If your backup was hashed with a message digest algorithm other than sha512, you must substitute the correct message digest command. A complete list of supported message digest algorithms can be found with `openssl list-message-digest-algorithms`.

4. Decrypt the `private.img` file.

```
[user@restore vm1]$ openssl enc -d -md MD5 -pass pass:"$backup_pass" -aes-256-cbc -
→in private.img.000 -out private.img.dec.000
```

**Note:** For multi-part files, a loop can be used:

```
find -name 'private.img.*' | sort -V | while read f; do
  openssl enc -d -md MD5 -pass pass:"$backup_pass" -aes-256-cbc -in $f -out
  ${f%.img/.img.dec}
done
```

**Note:** If your backup was encrypted with a cipher algorithm other than aes-256-cbc, you must substitute the correct cipher command. A complete list of supported cipher algorithms can be found with `openssl list-cipher-algorithms`.

5. Decompress the decrypted `private.img` file.

```
[user@restore vm1]$ zforce private.img.dec.*
[user@restore vm1]$ gunzip private.img.dec.000.gz
```

**Note:** If your backup was compressed with a program other than gzip, you must substitute the correct compression program.

6. Untar the decrypted and decompressed `private.img` file.

```
[user@restore vm1]$ tar -M -xvf private.img.dec.000
vm1/private.img
```

**Note:** For multi-part files, a script is required:

1. Create a new-volume-script:

```
#!/bin/sh
name=`expr $STAR_ARCHIVE : '\(.*\)\. *.*'`
suffix=`printf %03d $[ $STAR_VOLUME - 1 ]`
echo $name.$suffix >&$STAR_FD
```

2. `chmod +x new-volume-script`.
3. `tar --new-volume-script=./new-volume-script -xvf private.img.dec.000`. (The `--new-volume-script` option enables multi-volume untaring.)
7. Mount the `private.img` file and access your data.

```
[user@restore vm1]$ sudo mkdir /mnt/img
[user@restore vm1]$ sudo mount -o loop vm1/private.img /mnt/img/
[user@restore vm1]$ cat /mnt/img/home/user/your_data.txt
This data has been successfully recovered!
```

**Note:** You may wish to store a plain text copy of these instructions with your Qubes backups in the event that you fail to recall the above procedure while this web page is inaccessible. You may obtain a plaintext version of this file in Git repository housing all the documentation on [Github](#)

## Migrating between two physical machines

In order to migrate your Qubes system from one physical machine to another, simply follow the backup procedure on the old machine, install Qubes on the new machine, and follow the restoration procedure on the new machine. All of your settings and data will be preserved!

## Choosing a backup passphrase

Here are some things to consider when selecting a passphrase for your backups:

- If you plan to store the backup for a long time or on third-party servers, you should make sure to use a very long, high-entropy passphrase. (Depending on the decryption passphrase you use for your system drive, this may necessitate selecting a stronger passphrase. If your system drive decryption passphrase is already sufficiently strong, it may not.)
- An adversary who has access to your backups may try to substitute one backup for another. For example, when you attempt to retrieve a recent backup, the adversary may instead give you a very old backup containing a compromised VM. If you're concerned about this type of attack, you may wish to use a different passphrase for each backup, e.g., by appending a number or date to the passphrase.
- If you're forced to enter your system drive decryption passphrase in plain view of others (where it can be shoulder-surfed), then you may want to use a different passphrase for your backups (even if your system drive decryption passphrase is already maximally strong). On the other hand, if you're careful to avoid shoulder-surfing and/or have a passphrase that's difficult to detect via shoulder-surfing, then this may not be a problem for you.

## Notes

- For the technical details of the backup system, please refer to [this thread](#).
- If working with symlinks, note the issues described in [this thread](#).

## 1.12.12 How to copy and paste text

*This page is about copying and pasting plain text. If you wish to copy more complex data, such as rich text or images, see [copying and moving files between qubes](#) . For dom0, see [copying from \(and to\) dom0](#) .*

Qubes OS features a secure inter-qube clipboard that allows you to copy and paste text between qubes.

In order to copy text from qube A to qube B:

1. Select text from the source app in qube A, then copy it normally (e.g., by pressing Ctrl+C).
2. With the source app in qube A still in focus, press Ctrl+Shift+C. This copies the text from qube A's clipboard to the inter-qube clipboard.
3. Select the target app in qube B and press Ctrl+Shift+V. This copies the text from the inter-qube clipboard to qube B's clipboard and clears the inter-qube clipboard, ensuring that only qube B will have access to the copied text.
4. Paste the text in the target app in qube B normally (e.g., by pressing Ctrl+V).

This process might look complicated at first glance, but in practice it is actually very easy and fast once you get used to it. At the same time, it provides you with full control over exactly which qube receives the content of the inter-qube clipboard every time.

### Security

The inter-qube clipboard system is secure because it doesn't allow any qube other than your selected target to steal any contents from the inter-qube clipboard. Without such a system in place, any password you were to copy from the password manager in your vault qube to another qube, for example, would immediately be leaked to every other running qube in the system, including qubes that are untrusted by default, such as `sys-net`. By giving you precise control over exactly which qube receives the inter-qube clipboard content, then immediately wiping the inter-qube clipboard afterward, Qubes OS protects the confidentiality of the text being copied.

However, one should keep in mind that performing a copy and paste operation from *less trusted* to *more trusted* qube is always potentially insecure, since the data that we copy could exploit some hypothetical bug in the target qube. For example, the seemingly-innocent link that we copy from an untrusted qube could turn out to be a large buffer of junk that, when pasted into the target qube's word processor, could exploit a hypothetical bug in the undo buffer. This is a general problem and applies to any data transfer from *less trusted* to *more trusted* qubes. It even applies to copying files between physically separate (air-gapped) machines. Therefore, you should always copy clipboard data only from *more trusted* to *less trusted* qubes.

See also [this article](#) for more information on this topic, and some ideas of how we might solve this problem in some future version of Qubes, as well as [this message](#) from qubes-devel.

### Focus stealing

The above discussion assumes that you control which window is focused in dom0 at the time of the paste. However, if your dom0 window manager is configured to give focus to newly created windows (which, as of Qubes 4.0, is true in the default install with Xfce), then a malicious qube could “steal the focus” by creating a window just before you press Ctrl+Shift+V, and it would receive the data instead of your intended target. (Focus stealing is a risk any time you are typing confidential data, but a Qubes clipboard paste probably presents the greatest risk of leaking an entire password before you have time to react.) You may be able to mitigate this risk by changing the window manager configuration. For example, with Xfce, you could run `xfwm4-settings` in dom0, go to the “Focus” tab, and un-check “Automatically give focus to newly created windows”. However, we have not confirmed whether such settings are sufficient to prevent a malicious qube from stealing the focus in all cases.

### Clipboard automatic policy enforcement

The Qubes clipboard *RPC policy* is configurable in:

```
/etc/qubes-rpc/policy/qubes.ClipboardPaste
```

You may wish to configure this policy in order to prevent user error. For example, if you are certain that you never wish to paste *into* your “vault” app qube (and it is highly recommended that you do not), then you should edit the policy as follows:

```
@anyvm vault deny
@anyvm @anyvm ask
```

## Automatic clipboard wiping

By default data pasted into a qube will remain there until user copies something else or restarts the qube. It's possible to make the `qubes-gui` process inside a qube wipe the clipboard automatically after a minute from the last paste operation. This helps protect users from accidentally pasting the old content of the clipboard like a password in the wrong place like a browser search bar. Since qubes don't share the same clipboard, software like KeePassXC isn't able to automatically wipe the clipboard of other qubes.

To enable automatic wiping of the clipboard after a minute use `qvm-service`:

```
qvm-service --enable VMNAME gui-agent-clipboard-wipe
```

## Shortcut configuration

The copy/paste shortcuts are configurable via `qvm-features`, e.g.

```
qvm-features dom0 gui-default-secure-copy-sequence 'Mod4-c'
qvm-features dom0 gui-default-secure-paste-sequence 'Mod4-v'
```

would change the *copy/paste to global clipboard* to the Win key plus c for copy, or v for paste.

You need to restart Qubes for the changes to take effect.

### 1.12.13 How to copy and move files

*This page is about copying and moving files. If you wish to simply copy and paste text, that can be done more easily using the inter-qube clipboard. See [copying and pasting text between qubes](#). For `dom0`, see [copying from \(and to\) dom0](#).*

Qubes OS supports the secure copying and moving of files and directories (folders) between qubes.

For simplicity, these instructions will refer to copying/moving a single file, but they apply equally well to groups of files and directories, which are copied recursively.

1. Open a file manager in the qube containing the file you wish to copy (the source qube), right-click on the file you wish to copy or move, and select **Copy to Other AppVM...** or **Move to Other AppVM...**
2. A dialog box will appear in `dom0` asking for the name of the target qube (qube B). Enter or select the desired destination qube name.
3. If the target qube is not already running, it will be started automatically, and the file will be copied there. It will show up in this directory (which will automatically be created if it does not already exist):

```
/home/user/QubesIncoming/<source_qube>/<filename>
```

If you selected **Move** rather than **Copy**, the original file in the source qube will be deleted. (Moving a file is equivalent to copying the file, then deleting the original.)

4. If you wish, you may now move the file in the target qube to a different directory and delete the `/home/user/QubesIncoming/` directory when no longer needed.

The same operations are also available via these command-line tools:

```
qvm-copy [--without-progress] file [file]+
```

```
qvm-move [--without-progress] file [file]+
```

## Security

The inter-qube file copy system is secure because it doesn't allow other qubes to steal the files that are being copied, and it doesn't allow the source qube to overwrite arbitrary files on the destination qube. Moreover, this system doesn't use any sort of virtual block device for file copy. Instead, we use Xen shared memory, which eliminates a lot of processing of untrusted data. For example, the receiving qube is *not* forced to parse untrusted partitions or file systems. In this respect, the inter-qube file copy system provides even more security than file copy between two physically separated (air-gapped) machines! (See [Software compartmentalization vs. physical separation](#) for more on this.)

However, one should keep in mind that performing a data transfer from *less trusted* to *more trusted* qubes is always potentially insecure if the data will be parsed in the target qube. This is because the data that we copy could try to exploit some hypothetical bug in software running in the target qube. For example, a seemingly-innocent JPEG that we copy from an untrusted qube might contain a specially-crafted exploit for a bug in a JPEG-parsing application in the target qube. This is a general problem and applies to any data transfer from *less trusted* to *more trusted* qubes. It even applies to the scenario of copying files between air-gapped machines. Therefore, you should always copy data only from *more trusted* to *less trusted* qubes.

See also [this article](#) for more information on this topic, and some ideas of how we might solve this problem in some future version of Qubes.

### 1.12.14 How to copy from dom0

This page covers copying files and clipboard text between *dom0* and *domUs*. Since dom0 is special, the processes are different from *copying and pasting text between qubes* and *copying and moving files between qubes*.

#### Copying from dom0

##### Copying files from dom0

To copy a file from dom0 to a VM, simply use `qvm-copy-to-vm`:

```
qvm-copy-to-vm <target_vm> <file>
```

The file will arrive in the target VM in the `/home/user/QubesIncoming/dom0/` directory.

#### Copying and pasting clipboard text from dom0

Use the **Qubes Clipboard** widget:

1. Copy text to the clipboard normally in dom0 (e.g., by pressing Ctrl+C).
2. Click the **Qubes Clipboard** icon in the Notification Area.
3. Click “Copy dom0 clipboard”. This displays a notification that text has been copied to the inter-qube clipboard.
4. Press Ctrl+Shift+V in the target qube. This pastes the inter-qube clipboard contents into the target qube's normal clipboard.
5. Paste normally within that qube (e.g., by pressing Shift+V).

Alternatively, you can put your text in a file, then *copy it as a file*. Or, you can write the data you wish to copy into `/var/run/qubes/qubes-clipboard.bin`, then `echo -n dom0 > /var/run/qubes/qubes-clipboard.bin`. source. Then use Ctrl+Shift+V to paste the data to the target qube.



## Copying logs from dom0

In order to easily copy/paste the contents of logs from dom0 to the inter-VM clipboard, you can simply:

1. Right-click on the desired qube in the Qube Manager.
2. Click “Logs.”
3. Click on the desired log.
4. Click “Copy to Qubes clipboard.”

You may now paste the log contents in qube as you normally would (e.g., Ctrl+Shift+V, then Ctrl+V).

## Copying to dom0

Copying anything into dom0 is not advised, since doing so can compromise the security of your Qubes system. For this reason, there is no simple means of copying anything into dom0, unlike *copying from dom0*.

There should normally be few reasons for the user to want to copy anything from domUs to dom0, as dom0 only acts as a “thin trusted terminal”, and no user applications run there. Sometimes, new users feel the urge to copy a desktop wallpaper image into dom0, but that is not necessary. A safer approach is simply to display the image in *full-screen mode* in an app qube, then take a screenshot from dom0, which results in exactly the image needed for a wallpaper, created securely and natively in dom0.

If you are determined to copy some files to dom0 anyway, you can use the following method. (If you want to copy text, first save it into a text file.) Run this command in a dom0 terminal:

```
qvm-run --pass-io <src-vm> 'cat /path/to/file_in_src_domain' > /path/to/file_name_in_dom0
```

Note that you can use the same method to copy files from dom0 to domUs (if, for some reason, you don’t want to use `qvm-copy-to-vm`):

```
cat /path/to/file_in_dom0 | qvm-run --pass-io <dest-vm> 'cat > /path/to/file_name_in_
↪ appvm'
```

## 1.12.15 How to install software

When you wish to install software in Qubes OS, you should generally install it in a *template*. For installing templates themselves, see *how to install a template*. Advanced users may also be interested in learning how to install software in *standalones* and *dom0*.

Qubes OS is effectively a “meta” operating system (OS) that can run almost any arbitrary OS inside of itself. For example, the way software is normally installed in a Linux distribution (“distro”) is quite different from the way software is normally installed in Windows. This isn’t up to Qubes. Qubes is just the framework in which you’re running these other OSes. Therefore, if you want to install software in a Linux template, for example, you should do so in whatever way is normal for that Linux distro. Most Linux software is distributed via *packages*, which are stored in *software repositories* (“repos”). *Package managers* handle downloading, installing, updating, and removing packages. (Again, none of this is Qubes-specific.) If you’re not familiar with how software is normally installed in Linux distros via package managers or the software you want doesn’t seem to be available in your distro’s repos (or you’re in another situation not covered on this page), please read this *community guide to installing software in Qubes*.

The following instructions explain how to permanently install new software in a template. There are different instructions for software from the default repositories and all other software. (If you’re not sure, try the default repositories first.)

## Installing software from default repositories

1. Start the template.
2. Start either a terminal (e.g. `gnome-terminal`) or a dedicated software management application, such as `gpk-application`.
3. Install software as normally instructed inside that operating system, e.g.:
  - Fedora: `sudo dnf install <PACKAGE_NAME>`
  - Debian: `sudo apt install <PACKAGE_NAME>`
4. Shut down the template.
5. Restart all qubes based on the template.
6. (Recommended) In the relevant qubes' **Settings > Applications** tab, select the new application(s) from the list, and press **OK**. These new shortcuts will appear in the Applications Menu. (If you encounter problems, see [here](#) for troubleshooting.)

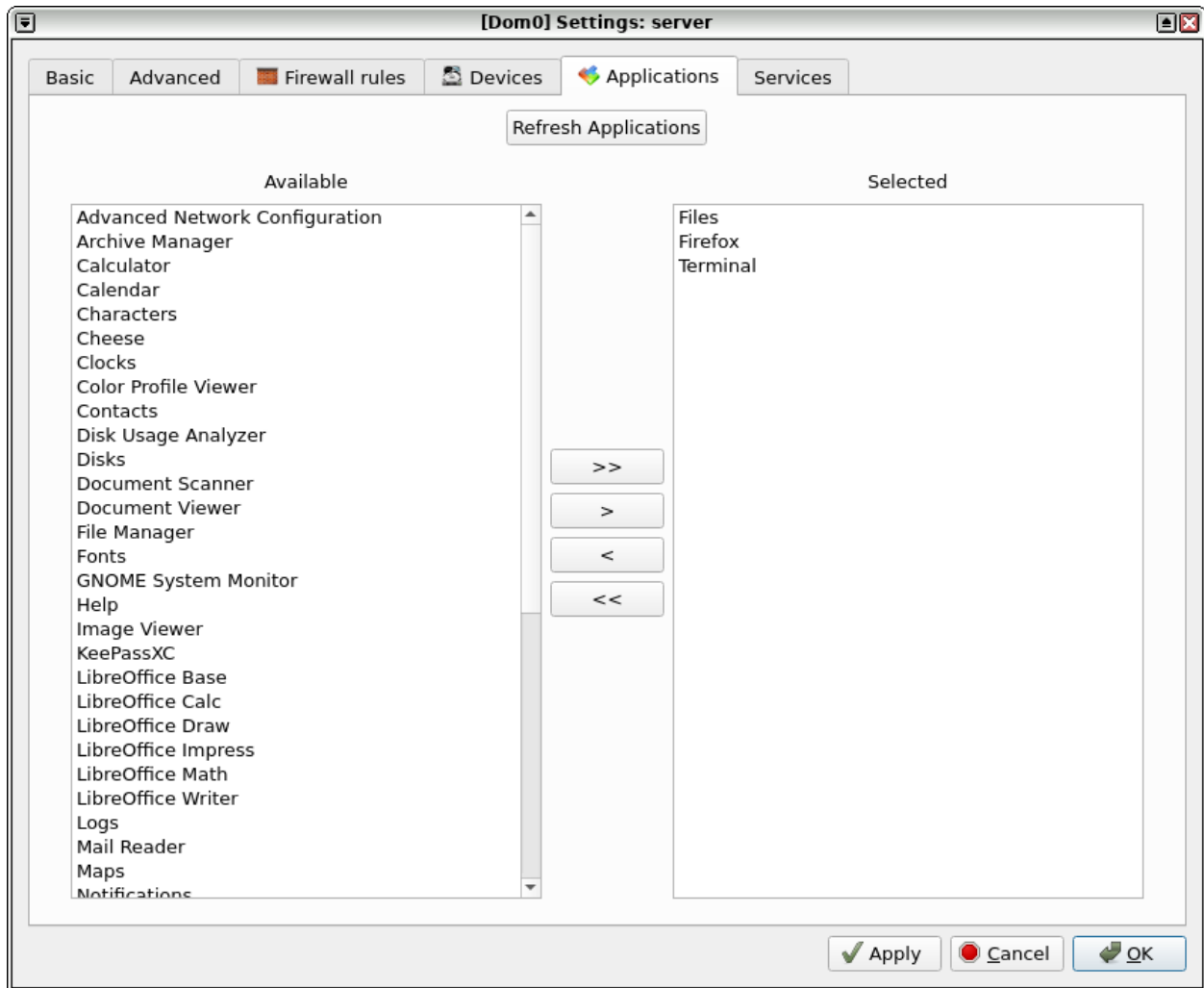


Fig. 1: The Applications tab in Qube Settings

## Installing software from other sources

**Warning:** This method gives your template direct network access, which is *risky*. This method is **not** recommended for trusted templates. Moreover, depending on how you install this software, it may not get updated automatically when you *update Qubes normally*, which means you may have to update it manually yourself.

Some software is not available from the default repositories and must be downloaded and installed from another source. This method assumes that you're trying to follow the instructions to install some piece of software in a normal operating system, except that operating system is running as a template in Qubes OS.

1. (Recommended) Clone the desired template (since this new template will probably be less trusted than the original).
2. (Recommended) In the new template's **Settings > Basic** tab, change the color label from black to red (or another color that signifies to you that the template is less trusted).
3. In the new template's **Settings > Basic** tab, change the **Networking** value from default (none) (current) to sys-firewall (or whichever network-providing qube you wish to use).
4. (Recommended) In the new template's **Settings > Firewall rules** tab, select "Limit outgoing Internet connections to..." and tick "Allow full access for 5 min." (This can help in case you forget to remove network access later.)
5. Follow the normal instructions for installing your software in the new template. For example, open a terminal and enter the commands as instructed. **Warning:** If you don't fully understand the commands you're entering, then this can be extremely risky, and the template should be regarded as *completely untrusted*.
6. (Recommended) In the new template's **Settings > Basic** tab, change the **Networking** value from sys-firewall (current) (or whichever network-providing qube you chose) back to default (none).
7. Shut down the new template.
8. Create or assign your desired app qubes to use the new template. If any app qubes were already assigned to the new template, restart them.
9. (Recommended) In the relevant qubes' **Settings > Applications** tab, select the new application(s) from the list, and press **OK**. These new shortcuts will appear in the Applications Menu. (If you encounter problems, see [here](#) for troubleshooting.)

## Troubleshooting

If things are still not working as expected:

- Review the instructions very carefully, making sure you follow each step.
- Make sure you **shut down the template after installing your software**.
- Make sure you **restart your app qube after shutting down your template**.
- Make sure your app qube is assigned to the right template.
- If your software requires special files or directories to be persistent, and you're an advanced user, see *standalones and HVMs* and *how to make any file persistent (bind-dirs)*.
- *Ask for help.*

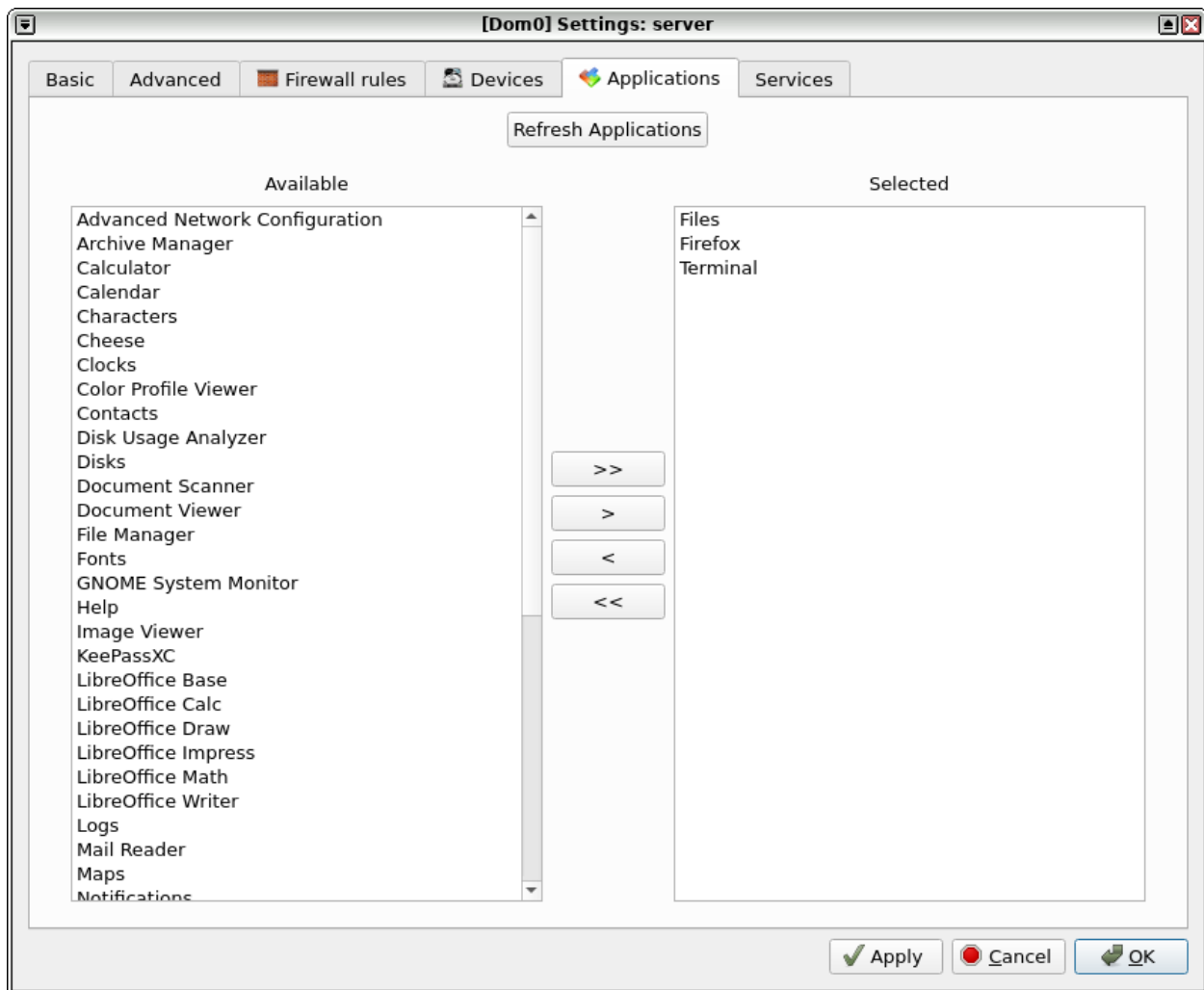


Fig. 2: The Applications tab in Qube Settings

## How to update software

Please see *How to Update*.

## Why don't templates have network access?

In order to protect you from performing risky activities in templates, they do not have normal network access by default. Instead, templates use an *updates proxy* that allows you to install and update software using the distribution package manager without giving the template direct network access. **The updates proxy is already setup to work automatically out-of-the-box and requires no special action from you.** Most users should simply follow the normal instructions for *installing software from default repositories* and *updating* software. If your software is not available in the default repositories, see *installing software from other sources*.

## Advanced

The following sections cover advanced topics pertaining to installing and updating software in domUs.

## Testing repositories

If you wish to install updates that are still in *testing*, you must enable the appropriate testing repositories.

**Note:** The following repos are in templates and standalones. For dom0 testing repos, see *here*. For testing new templates, please see *here*.

## Fedora

There are three Qubes VM testing repositories (where \* denotes the Release):

- `qubes-vm-*-current-testing` – testing packages that will eventually land in the stable (`current`) repository
- `qubes-vm-*-security-testing` – a subset of `qubes-vm-*-current-testing` that contains packages that qualify as security fixes
- `qubes-vm-*-unstable` – packages that are not intended to land in the stable (`qubes-vm-*-current`) repository; mostly experimental debugging packages

To temporarily enable any of these repos, use the `--enablerepo=<repo-name>` option. Example commands:

```
sudo dnf upgrade --enablerepo=qubes-vm-*-current-testing
sudo dnf upgrade --enablerepo=qubes-vm-*-security-testing
sudo dnf upgrade --enablerepo=qubes-vm-*-unstable
```

To enable or disable any of these repos permanently, change the corresponding `enabled` value to 1 in `/etc/yum.repos.d/qubes-*.repo`.

### Debian

Debian also has three Qubes VM testing repositories (where \* denotes the Release):

- **\*-testing** – testing packages that will eventually land in the stable (current) repository
- **\*-securitytesting** – a subset of \*-testing that contains packages that qualify as security fixes
- **\*-unstable** – packages that are not intended to land in the stable repository; mostly experimental debugging packages

To enable or disable any of these repos permanently, uncomment the corresponding deb line in `/etc/apt/sources.list.d/qubes-r*.list`.

### Standalones

The process for installing and updating software in *standalones* is the same as described above for templates, except no qubes are based on standalones, so there are no other qubes to restart.

### RPMFusion for Fedora templates

If you would like to enable the [RPM Fusion](#) repositories, open a Terminal of the template and type the following commands, depending on which RPM Fusion repositories you wish to enable (see [RPM Fusion](#) for details):

```
sudo dnf config-manager --set-enabled rpmpfusion-free
sudo dnf config-manager --set-enabled rpmpfusion-free-updates
sudo dnf config-manager --set-enabled rpmpfusion-nonfree
sudo dnf config-manager --set-enabled rpmpfusion-nonfree-updates
sudo dnf upgrade --refresh
```

This will permanently enable the RPM Fusion repos. If you install software from here, it's important to keep these repos enabled so that you can receive future updates. If you only enable these repos temporarily to install a package the Qubes update mechanism may persistently notify you that updates are available, since it cannot download them.

### Reverting changes to a template

Perhaps you've just updated your template, and the update broke your template. Or perhaps you've made a terrible mistake, like accidentally confirming the installation of an unsigned package that could be malicious. If you want to undo changes to a template, there are three basic methods:

1. **Root revert.** This is appropriate for misconfigurations, but not for security concerns. It will preserve your customizations.
2. **Reinstall the template.** This is appropriate for both misconfigurations and security concerns, but you will lose all customizations.
3. **Full revert.** This is appropriate for both misconfigurations and security concerns, and it can preserve your customizations. However, it is a bit more complex.

## Root revert

**Important:** This command will roll back any changes made *during the last time the template was run, but **not** before*. This means that if you have already restarted the template, using this command is unlikely to help, and you'll likely want to reinstall it from the repository instead. On the other hand, if the template is already broken or compromised, it won't hurt to try reverting first. Just make sure to **back up** all of your data and changes first!

1. Shut down <template>. If you've already just shut it down, do **not** start it again (see above).
2. In a dom0 terminal:

```
qvm-volume revert <template>:root
```

## Reinstall the template

Please see [How to Reinstall a template](#).

## Full revert

This is like the simple revert, except:

- You must also revert the private volume with `qvm-volume revert <template>:private`. This requires you to have an old revision of the private volume, which does not exist with the current default config. However, if you don't have anything important in the private volume (likely for a template), then you can work around this by just resetting the private volume with `qvm-volume import --no-resize <template>:private /dev/null`.
- The saved revision of the volumes must be uncompromised. With the default `revisions_to_keep=1` for the root volume, you must **not** have started the template since the compromising action.

## Updates proxy

Updates proxy is a service which allows access from package managers configured to use the proxy by default, but can be used by any other program that accepts proxy arguments. The purpose of the proxy, instead of direct network access, is meant to mitigate user errors of using applications such as the browser in the template. Not necessarily what part of the network they can access, but only to applications trusted by the user, configured to use the proxy. The http proxy (tinyproxy) does not filter traffic because it is hard to list all the repository mirrors and keep that list up to date). it also does not cache anything.

The proxy is running in selected VMs (by default all the NetVMs (1)) and intercepts traffic directed to 127.0.0.1:8082. Thanks to such configuration all the VMs can use the same proxy address. If the VM is configured to have access to the updates proxy (2), the startup scripts will automatically configure dnf/apt to really use the proxy (3). Also access to updates proxy is independent of any other firewall settings (VM will have access to updates proxy, even if policy is set to block all the traffic).

There are two services (`qvm-service`, [service framework](#)):

1. `qubes-updates-proxy` (and its deprecated name: `qubes-yum-proxy`) - a service providing a proxy for templates - by default enabled in NetVMs (especially: sys-net)
2. `updates-proxy-setup` (and its deprecated name: `yum-proxy-setup`) - use a proxy provided by another VM (instead of downloading updates directly), enabled by default in all templates

Both the old and new names work. The defaults listed above are applied if the service is not explicitly listed in the services tab.

## Technical details

The updates proxy uses RPC/qrexec. The proxy is configured in qrexec policy in dom0: `/etc/qubes-rpc/policy/qubes.UpdatesProxy`. By default this is set to `sys-net` and/or `sys-whonix`, depending on firstboot choices. This new design allows for templates to be updated even when they are not connected to any NetVM.

Example policy file in R4.1 (with Whonix installed, but not set as default UpdateVM for all templates):

```
# any VM with tag `whonix-updatevm` should use `sys-whonix`; this tag is added to `whonix-
↪gw` and `whonix-ws` during installation and is preserved during template clone
@tag:whonix-updatevm @default allow,target=sys-whonix
@tag:whonix-updatevm @anyvm deny

# other templates use sys-net
@type:TemplateVM @default allow,target=sys-net
@anyvm @anyvm deny
```

## Installing Snap Packages

Snap packages do not use the normal update channels for Debian and Fedora (apt and dnf) and are often installed as the user rather than as root. To support these in an app qube you need to take the following steps:

1. In the **template** you must install `snapt` and `qubes-snapt-helper`. Open a terminal in the template and run:

```
[user@fedora-36-snap-demo ~]$ sudo dnf install snapt qubes-snapt-helper
Last metadata expiration check: 0:33:05 ago on Thu 03 Nov 2022 04:34:06.
Dependencies resolved.

=====
Package                Arch    Version
↪Repository            Size
=====
Installing:
snapt                  x86_64  2.56.2-4.fc36      updates ↪
↪                14 M
qubes-snapt-helper     noarch  1.0.4-1.fc36      qubes-vm-
↪r4.1-current  10 k
Installing dependencies:
[...]

Transaction Summary
=====
Install 19 Packages

Total download size: 27 M
Installed size: 88 M
Is this ok [y/N]: y

Downloading Packages:
[..]
Failed to resolve booleanif statement at /var/lib/selinux/targeted/tmp/modules/200/
↪snappy/cil:1174
/usr/sbin/semodule: Failed!
[...]
```

(continues on next page)



(continued from previous page)

```
Last metadata expiration check: 0:33:05 ago on Thu 03 Nov 2022 04:34:06.
Notifying dom0 about installed applications
```

```
Installed:
```

```
  snapd-2.56.2-4.fc36.x86_64                      qubes-
  ↪ snapd-helper-1.0.4-1.fc36.noarch
  [...]
Complete!
```

You may see the following message:

```
Failed to resolve booleanif statement at /var/lib/selinux/targeted/tmp/modules/200/
↪ snappy/cil:1174
/usr/sbin/semodule: Failed!
```

This is expected and you can safely continue. Shutdown the template:

```
[user@fedora-36-snap-demo ~]$ sudo shutdown -h now
```

2. Now open the **app qube** in which you would like to install the Snap application and run a terminal:

```
[user@snap-demo-app qube ~]$ snap install <package>
```

When the install is complete you can close the terminal window.

3. Refresh the Applications list for the app qube. In the Qubes Menu for the **app qube\*** launch the Qube Settings. Then go to the Applications tab and click “Refresh Applications” The refresh will take a few minutes; after it’s complete the Snap app will appear in the app qube’s list of available applications. At this point the snap will be persistent within the app qube and will receive updates when the app qube is running.

## Autostarting Installed Applications

If you want a desktop app to start automatically every time a qube starts you can create a link to it in the `~/.config/autostart` directory of the **app qube**. This might be useful for Qubes that you set to automatically start on boot or for Qubes that have a set of apps you typically use all day, such as a chat app.

1. Open a terminal in the **app qube** where you would like the app to launch.
2. List the names of the available desktop shortcuts by running the command `ls /usr/share/applications` and find the exact name of the shortcut to the app you want to autostart:

```
[user@example-app qube ~]$ ls /usr/share/applications/
bluetooth-sendto.desktop
eog.desktop
firefox.desktop
...
xterm.desktop
yelp.desktop
```

3. Create the autostart directory:

```
[user@example-app qube ~]$ mkdir -p ~/.config/autostart
```

4. Make a link to the desktop app file you'd like to start in the autostart directory. For example, the command below will link the Thunderbird app into the autostart directory:

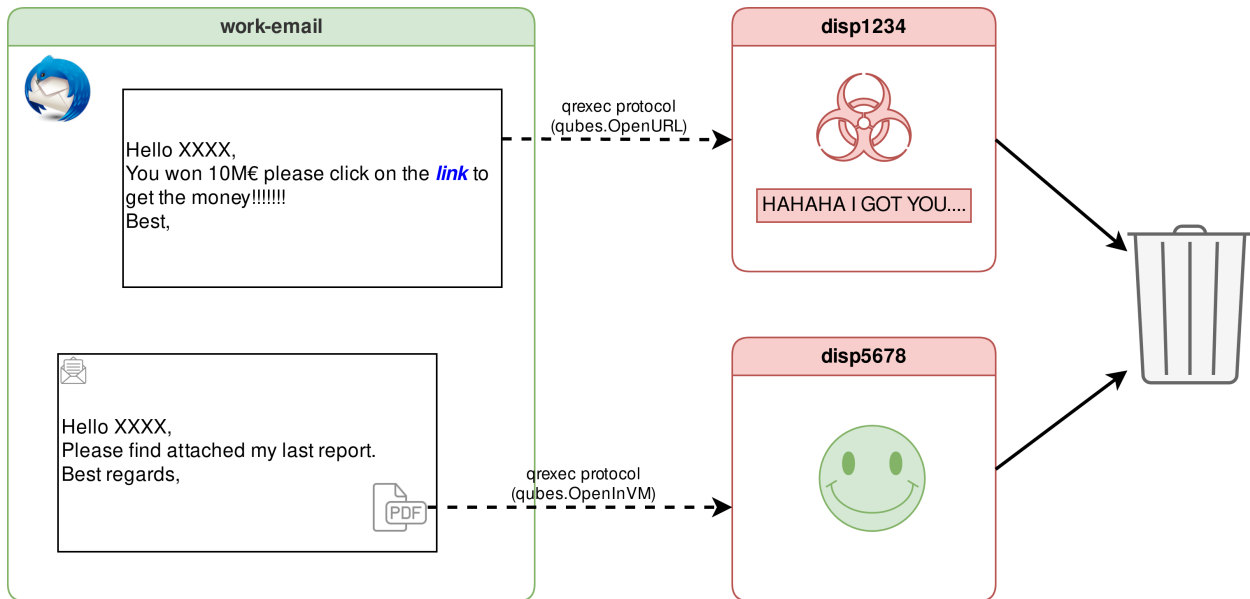
```
[user@example-app qube ~]$ ln -s /usr/share/applications/mozilla-thunderbird.  
→desktop ~/.config/autostart/mozilla-thunderbird.desktop
```

Note that the app will autostart only when the app qube starts. If you would like the app qube to autostart, select the “Start qube automatically on boot” checkbox in the app qube’s Qube Settings.

### 1.12.16 How to use disposables

A *disposable* is a lightweight *qube* that can be created quickly and will self-destruct when closed. Disposables are usually created in order to host a single application, like a viewer, editor, or web browser.

From inside an app qube, choosing the **Open in disposable** option on a file will launch a disposable for just that file. Changes made to a file opened in a disposable are passed back to the originating qube. This means that you can safely work with untrusted files without risk of compromising your other qubes. Disposables can be launched either directly from dom0’s app menu or terminal window, or from within app qubes. Disposables are generated with names like `disp####`, where `####` is random number.



This diagram provides a general example of how disposables can be used to safely open untrusted links and attachments in disposables. See [this article](#) for more on why one would want to use a disposable.

### Security

If a *disposable template* becomes compromised, then any disposable based on that disposable template could be compromised. In particular, the *default* disposable template is important because it is used by the “Open in disposable” feature. This means that it will have access to everything that you open with this feature. For this reason, it is strongly recommended that you base the default disposable template on a trusted template.

## Disposables and Local Forensics

At this time, disposables should not be relied upon to circumvent local forensics, as they do not run entirely in RAM. For details, see [this thread](#).

When it is essential to avoid leaving any trace, consider using [Tails](#).

## Disposables and Networking

Similarly to how app qubes are based on their underlying *template*, disposables are based on their underlying *disposable template*. R4.0 introduces the concept of multiple disposable templates, whereas R3.2 was limited to only one.

On a fresh installation of Qubes, the default disposable template is called `fedora-X-dvm` or `debian-X-dvm` (where `X` is a release number). If you have included the Whonix option in your install, there will also be a `whonix-ws-dvm` disposable template available for your use.

You can set any app qube to have the ability to act as a disposable template with:

```
qvm-prefs <APP_QUBE> template_for_dispvms True
```

The default system wide disposable template can be changed with `qubes-prefs default_dispvm`. By combining the two, choosing `Open in disposable` from inside an app qube will open the document in a disposable based on the default disposable template you specified.

You can change this behavior for individual qubes: in the Application Menu, open Qube Settings for the qube in question and go to the “Advanced” tab. Here you can edit the “Default disposable” setting to specify which disposable template will be used to launch disposables from that qube. This can also be changed from the command line with:

```
qvm-prefs <QUBE> default_dispvm <DISPOSABLE_TEMPLATE>
```

For example, `anon-whonix` has been set to use `whonix-ws-dvm` as its `default_dispvm`, instead of the system default. You can even set an app qube that has also been configured as a disposable template to use itself, so disposables launched from within the app qube/disposable template would inherit the same settings.

Network and firewall settings for disposable templates can be set as they can for a normal qube. By default a disposable will inherit the network and firewall settings of the disposable template on which it is based. This is a change in behavior from R3.2, where disposables would inherit the settings of the app qube from which they were launched. Therefore, launching a disposable from an app qube will result in it using the network/firewall settings of the disposable template on which it is based. For example, if an app qube uses `sys-net` as its net qube, but the default system disposable uses `sys-whonix`, any disposable launched from this app qube will have `sys-whonix` as its net qube.

**Warning:** The opposite is also true. This means if you have changed `anon-whonix`’s `default_dispvm` to use the system default, and the system default disposable uses `sys-net`, launching a disposable from inside `anon-whonix` will result in the disposable using `sys-net`.

A disposable launched from the app menu inherits the net qube and firewall settings of the disposable template on which it is based. Note that changing the net qube setting for the system default disposable template *does* affect the net qube of disposables launched from the app menu. Different disposable templates with individual net qube settings can be added to the app menu.

**Important Notes:** Some disposable templates will automatically create a menu item to launch a disposable. If you do not see an entry and want to add one, please use the command:

```
qvm-features <DISPOSABLE_TEMPLATE> appmenus-dispvm 1
```

To launch a disposable template from the command line, execute the following command in `dom0`:

```
qvm-run --dispvml=<DISPOSABLE_TEMPLATE> --service qubes.StartApp+<APPLICATION>
```

## Opening a file in a disposable via GUI

In an app qube's file manager, right click on the file you wish to open in a disposable, then choose "View in disposable" or "Edit in disposable". Wait a few seconds and the default application for this file type should appear displaying the file content. This app is running in its own dedicated qube – a disposable created for the purpose of viewing or editing this very file. Once you close the viewing application the whole disposable will be destroyed. If you have edited the file and saved the changes, the changed file will be saved back to the original app qube, overwriting the original.

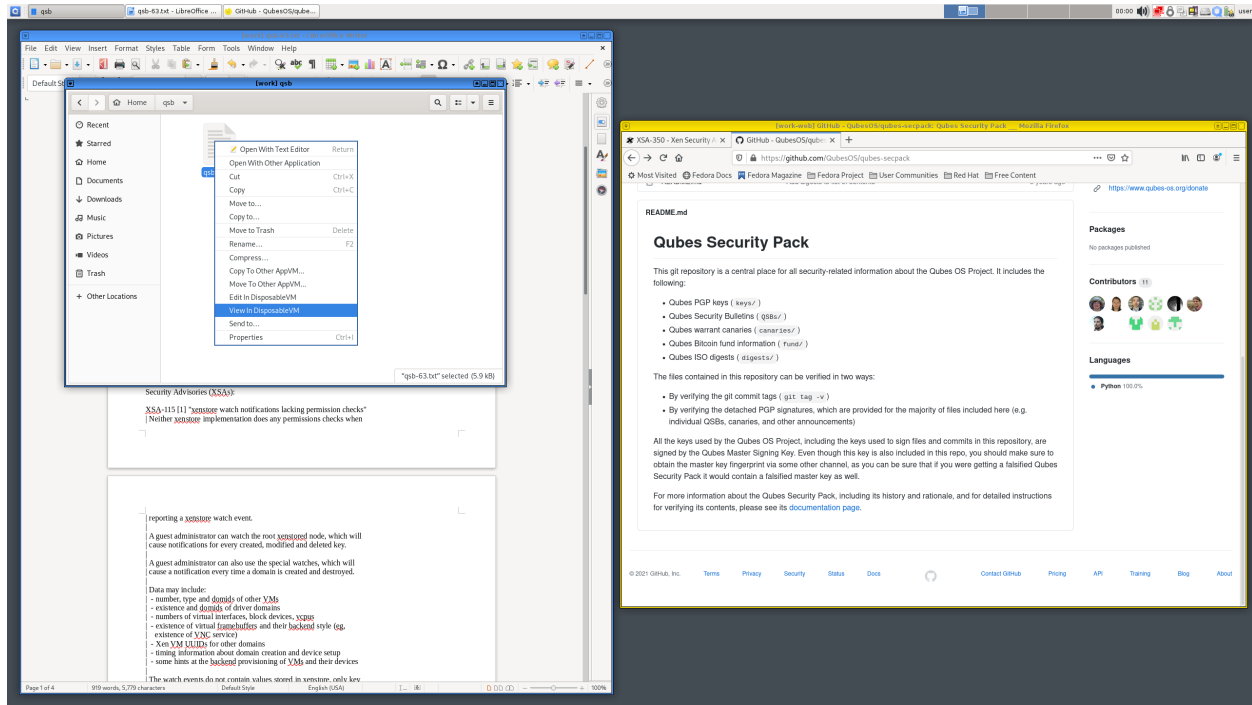


Fig. 3: r4.0-open-in-dispvm-1.png

## Opening a fresh web browser instance in a new disposable

Sometimes it is desirable to open an instance of Firefox within a new fresh disposable. This can be done easily using the app menu: just go to **Application Menu -> Disposable -> Disposable: Firefox web browser**. Wait a few seconds until a web browser starts. Once you close the viewing application the whole disposable will be destroyed.

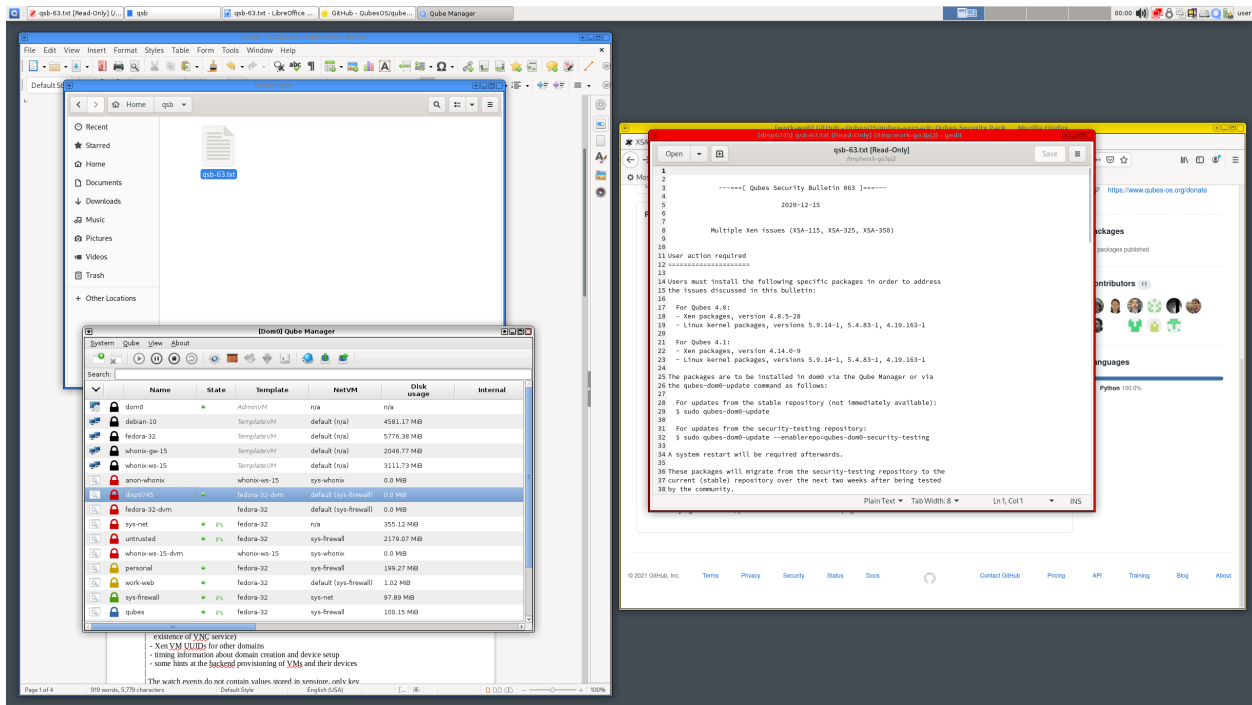


Fig. 4: r4.0-open-in-dispvm-2.png



Fig. 5: r4.0-open-in-dispvm-3.png

### Opening a file in a disposable via command line (from app qube)

Use the `qvm-open-in-dvm` command from a terminal in your app qube:

```
[user@work-pub ~]$ qvm-open-in-dvm Downloads/apple-sandbox.pdf
```

Note that the `qvm-open-in-dvm` process will not exit until you close the application in the disposable.

### Making a particular application open everything in a disposable

You can use the `qvm-service` command or the services GUI to cause an application in a qube to open files and URLs in a disposable. To do this, enable a service named `app-dispvm.X` in that qube, where `X` is the application ID. For instance, to have Thunderbird open all attachments in a disposable, enable the `app-dispvm.thunderbird` service.

This feature is currently somewhat experimental, and only works for Linux qubes. It is known to work with Thunderbird and Wire, but it may fail to work with some applications that do not honor all XDG environment variables. If the feature does not work for you, please file a bug report.

### Opening particular types of files in a disposable

You can set `qvm-open-in-dvm.desktop` as the handler for a given MIME type. This will cause all files of that type to open in a disposable. This works in disposable templates too, but be careful: if your disposable template is set to use `qvm-open-in-dvm.desktop` to open a certain kind of file, every disposable based on it will be as well. If the disposable template is its own default disposable template (as is often the case), this will result in a loop: `qvm-open-in-dvm` will execute `qubes.OpenURL` in a new disposable, but that will in turn execute `qvm-open-in-dvm`. The cycle will repeat until no new disposables can be created, most likely because your system has run out of memory.

This will *not* override the internal handling of PDF documents in Web browsers. This is typically okay, though: in-browser PDF viewers have a fairly good security record, especially when compared to non-browser PDF viewers. In particular, the attack surface of PDF viewing in Firefox is usually less than that of viewing an ordinary Web page.

### Starting an arbitrary program in a disposable from an app qube

Sometimes it can be useful to start an arbitrary program in a disposable. The disposable will stay running so long as the process which started the disposable has not exited. Some applications, such as GNOME Terminal, do not wait for the application to close before the process exits (details [here](#)). Starting an arbitrary program can be done from an app qube by running

```
[user@vault ~]$ qvm-run '@dispvm' xterm
```

The created disposable can be accessed via other tools (such as `qvm-copy-to-vm`) using its `disp####` name as shown in the Qubes Manager or `qvm-ls`.

## Starting an arbitrary application in a disposable via command line from dom0

The Application Launcher has shortcuts for opening a terminal and a web browser in dedicated disposables, since these are very common tasks. The disposable will stay running so long as the process which started the disposable has not exited. Some applications, such as GNOME Terminal, do not wait for the application to close before the process exits (details [here](#)). It is possible to start an arbitrary application in a disposable directly from dom0 by running:

```
$ qvm-run --dispvml=<DISPOSABLE_TEMPLATE> --service qubes.StartApp+xterm
```

The label color will be inherited from <DISPOSABLE\_TEMPLATE>. (The disposable Application Launcher shortcut used for starting programs runs a very similar command to the one above.)

## Opening a link in a disposable based on a non-default disposable template from a qube

Suppose that the default disposable template for your email qube has no networking (e.g., so that untrusted attachments can't phone home). However, sometimes you want to open email links in disposables. Obviously, you can't use the default disposable template, since it has no networking, so you need to be able to specify a different disposable template. You can do that with this command from the email qube (as long as your RPC policies allow it):

```
$ qvm-open-in-vm @dispvml:<ONLINE_DISPOSABLE_TEMPLATE> https://www.qubes-os.org
```

This will create a new disposable based on <ONLINE\_DISPOSABLE\_TEMPLATE>, open the default web browser in that disposable, and navigate to <https://www.qubes-os.org>.

## Example of RPC policies to allow this behavior

In dom0, add the following line at the beginning of the file `/etc/qubes-rpc/policy/qubes.OpenURL`

```
@anyvm @dispvml:<ONLINE_DISPOSABLE_TEMPLATE> allow
```

This line means: - FROM: Any qube - TO: A disposable based on <ONLINE\_DISPOSABLE\_TEMPLATE> - WHAT: Allow sending an "Open URL" request

In other words, any qube will be allowed to create a new disposable based on <ONLINE\_DISPOSABLE\_TEMPLATE> and open a URL inside of that disposable.

More information about RPC policies for disposables can be found [here](#).

## Customizing disposables

You can change the template used to generate the disposables, and change settings used in the disposable savefile. These changes will be reflected in every new disposable based on that template. Full instructions can be found [here](#).

### 1.12.17 How to enter fullscreen mode

#### What is fullscreen mode?

Normally, the Qubes GUI virtualization daemon restricts the VM from “owning” the full screen, ensuring that there are always clearly marked decorations drawn by the trusted Window Manager around each of the VMs window. This allows the user to easily realize to which domain a specific window belongs. See the [screenshots](#) page for examples.

#### Why is fullscreen mode potentially dangerous?

If one allowed one of the VMs to “own” the full screen, e.g. to show a movie on a full screen, it might not be possible for the user to know if the applications/VM really “released” the full screen, or if it has started emulating the whole desktop and is pretending to be the trusted Window Manager, drawing shapes on the screen that look e.g. like other windows, belonging to other domains (e.g. to trick the user into entering a secret passphrase into a window that looks like belonging to some trusted domain).

#### Secure use of fullscreen mode

However, it is possible to deal with fullscreen mode in a secure way assuming there are mechanisms that can be used at any time to switch between windows or show the full desktop and that cannot be intercepted by the VM. The simplest example is the use of Alt+Tab for switching between windows, which is a shortcut handled by dom0.

Other examples such mechanisms are the KDE “Present Windows” and “Desktop Grid” effects, which are similar to Mac’s “Expose” effect, and which can be used to immediately detect potential “GUI forgery”, as they cannot be intercepted by any of the VM (as the GUID never passes down the key combinations that got consumed by KDE Window Manager), and so the VM cannot emulate those. Those effects are enabled by default in KDE once Compositing gets enabled in KDE (System Settings -> Desktop -> Enable Desktop Effects), which is recommended anyway. By default, they are triggered by Ctrl-F8 and Ctrl-F9 key combinations, but can also be reassigned to other shortcuts.

#### Enabling fullscreen mode for select VMs

You can always put a window into fullscreen mode in Xfce4 using the trusted window manager by right-clicking on a window’s title bar and selecting “Fullscreen” or pressing `alt + f11`. This functionality should still be considered safe, since a VM window still can’t voluntarily enter fullscreen mode. The user must select this option from the trusted window manager in dom0. To exit fullscreen mode from here, press `alt + space` to bring up the title bar menu again, then select “Leave Fullscreen” or simply press `alt + f11`. For StandaloneHVMs, you should set the screen resolution in the qube to that of the host, (or larger), *before* setting fullscreen mode in Xfce4.

As an alternative to the Xfce4 method, you can enable fullscreen mode for select VMs by creating the following entry in the `/etc/qubes/guid.conf` file in dom0:

```
VM: {  
  personal: {  
    allow_fullscreen = true;  
  };  
};
```

The string ‘personal’ above is an example only and should be replaced by the actual name of the VM for which you want to enable this functionality.

**Note:** There should be only one `VM: {}` block in the file (or you will [get into problems](#)).

One can also enable this functionality for all the VMs globally in the same file, by modifying the ‘global’ section:



```
global: {
  # default values
  allow_fullscreen = true;
  #allow_utf8_titles = false;
  #secure_copy_sequence = "Ctrl-Shift-c";
  #secure_paste_sequence = "Ctrl-Shift-v";
  #windows_count_limit = 500;
};
```

Be sure to restart the VM(s) after modifying this file, for the changes to take effect.

### 1.12.18 How to use devices

This is an overview of device handling in Qubes OS. For specific devices (*block*, *USB* and *PCI* devices), please visit their respective pages.

**Important security warning:** Device handling comes with many security implications. Please make sure you carefully read and understand the *security considerations*.

#### Introduction

The interface to deal with devices of all sorts was unified in Qubes 4.0 with the `qvm-device` command and the Qubes Devices Widget. In Qubes 3.X, the Qubes VM Manager dealt with attachment as well. This functionality was moved to the Qubes Device Widget, the tool tray icon with a yellow square located in the top right of your screen by default.

There are currently four categories of devices Qubes understands: - Microphones - Block devices - USB devices - PCI devices

Microphones, block devices and USB devices can be attached with the GUI-tool. PCI devices can be attached using the Qube Settings, but require a VM reboot.

#### General Qubes Device Widget Behavior And Handling




When clicking on the tray icon (which looks similar to this): several device-classes separated by lines are displayed as tooltip. Block devices are displayed on top, microphones one below and USB-devices at the bottom.

On most laptops, integrated hardware such as cameras and fingerprint-readers are implemented as USB-devices and can be found here.

#### Attaching Using The Widget

Click the tray icon. Hover on a device you want to attach to a VM. A list of running VMs (except dom0) appears. Click on one and your device will be attached!

## Detaching Using The Widget

To detach a device, click the Qubes Devices Widget icon again. Attached devices are displayed in bold. Hover the one you want to detach. A list of VMs appears, one showing the eject symbol: 

## Attaching a Device to Several VMs

Only mic should be attached to more than one running VM. You may *assign* a device to more than one VM (using the `--persistent` option), however, only one of them can be started at the same time.

But be careful: There is a [bug in qvm-device block or qvm-block](#) which will allow you to *attach* a block device to two running VMs. Don't do that!

## General qvm-device Command Line Tool Behavior

All devices, including PCI-devices, may be attached from the commandline using the `qvm-device`-tools.

## Device Classes

`qvm-device` expects `DEVICE_CLASS` as first argument. `DEVICE_CLASS` can be one of

- `pci`
- `usb`
- `block`
- `mic`

## Actions

`qvm-device` supports three actions:

- `list` (`ls`, `l`) - list all devices of `DEVICE_CLASS`
- `attach` (`at`, `a`) - attach a specific device of `DEVICE_CLASS`
- `detach` (`dt`, `d`) - detach a specific device of `DEVICE_CLASS`

## Global Options

These three options are always available:

- `--help`, `-h` - show help message and exit
- `--verbose`, `-v` - increase verbosity
- `--quiet`, `-q` - decrease verbosity

A full command consists of one `DEVICE_CLASS` and one action. If no action is given, `list` is implied. `DEVICE_CLASS` however is required.

**SYNOPSIS:** `qvm-device DEVICE_CLASS {action} [action-specific arguments] [options]`

## Actions

Actions are applicable to every `DEVICE_CLASS` and expose some additional options.

### Listing Devices

The `list` action lists known devices in the system. `list` accepts VM-names to narrow down listed devices. Devices available in, as well as attached to the named VMs will be listed.

`list` accepts two options:

- `--all` - equivalent to specifying every VM name after `list`. No VM-name implies `--all`.
- `--exclude` - exclude VMs from `--all`. Requires `--all`.

**SYNOPSIS** `qvm-device DEVICE_CLASS {list|ls|l} [--all [--exclude VM [VM [...]]] | VM [VM [...]]]`

### Attaching Devices

The `attach` action assigns an exposed device to a VM. This makes the device available in the VM it's attached to. Required argument are `targetVM` and `sourceVM:deviceID`. (`sourceVM:deviceID` can be determined from `list` output)

`attach` accepts two options:

- `--persistent` - attach device on `targetVM`-boot. If the device is unavailable (physically missing or `sourceVM` not started), booting the `targetVM` fails.
- `--option, -o` - set additional options specific to `DEVICE_CLASS`.

**SYNOPSIS** `qvm-device DEVICE_CLASS {attach|at|a} targetVM sourceVM:deviceID [options]`

### Detaching Devices

The `detach` action removes an assigned device from a `targetVM`. It won't be available afterwards anymore. Though it tries to do so gracefully, beware that data-connections might be broken unexpectedly, so close any transaction before detaching a device!

If no specific `sourceVM:deviceID` combination is given, *all devices of that `DEVICE_CLASS` will be detached*.

`detach` accepts no options.

**SYNOPSIS** `qvm-device DEVICE_CLASS {detach|dt|d} targetVM [sourceVM:deviceID]`

## 1.12.19 How to use block storage devices

*This page is part of [device handling in qubes](#).*

If you don't know what a "block device" is, just think of it as a fancy way to say "something that stores data".

## Using the Devices Widget to Attach a Drive

(**Note:** In the present context, the term “USB drive” denotes any [USB mass storage device](#). In addition to smaller flash memory sticks, this includes things like USB external hard drives.)

Qubes OS supports the ability to attach a USB drive (or just its partitions) to any qube easily, no matter which qube handles the USB controller.



Attaching USB drives is integrated into the Devices Widget: Simply insert your USB drive and click on the widget. You will see multiple entries for your USB drive; typically, `sys-usb:sda`, `sys-usb:sda1`, and `sys-usb:2-1` for example. Entries starting with a number (e.g. here 2-1) are the *whole usb-device*. Entries without a number (e.g. here `sda`) are the whole block-device. Other entries are partitions of that block-device (e.r. here `sda1`).

The simplest option is to attach the entire block drive. In our example, this is `sys-usb:sda`, so hover over it. This will pop up a submenu showing running VMs to which the USB drive can be connected. Click on one and your USB drive will be attached!

**Note:** attaching individual partitions (e.g. `sys-usb:sda1`) can be slightly more secure because it doesn’t force the target app qube to parse the partition table. However, it often means the app qube won’t detect the new partition and you will need to manually mount it inside the app qube. See below for more detailed steps.

## Block Devices in VMs

If not specified otherwise, block devices will show up as `/dev/xvdi*` in a linux VM, where `*` may be the partition-number. If a block device isn’t automatically mounted after attaching, open a terminal in the VM and execute:

```
cd ~
mkdir mnt
sudo mount /dev/xvdi2 mnt
```

where `xvdi2` needs to be replaced with the partition you want to mount. This will make your drive content accessible under `~/mnt`.

Beware that when you attach a whole block device, partitions can be identified by their trailing integer (i.e. `/dev/xvdi2` for the second partition, `/dev/xvdi` for the whole device), whereas if you attach a single partition, the partition has *no trailing integer*.

If several different block-devices are attached to a single VM, the last letter of the device node name is advanced through the alphabet, so after `xvdi` the next device will be named `xvdj`, the next `xvdk`, and so on.

To specify this device node name, you need to use the command line tool and its *frontend-dev-option*.

## Command Line Tool Guide

The command-line tool you may use to mount whole USB drives or their partitions is `qvm-block`, a shortcut for `qvm-device block`.

`qvm-block` won’t recognise your device by any given name, but rather the device-node the sourceVM assigns. So make sure you have the drive available in the sourceVM, then list the available block devices (step 1.) to find the corresponding device-node.

In case of a USB-drive, make sure it’s attached to your computer. If you don’t see anything that looks like your drive, run `sudo udevadm trigger --action=change` in your USB-qube (typically `sys-usb`)

1. In a dom0 console (running as a normal user), list all available block devices:

```
qvm-block
```

This will list all available block devices in your system across all VMs. The name of the qube hosting the block device is displayed before the colon in the device ID. The string after the colon is the ID of the device used within the qube, like so:

```
sourceVM:sdb      Cruzeiro () 4GiB
sourceVM:sdb1     Disk () 2GiB
```

2. Assuming your block device is attached to `sys-usb` and its device node is `sdb`, we attach the device to a qube with the name `work` like so:

```
qvm-block attach work sys-usb:sdb
```

This will attach the device to the qube as `/dev/xvdi` if that name is not already taken by another attached device, or `/dev/xvdj`, etc. You may also mount one partition at a time by using the same command with the partition number, e.g. `sdb1`.

3. The block device is now attached to the qube. If using a default qube, you may open the Nautilus file manager in the qube, and your drive should be visible in the **Devices** panel on the left. If you've attached a single partition (e.g. `sdb2` instead of `sdb` in our example), you may need to manually mount before it becomes visible:

```
cd ~
mkdir mnt
sudo mount /dev/xvdi mnt
```

4. When you finish using the block device, click the eject button or right-click and select **Unmount**. If you've manually mounted a single partition in the above step, use:

```
sudo umount mnt
```

5. In a dom0 console, detach the device

```
qvm-block detach work sys-usb:sdb
```

6. You may now remove the device or attach it to another qube.

## Recovering From Premature Device Destruction

If the you fail to detach the device before it's destroyed in the sourceVM (e.g. by physically detaching the thumbdrive), [there will be problems](#).

To recover from this error state, in dom0 run

```
virsh detach-disk targetVM xvdi
```

(where `targetVM` is to be replaced with the VM name you attached the device to and `xvdi` is to be replaced with the used *frontend device node*.)

However, if the block device originated in dom0, you will have to refer to the next section.

## What if I removed the device before detaching it from the VM?

Currently (until issue [1082](#) gets implemented), if you remove the device before detaching it from the qube, Qubes OS (more precisely, `libvirtd`) will think that the device is still attached to the qube and will not allow attaching further devices under the same name. The easiest way to recover from such a situation is to reboot the qube to which the device was attached. If this isn't an option, you can manually recover from the situation by following these steps:

1. Physically connect the device back. You can use any device as long as it will be detected under the same name (for example, `sdb`).
2. Attach the device manually to the same VM using the `xl block-attach` command. It is important to use the same “frontend” device name (by default, `xvdi`). You can get it from the `qvm-block` listing:

```
[user@dom0 ~]$ qvm-block
sys-usb:sda DataTraveler_2.0 () 246 MiB (attached to 'testvm' as 'xvdi')
[user@dom0 ~]$ sudo xl block-attach testvm phy:/dev/sda backend=sys-usb xvdi
```

In above example, all `xl block-attach` parameters can be deduced from the output of `qvm-block`. In order:

- `testvm` - name of target qube to which device was attached - listed in brackets by `qvm-block` command
  - `phy:/dev/sda` - physical path at which device appears in source qube (just after source qube name in `qvm-block` output)
  - `backend=sys-usb` - name of source qube, can be omitted in the case of `dom0`
  - `xvdi` - “frontend” device name (listed at the end of line in `qvm-block` output)
3. Now properly detach the device, either using Qubes VM Manager or the `qvm-block -d` command.


## Attaching a File

To attach a file as block device to another qube, first turn it into a loopback device inside the sourceVM.

1. In the linux sourceVM run

```
sudo losetup -f --show /path/to/file
```

This command will create the device node `/dev/loop0` or, if that is already in use, increase the trailing integer until that name is still available. Afterwards it prints the device-node-name it found.

2. If you want to use the GUI, you're done. Click the Device Manager  and select the `loop0`-device to attach it to another qube. If you rather use the command line, continue: In `dom0`, run `qvm-block` to display known block devices. The newly created loop device should show up:

```
~]$ qvm-block
BACKEND:DEVID DESCRIPTION USED BY
sourceVM:loop0 /path/to/file
```

3. Attach the `loop0`-device using `qvm-block` as usual:

```
qvm-block a targetVM sourceVM:loop0
```

4. After detaching, destroy the loop-device inside the sourceVM as follows:

```
sudo losetup -d /dev/loop0
```

## Additional Attach Options

Attaching a block device through the command line offers additional customisation options, specifiable via the `--option/-o` option. (Yes, confusing wording, there's an [issue for that](#).)

### frontend-dev

This option allows you to specify the name of the device node made available in the targetVM. This defaults to `xvdi` or, if already occupied, the first available device node name in alphabetical order. (The next one tried will be `xvdj`, then `xvdk`, and so on ...)

usage example:

```
qvm-block a work sys-usb:sda1 -o frontend-dev=xvdz
```

This command will attach the partition `sda1` to `work` as `/dev/xvdz`.

### read-only

Attach device in read-only mode. Protects the block device in case you don't trust the targetVM.

If the device is a read-only device, this option is forced true.

usage example:

```
qvm-block a work sys-usb:sda1 -o read-only=true
```

There exists a shortcut to set read-only true, `--ro`:

```
qvm-block a work sys-usb:sda1 --ro
```

The two commands are equivalent.

### devtype

Usually, a block device is attached as disk. In case you need to attach a block device as cdrom, this option allows that.

usage example:

```
qvm-block a work sys-usb:sda1 -o devtype=cdrom
```

This option accepts `cdrom` and `disk`, default is `disk`.

## 1.12.20 How to use USB devices

*This page is part of [device handling in qubes](#).*

If you are looking to handle USB *storage* devices (thumbdrives or USB-drives), please have a look at the [block device](#) page.

**Note:** Attaching USB devices to VMs requires a [USB qube](#).

**Important security warning:** USB passthrough comes with many security implications. Please make sure you carefully read and understand the [security considerations](#). Whenever possible, attach a [block device](#) instead.



Examples of valid cases for USB-passthrough:

- microcontroller programming
- external audio devices
- *optical drives* for recording

(If you are thinking to use a two-factor-authentication device, *there is an app for that*. But it has some *issues*.)

## Attaching And Detaching a USB Device


### With Qubes Device Manager

Click the device-manager-icon:  A list of available devices appears. USB-devices have a USB-icon to their right: 

Hover on one device to display a list of VMs you may attach it to.

Click one of those. The USB device will be attached to it. You're done.

After you finished using the USB-device, you can detach it the same way by clicking on the Devices Widget. You will see an entry in bold for your device such as **sys-usb:2-5 - 058f\_USB\_2.0\_Camera**. Hover on the attached device

to display a list of running VMs. The one to which your device is connected will have an eject button  next to it. Click that and your device will be detached.

### With The Command Line Tool

In dom0, you can use `qvm-usb` from the commandline to attach and detach devices.

Listing available USB devices:

```
[user@dom0 ~]$ qvm-usb
BACKEND:DEVID  DESCRIPTION                      USED BY
sys-usb:2-4    04ca:300d 04ca_300d
sys-usb:2-5    058f:3822 058f_USB_2.0_Camera
sys-usb:2-1    03f0:0641 PixArt_HP_X1200_USB_Optical_Mouse
```

Attaching selected USB device:

```
[user@dom0 ~]$ qvm-usb attach work sys-usb:2-5
[user@dom0 ~]$ qvm-usb
BACKEND:DEVID  DESCRIPTION                      USED BY
sys-usb:2-4    04ca:300d 04ca_300d
sys-usb:2-5    058f:3822 058f_USB_2.0_Camera  work
sys-usb:2-1    03f0:0641 PixArt_Optical_Mouse
```

Now, you can use your USB device (camera in this case) in the work qube. If you see the error `ERROR: qubes-usb-proxy not installed in the VM` instead, please refer to the [Installation Section](#).

When you finish, detach the device.



```
[user@dom0 ~]$ qvm-usb detach work sys-usb:2-5
[user@dom0 ~]$ qvm-usb
BACKEND:DEVID      DESCRIPTION                                USED BY
sys-usb:2-4        04ca:300d 04ca_300d
sys-usb:2-5        058f:3822 058f_USB_2.0_Camera
sys-usb:2-1        03f0:0641 PixArt_Optical_Mouse
```

## Maintenance And Customisation

### Creating And Using a USB qube

If you've selected to install a usb-qube during system installation, everything is already set up for you in sys-usb. If you've later decided to create a usb-qube, please follow [this guide](#).

### Installation Of qubes-usb-proxy

To use this feature, the qubes-usb-proxy package needs to be installed in the templates used for the USB qube and qubes you want to connect USB devices to. This section exists for reference or in case something broke and you need to reinstall qubes-usb-proxy. Under normal conditions, qubes-usb-proxy should already be installed and good to go.

If you receive this error: `ERROR: qubes-usb-proxy not installed in the VM`, you can install the qubes-usb-proxy with the package manager in the VM you want to attach the USB device to.

- Fedora:

```
sudo dnf install qubes-usb-proxy
```

- Debian/Ubuntu:

```
sudo apt-get install qubes-usb-proxy
```

## Using USB Keyboards And Other Input Devices

**Warning:** especially keyboards need to be accepted by default when using them to login! Please make sure you carefully read and understood the [security considerations](#) before continuing!

Mouse and keyboard setup are part of [setting up a USB qube](#).

### Finding The Right USB Controller

Some USB devices are not compatible with the USB pass-through method Qubes employs. In situations like these, you can try to pass through the entire USB controller to a qube as PCI device. However, with this approach one cannot attach single USB devices but has to attach the whole USB controller with whatever USB devices are connected to it.

If you have multiple USB controllers, you must first figure out which PCI device is the right controller.

First, find out which USB bus the device is connected to (note that these steps need to be run from a terminal inside your USB qube):

```
lsusb
```

For example, I want to attach a broadband modem to the NetVM. In the output of `lsusb` it may be listed as something like:

```
Bus 003 Device 003: ID 413c:818d Dell Computer Corp.
```

(In this case, the device isn't fully identified)

The device is connected to USB bus #3. Check which other devices are connected to the same bus, since *all* of them will be attach to the same VM.

To find the right controller, follow the usb bus:

```
readlink /sys/bus/usb/devices/usb3
```

This should output something like:

```
../../../../devices/pci-0/pci0000:00/0000:00:1a.0/usb3
```

Now you see the path and the text between `/pci0000:00/0000:` and `/usb3` i.e. `00:1a.0` is the BDF address. Strip the address and pass it to the *qvm-pci tool* to attach the controller to the targetVM.

For example, On R 4.0 the command would look something like

```
qvm-pci attach --persistent personal dom0:00_1a.0
```

### 1.12.21 How to use PCI devices

*This page is part of [device handling in qubes](#) .*

**Warning:** Only dom0 exposes PCI devices. Some of them are strictly required in dom0 (e.g., the host bridge). You may end up with an unusable system by attaching the wrong PCI device to a VM. PCI passthrough should be safe by default, but non-default options may be required. Please make sure you carefully read and understand the *[security considerations](#)* before deviating from default behavior.

#### Introduction

Unlike other devices (*USB*, *block*, mic), PCI devices need to be attached on VM-bootup. Similar to how you can't attach a new sound-card after your computer booted (and expect it to work properly), attaching PCI devices to already booted VMs isn't supported. Moreover, PCI devices can be attached only to VMs running in certain virtualization modes. See *[FAQ: Which virtualization modes do VMs use?](#)*

The Qubes installer attaches all network class controllers to `sys-net` and all USB controllers to `sys-usb` by default, if you chose to create the network and USB qube during install. While this covers most use cases, there are some occasions when you may want to manually attach one NIC to `sys-net` and another to a custom NetVM, or have some other type of PCI controller you want to manually attach.

Some devices expose multiple functions with distinct BDF-numbers. Limits imposed by the PC and VT-d architectures may require all functions belonging to the same device to be attached to the same VM. This requirement can be dropped with the `no-strict-reset` option during attachment, bearing in mind the aforementioned *[security considerations](#)*. In the steps below, you can tell if this is needed if you see the BDF for the same device listed multiple times with only the number after the “.” changing.

While PCI device can only be used by one powered on VM at a time, it *is* possible to *assign* the same device to more than one VM at a time. This means that you can use the device in one VM, shut that VM down, start up a different VM

(to which the same device is now attached), then use the device in that VM. This can be useful if, for example, you have only one USB controller, but you have multiple security domains which all require the use of different USB devices.

## Attaching Devices Using the GUI

The qube settings for a VM offers the “Devices”-tab. There you can attach PCI-devices to a qube.

1. To reach the settings of any qube either
  - Press Alt+F3 to open the application finder, type in the VM name, select the “[appmenu][VM-name]: Qube Settings” menu entry and press enter or click “Launch”!
  - Select the VM in Qube Manager and click the settings-button or right-click the VM and select Qube settings.
  - Click the Domain Manager, hover the VM you want to attach a device to and select “settings” in the additional menu. (only running VMs!)
2. Select the “Devices” tab on the top bar.
3. Select a device you want to attach to the qube and click the single arrow right! (>)
4. You’re done. If everything worked out, once the qube boots (or reboots if it’s running) it will start with the pci device attached.
5. In case it doesn’t work out, first try disabling memory-balancing in the settings (“Advanced” tab). If that doesn’t help, read on to learn how to disable the strict reset requirement!

## qvm-pci Usage

The `qvm-pci` tool allows PCI attachment and detachment. It’s a shortcut for `qvm-device pci`

To figure out what device to attach, first list the available PCI devices by running (as user) in dom0:

```
qvm-pci
```

This will show you the `backend:BDF` (Bus\_Device.Function) address of each PCI device. It will look something like `dom0:00_1a.0`. Once you’ve found the address of the device you want to attach, then attach it like this:

```
qvm-pci attach targetVM sourceVM:[BDF] --persistent
```

Since PCI devices have to be attached on bootup, attaching has to happen with the `--persistent` option.

For example, if `00_1a.0` is the BDF of the device you want to attach to the “work” domain, you would do this:

```
qvm-pci attach work dom0:00_1a.0 --persistent
```

## Possible Issues

Visit the [PCI Troubleshooting guide](#) to see issues that may arise due to PCI devices and how to troubleshoot them.

### Additional Attach Options

Attaching a PCI device through the commandline offers additional options, specifiable via the `--option/-o` option. (Yes, confusing wording, there's an [issue for that](#).)

`qvm-pci` exposes two additional options. Both are intended to fix device or driver specific issues, but both come with *heavy security implications*! **Make sure you understand them before continuing!**

#### no-strict-reset

Do not require PCI device to be reset before attaching it to another VM. This may leak usage data even without malicious intent!

usage example:

```
qvm-pci a work dom0:00_1a.0 --persistent -o no-strict-reset=true
```

#### permissive

Allow write access to full PCI config space instead of whitelisted registers. This increases attack surface and possibility of *side channel attacks*.

usage example:

```
qvm-pci a work dom0:00_1a.0 --persistent -o permissive=true
```

### Bringing PCI Devices Back to dom0

By default, when a device is detached from a VM (or when a VM with an attached PCI device is shut down), the device is *not* automatically attached back to dom0.

This is an intended feature.

A device which was previously attached to a VM less trusted than dom0 (which, in Qubes, is *all* of them) could attack dom0 if it were automatically reattached there.

In order to re-enable the device in dom0, either:

- Reboot the physical machine. (Best practice)

or

- Go to the sysfs (`/sys/bus/pci`), find the right device, detach it from the pciback driver, and attach it back to the original driver. Replace `<BDF>` with your full device, for example `0000:00:1c.2`:

```
echo <BDF> > /sys/bus/pci/drivers/pciback/unbind
MODALIAS=`cat /sys/bus/pci/devices/<BDF>/modalias`
MOD=`modprobe -R $MODALIAS | head -n 1`
echo <BDF> > /sys/bus/pci/drivers/$MOD/bind
```

It is **strongly discouraged to reattach PCI devices to dom0**, especially if they don't support resetting!

### 1.12.22 How to use optical discs

Passthrough reading and recording (a.k.a., “burning”) are not supported by Qubes OS. This is not a limitation of Xen, which provides scsiback and scsifront drivers, but of Qubes OS. It will be fixed in the future.

Currently, the only options for reading and recording optical discs (e.g., CDs, DVDs, BRDs) in Qubes are:

1. Use a USB optical drive.
2. Attach a SATA optical drive to a secondary SATA controller, then assign this secondary SATA controller to a VM.
3. Use a SATA optical drive attached to dom0. (**Caution:** This option is *potentially dangerous*.)

To access an optical disc via USB follow the *typical procedure for attaching a USB device*, then check with the **Qubes Devices** widget to see what device in the target qube the USB optical drive was attached to. Typically this would be `sr0`. For example, if `sys-usb` has device 3-2 attached to the `work` qube’s `sr0`, you would mount it with `mount /dev/sr0 /mnt/removable`. You could also write to a disc with `wodim -v dev=/dev/sr0 -eject /home/user/Qubes.iso`.

### 1.12.23 How to reinstall a template

If you suspect your *template* is broken, misconfigured, or compromised, you can reinstall any template that was installed from the Qubes repository.

#### Automatic Method

First, copy any files that you wish to keep from the template’s `/home` and `/rw` folders to a safe storage location. Then, in a `dom0` terminal, run:

```
$ sudo qubes-dom0-update --action=reinstall qubes-template-package-name
```

Replace `qubes-template-package-name` with the name of the *package* of the template you wish to reinstall. For example, use `qubes-template-fedora-25` if you wish to reinstall the `fedora-25` template. Only one template can be reinstalled at a time.

Note that Qubes may initially refuse to perform the reinstall if the exact revision of the template package on your system is no longer in the Qubes online repository. In this case, you can specify `upgrade` as the action instead and the newer version will be used. The other `dnf` package actions that are supported in addition to `reinstall` and `upgrade` are `upgrade-to` and `downgrade`. Note that the `upgrade`, `upgrade-to`, and `downgrade` commands are only supported under Fedora based UpdateVMs. If you receive a message about them being unsupported, review the manual reinstallation method below.

**Reminder:** If you’re trying to reinstall a template that is not in an enabled repo, you must enable that repo. For example:

```
$ sudo qubes-dom0-update --enablerepo=qubes-templates-community --action=reinstall qubes-
↪template-whonix-ws
```

**Note:** VMs that are using the reinstalled template will not be affected until they are restarted.

## Manual Method

In what follows, the term “target template” refers to whichever template you want to reinstall. If you want to reinstall more than one template, repeat these instructions for each one.

1. Clone the existing target template. This can be a good idea if you’ve customized the existing template and want to keep your customizations. On the other hand, if you suspect that this template is broken, misconfigured, or compromised, be certain you do not start any VMs using it in the below procedure.
2. Temporarily change all VMs based on the target template to the new clone template, or remove them. This can be a good idea if you have user data in these VMs that you want to keep. On the other hand, if you suspect that these VMs (or the templates on which they are based) are broken, misconfigured, or compromised, you may want to remove them instead. You can do this in Qubes Manager by right-clicking on the VM and clicking **Remove VM**, or you can use the command `qvm-remove <vm-name>` in dom0.
3. Uninstall the target template from dom0:

```
$ sudo dnf remove <template-package-name>
```

For example, to uninstall the `whonix-gw` template:

```
$ sudo dnf remove qubes-template-whonix-gw
```

4. Reinstall the target template in dom0:

```
$ sudo qubes-dom0-update --enablerepo=<optional-additional-repo> \
  <template-package-name>
```

For example, to install the `whonix-gw` template:

```
$ sudo qubes-dom0-update --enablerepo=qubes-templates-community \
  qubes-template-whonix-gw
```

5. If you temporarily changed all VMs based on the target template to the clone template in step 3, change them back to the new target template now. If you instead removed all VMs based on the old target template, you can recreate your desired VMs from the newly reinstalled target template now.
6. Delete the cloned template. You can do this in Qubes Manager by right-clicking on the VM and clicking **Remove VM**, or you can use the command `qvm-remove <vm-name>` in dom0.

### 1.12.24 Templates

In *Getting Started*, we covered the distinction in Qubes OS between where you *install* your software and where you *run* your software. Your software is installed in *templates*. Each template shares its root filesystem (i.e., all of its programs and system files) with all the qubes based on it. *App qubes* are where you run your software and store your data.

The template system has significant benefits:

- **Security:** Each qube has read-only access to the template on which it’s based, so if a qube is compromised, it cannot infect its template or any of the other qubes based on that template.
- **Storage:** Each qube based on a template uses only the disk space required to store its own data (i.e., your files in its home directory), which dramatically saves on disk space.
- **Speed:** It is extremely fast to create new app qubes, since the root filesystem already exists in the template.
- **Updates:** Updates are naturally centralized, since updating a template means that all qubes based on it will automatically use those updates after they’re restarted.

An important side effect of this system is that any software installed in an app qube (rather than in the template on which it is based) will disappear after the app qube reboots (see *Inheritance and Persistence*). For this reason, we recommend installing most of your software in templates, not app qubes.

The default template in Qubes is based on Fedora, but there are additional templates based on other Linux distributions. There are also templates available with or without certain software preinstalled. You may find it useful to have multiple templates installed in order to provide:

- Different security levels (e.g., more or less trusted software installed)
- Different environments (e.g., Fedora, Debian, Whonix)
- Different tools (e.g., office, media, development, hardware drivers)

## Official

These are the official Qubes OS Project templates. We build and release updates for these templates. We guarantee that the binary updates are compiled from exactly the same source code as we publish.

- *Fedora* (default)
- *Fedora Minimal*
- *Fedora Xfce*
- *Debian*
- *Debian Minimal*

## Community

These templates are supported by the Qubes community. Some of them are available in ready-to-use binary package form (built by the Qubes developers), while others are available only in source code form. In all cases, the Qubes OS Project does not provide updates for these templates. However, such updates may be provided by the template maintainer.

By installing these templates, you are trusting not only the Qubes developers and the distribution maintainers, but also the template maintainer. In addition, these templates may be somewhat less stable, since the Qubes developers do not test them.

- *Whonix*
- *Ubuntu*
- *Arch Linux*
- *CentOS*
- *CentOS Minimal*
- *Gentoo*
- *Gentoo Minimal*

## Windows

Windows templates are constructed differently from Linux-based templates as Windows is a closed source system that can be modified only after installing. So it is not possible to provide preconfigured Windows templates for Qubes. The process of installing a Windows qube and connecting it to the Qubes environment via installing Qubes Windows Tools (QWT) is described in several chapters in Windows qubes.

## Installing

Certain templates come preinstalled with Qubes OS. However, there may be times when you wish to install a fresh template from the Qubes repositories, e.g.:

- When a template version you're using reaches *end-of-life*.
- When a new version of a template that you wish to use becomes *supported*.
- When you suspect your template has been compromised.
- When you have made modifications to your template that you no longer want.

You can use a command line tool - `qvm-template` - or a GUI - `qvm-template-gui`.

At the command line in dom0, `qvm-template list --available` will show available templates. To install a template, use:

```
$ qvm-template install <template_name>
```

You can also use `qvm-template` to upgrade or reinstall templates. | Repo definitions are stored in `/etc/qubes/repo-templates` and associated keys in `/etc/qubes/repo-templates/keys`. | There are additional repos for testing releases and community | templates. To temporarily enable any of these repos, use the `| --enablerepo=<repo-name>` option. E.g. :

```
$ qvm-template --enablerepo qubes-templates-community install <template_name>
```

To permanently enable a repo, set the line `enabled = 1` in the repo definition in `/etc/qubes/repo-templates`. To permanently disable, set the line to `enabled = 0`.

If you wish to install a template that is in testing, please see [here](#).

## After Installing

After installing a fresh template, we recommend performing the following steps:

1. *Update the template.*
2. *Switch any app qubes that are based on the old template to the new one.*
3. If desired, *uninstall the old template.*



## Network access

For information about how templates access the network, please see [Why don't templates have network access?](#) and the [Updates proxy](#).

## Updating

Please see [How to Update](#).

## Installing Software

Please see [How to Install Software](#).

## Uninstalling

If you want to remove a template you must make sure that it is not being used. You should check that the template is not being used by any qubes, and also that it is not set as the default template.

The procedure for uninstalling a template depends on how it was created.

If the template was originally created by cloning another template, then you can delete it the same way as you would any other qube. In the Qube Manager, right-click on the template and select **Delete qube**. (If you're not sure, you can safely try this method first to see if it works.)

If, on the other hand, the template came pre-installed or was installed by installing a template package in dom0, per the instructions [above](#), then you must execute the following type of command in dom0 in order to uninstall it:

```
$ sudo dnf remove qubes-template-<DISTRO_NAME>-<RELEASE_NUMBER>
```

`qubes-template-<DISTRO_NAME>-<RELEASE_NUMBER>` is the name of the desired template package.

You may see warning messages like the following:

```
warning: file /var/lib/qubes/vm-templates/fedora-XX/whitelisted-appmenus.list: remove_
↳ failed: No such file or directory
warning: file /var/lib/qubes/vm-templates/fedora-XX/vm-whitelisted-appmenus.list: remove_
↳ failed: No such file or directory
warning: file /var/lib/qubes/vm-templates/fedora-XX/root.img.part.04: remove failed: No_
↳ such file or directory
warning: file /var/lib/qubes/vm-templates/fedora-XX/root.img.part.03: remove failed: No_
↳ such file or directory
warning: file /var/lib/qubes/vm-templates/fedora-XX/root.img.part.02: remove failed: No_
↳ such file or directory
warning: file /var/lib/qubes/vm-templates/fedora-XX/root.img.part.01: remove failed: No_
↳ such file or directory
warning: file /var/lib/qubes/vm-templates/fedora-XX/root.img.part.00: remove failed: No_
↳ such file or directory
warning: file /var/lib/qubes/vm-templates/fedora-XX/netvm-whitelisted-appmenus.list:_
↳ remove failed: No such file or directory
warning: file /var/lib/qubes/vm-templates/fedora-XX/icon.png: remove failed: No such_
↳ file or directory
warning: file /var/lib/qubes/vm-templates/fedora-XX/clean-volatile.img.tar: remove_
↳ failed: No such file or directory
```

(continues on next page)

(continued from previous page)

```
warning: file /var/lib/qubes/vm-templates/fedora-XX/apps.templates: remove failed: No such file or directory
warning: file /var/lib/qubes/vm-templates/fedora-XX/apps.tempicons: remove failed: No such file or directory
warning: file /var/lib/qubes/vm-templates/fedora-XX/apps: remove failed: No such file or directory
warning: file /var/lib/qubes/vm-templates/fedora-XX: remove failed: No such file or directory
```

These are normal and expected. Nothing is wrong, and no action is required to address these warnings.

If the uninstallation command doesn't work, pay close attention to any error message: it may tell you what qube is using the template, or if the template is default. In other cases, please see [VM Troubleshooting](#).

If the Applications Menu entry doesn't go away after you uninstall a template, execute the following type of command in dom0:

```
$ rm ~/.local/share/applications/<TEMPLATE_NAME>
```

Applications Menu entries for backups of removed qubes can also be found in `/usr/local/share/applications/` of dom0.

```
$ rm /usr/local/share/applications/<TEMPLATE_NAME>
```

## Reinstalling

Please see [How to Reinstall a Template](#).

## Switching

When you install a new template or *upgrade* a template, it is recommended that you switch everything that was using the old template to the new template:

1. **Make the new template the default template.** In the App Menu, go to Qubes Tools, then click on Qubes Global Settings. In the Qube Defaults section, next to Template, select the new template from the drop-down list. Press OK.
2. **Base your *disposable templates* on the new template.**
  - If your only keyboard and mouse are *not* connected through a *USB qube*, or that USB qube is *not* a disposable, then shut down all disposables. In the App Menu, go to Qubes Tools, then click on Qube Manager. In the Qube Manager, find your disposable template(s). (By default, they end in `-dvm`.) Right click, hover over Template, then click on the new template. Repeat for each disposable template.
  - If your only keyboard or mouse *are* connected through a USB qube, and that USB qube *is* a disposable, then you will have to enter a special command that shuts down all of your qubes, switches the USB qube's disposable template to the new template, then starts the USB qube again. In order to avoid being locked out of your system, you must be very careful to enter this command without typos and with the correct substitutions. In the App Menu, click on Terminal Emulator. Type the command below, substituting `<SYS_USB_DISPOSABLE_TEMPLATE>` with the name of the disposable template on which `sys-usb` is based, `<NEW_TEMPLATE>` with the name of the new template, and `<USB_QUBE>` with the name of your USB qube. Other than these substitutions, make sure to enter the command exactly as written.

```
qvm-shutdown --wait --all; qvm-prefs <SYS_USB_DISPOSABLE_TEMPLATE> template
↪<NEW_TEMPLATE>; qvm-start <USB_QUBE>
```

With substitutions, your command should look similar to this example. (Warning: This is just an example. Do not attempt to use it.)

```
qvm-shutdown --wait --all; qvm-prefs fedora-01-dvm template fedora-02; qvm-
↪start sys-usb
```

3. **Base your app qubes on the new template.** In the Qube Manager, click on the Template heading to sort by template. Select all the qubes based on the old template by clicking on the first one, holding shift, then clicking on the last one. With multiple qubes selected, right-click on any of them, hover your cursor over Template, then click on the new template.

## Advanced

The following sections cover advanced topics pertaining to templates.

### Inheritance and persistence

Whenever an app qube is created, the contents of the `/home` directory of its parent template are *not* copied to the child app qube's `/home`. The child app qube's `/home` is always independent from its parent template's `/home`, which means that any subsequent changes to the parent template's `/home` will not affect the child app qube's `/home`.

Once an app qube has been created, any changes in its `/home`, `/usr/local`, or `/rw/config` directories will be persistent across reboots, which means that any files stored there will still be available after restarting the app qube. No changes in any other directories in app qubes persist in this manner. If you would like to make changes in other directories which *do* persist in this manner, you must make those changes in the parent template.

Qube Type	Inheritance <sup>1</sup>	Persistence <sup>2</sup>
template	N/A (templates cannot be based on templates)	everything
app qube <sup>3</sup>	<code>/etc/skel</code> to <code>/home</code> ; <code>/usr/local.orig</code> to <code>/usr/local</code>	<code>/rw</code> (includes <code>/home</code> , <code>/usr/local</code> , and <code>bind-dirs</code> )
disposable	<code>/rw</code> (includes <code>/home</code> , <code>/usr/local</code> , and <code>bind-dirs</code> )	nothing

### Trusting your templates

As the template is used for creating filesystems for other app qubes where you actually do the work, it means that the template is as trusted as the most trusted app qube based on this template. In other words, if your template gets compromised, e.g. because you installed an application, whose *installer's scripts* were malicious, then *all* your app qubes (based on this template) will inherit this compromise.

There are several ways to deal with this problem:

- Only install packages from trusted sources – e.g. from the pre-configured Fedora repositories. All those packages are signed by Fedora, and we expect that at least the package's installation scripts are not malicious. This

<sup>1</sup> Upon creation

<sup>2</sup> Following shutdown

<sup>3</sup> Includes *disposable templates*

is enforced by default (at the *firewall qube level*), by not allowing any networking connectivity in the default template, except for access to the Fedora repos.

- Use *standalones* (see below) for installation of untrusted software packages.
- Use multiple templates (see below) for different classes of domains, e.g. a less trusted template, used for creation of less trusted app qubes, would get various packages from less trusted vendors, while the template used for more trusted app qubes will only get packages from the standard Fedora repos.

Some popular questions:

So, why should we actually trust Fedora repos – it also contains large amount of third-party software that might be buggy, right?

As far as the template's compromise is concerned, it doesn't really matter whether `/usr/bin/firefox` is buggy and can be exploited, or not. What matters is whether its *installation* scripts (such as `%post` in the `rpm.spec`) are benign or not. A template should be used only for installation of packages, and nothing more, so it should never get a chance to actually run `/usr/bin/firefox` and get infected from it, in case it was compromised. Also, some of your more trusted app qubes would have networking restrictions enforced by the *firewall qube*, and again they should not fear this proverbial `/usr/bin/firefox` being potentially buggy and easy to compromise.

But why trust Fedora?

Because we chose to use Fedora as a vendor for the Qubes OS foundation (e.g. for `dom0` packages and for app qube packages). We also chose to trust several other vendors, such as Xen.org, kernel.org, and a few others whose software we use in `dom0`. We had to trust *somebody* as we are unable to write all the software from scratch ourselves. But there is a big difference in trusting all Fedora packages to be non-malicious (in terms of installation scripts) vs. trusting all those packages are non-buggy and non-exploitable. We certainly do not assume the latter.

So, are the templates as trusted as `dom0`?

Not quite. `Dom0` compromise is absolutely fatal, and it leads to Game Over™. However, a compromise of a template affects only a subset of all your app qubes (in case you use more than one template, or also some *standalones*). Also, if your app qubes are network disconnected, even though their filesystems might get compromised due to the corresponding template compromise, it still would be difficult for the attacker to actually leak out the data stolen in an app qube. Not impossible (due to existence of covert channels between VMs on x86 architecture), but difficult and slow.

### Note on treating app qubes' root filesystem non-persistence as a security feature

Any app qube that is based on a template has its root filesystem non-persistent across qube reboots. In other words, whatever changes the qube makes (or the malware running in this qube makes) to its root filesystem, are automatically discarded whenever one restarts the qube.

This might seem like an excellent anti-malware mechanism to be used inside the qube. However, one should be careful with treating this property as a reliable way to keep the qube malware-free. This is because the non-persistence, in the case of normal qubes, applies only to the root filesystem and not to the user filesystem (on which the `/home`, `/rw`, and `/usr/local` are stored) for obvious reasons. It is possible that malware, especially malware that could be specifically written to target Qubes, could install its hooks inside the user home directory files only. Examples of obvious places for such hooks could be: `.bashrc`, the Firefox profile directory which contains the extensions, or some PDF or DOC documents that are expected to be opened by the user frequently (assuming the malware found an exploitable bug in the PDF or DOC reader), and surely many others places, all in the user's home directory.

One advantage of the non-persistent rootfs though, is that the malware is still inactive before the user's filesystem gets mounted and "processed" by system/applications, which might theoretically allow for some scanning programs (or a skilled user) to reliably scan for signs of infections of the app qube. But, of course, the problem of finding malware hooks in general is hard, so this would work likely only for some special cases (e.g. an app qube which doesn't use Firefox, as otherwise it would be hard to scan the Firefox profile directory reliably to find malware hooks there). Also note that the user filesystem's metadata might get maliciously modified by malware in order to exploit a hypothetical

bug in the app qube kernel whenever it mounts the malformed filesystem. However, these exploits will automatically stop working (and so the infection might be cleared automatically) after the hypothetical bug got patched and the update applied (via template update), which is an exceptional feature of Qubes OS.

Also note that disposable qubes do not have persistent user filesystem, and so they start up completely “clean” every time. Note the word “clean” means in this context: the same as their template filesystem, of course.

## Important Notes

- `qvm-trim-template` is no longer necessary or available in Qubes 4.0 and higher. All qubes are created in a thin pool and trimming is handled automatically. No user action is required. See [Disk Trim](#) for more information.
- RPM-installed templates are “system managed” and therefore cannot be backed up using Qubes’ built-in backup function. In order to ensure the preservation of your custom settings and the availability of a “known-good” backup template, you may wish to clone the default system template and use your clone as the default template for your app qubes.
- Some templates are available in ready-to-use binary form, but some of them are available only as source code, which can be built using the [Qubes Builder](#). In particular, some template “flavors” are available in source code form only. For the technical details of the template system, please see [Template Implementation](#). Take a look at the [Qubes Builder](#) documentation for instructions on how to compile them.

### 1.12.25 Fedora templates

The Fedora *template* is the default template in Qubes OS. This page is about the standard (or “full”) Fedora template. For the minimal and Xfce versions, please see the [Minimal templates](#) and [Xfce templates](#) pages.

## Installing

To *install* a specific Fedora template that is not currently installed in your system, use the following command in dom0:

```
$ sudo qubes-dom0-update qubes-template-fedora-XX
```

(Replace `XX` with the Fedora version number of the template you wish to install.)

To reinstall a Fedora template that is already installed in your system, see [How to Reinstall a template](#).

## After Installing

After installing a fresh Fedora template, we recommend performing the following steps:

1. *Update the template.*
2. *Switch any app qubes that are based on the old template to the new one.*
3. If desired, *uninstall the old template.*

## Installing software

See *How to Install Software*.

## Updating

For routine daily updates within a given release, see *How to Update*.

## Upgrading

There are two ways to upgrade your template to a new Fedora release:

- **Recommended:** *Install a fresh template to replace the existing one.* This option may be simpler for less experienced users. After you install the new template, redo all desired template modifications and *switch everything that was set to the old template to the new template*. You may want to write down the modifications you make to your templates so that you remember what to redo on each fresh install. To see a log of package manager actions, open a terminal in the old Fedora template and use the `dnf history` command.
- **Advanced:** *Perform an in-place upgrade of an existing Fedora template.* This option will preserve any modifications you've made to the template, **but it may be more complicated for less experienced users.**

### 1.12.26 How to upgrade a Fedora template in-place

**Danger:** Warning: This page is intended for advanced users only. Most users seeking to upgrade should instead install a new Fedora template. Learn more about the two options here.

This page provides instructions for performing an in-place upgrade of an installed *Fedora Template*. If you wish to install a new, unmodified Fedora template instead of upgrading a template that is already installed in your system, please see the *Fedora Template* page instead. (*Learn more about the two options.*)

#### Summary instructions for standard Fedora templates

**Note:** The prompt on each line indicates where each command should be entered: `dom0`, `fedora-<old>`, or `fedora-<new>`, where `<old>` is the Fedora version number *from* which you are upgrading, and `<new>` is the Fedora version number *to* which you are upgrading.

```
[user@dom0 ~]$ qvm-clone fedora-<old> fedora-<new>
[user@dom0 ~]$ truncate -s 5GB /var/tmp/template-upgrade-cache.img
[user@dom0 ~]$ qvm-run -a fedora-<new> gnome-terminal
[user@dom0 ~]$ dev=$(sudo losetup -f --show /var/tmp/template-upgrade-cache.img)
[user@dom0 ~]$ qvm-block attach fedora-<new> dom0: ${dev}##*/}
[user@fedora-<new> ~]$ sudo mkfs.ext4 /dev/xvdi
[user@fedora-<new> ~]$ sudo mount /dev/xvdi /mnt/removable
[user@fedora-<new> ~]$ sudo dnf clean all
[user@fedora-<new> ~]$ sudo dnf --releasever=<new> --setopt=cachedir=/mnt/removable --
↪best --allowerasing distro-sync
[user@dom0 ~]$ qvm-shutdown fedora-<new>
[user@dom0 ~]$ sudo losetup -d $dev
[user@dom0 ~]$ rm /var/tmp/template-upgrade-cache.img
```

**Recommended:** *Switch everything that was set to the old template to the new template.*

## Detailed instructions for standard Fedora templates

These instructions will show you how to upgrade the standard Fedora template. The same general procedure may be used to upgrade any template based on the standard Fedora template.

**Note:** The prompt on each line indicates where each command should be entered: `dom0`, `fedora-<old>`, or `fedora-<new>`, where `<old>` is the Fedora version number *from* which you are upgrading, and `<new>` is the Fedora version number *to* which you are upgrading.

1. Ensure the existing template is not running.

```
[user@dom0 ~]$ qvm-shutdown fedora-<old>
```

2. Clone the existing template and start a terminal in the new template.

```
[user@dom0 ~]$ qvm-clone fedora-<old> fedora-<new>
[user@dom0 ~]$ qvm-run -a fedora-<new> gnome-terminal
```

3. Attempt the upgrade process in the new template.

```
[user@fedora-<new> ~]$ sudo dnf clean all
[user@fedora-<new> ~]$ sudo dnf --releasever=<new> distro-sync --best --alloweraseing
```

**Note:** `dnf` might ask you to approve importing a new package signing key. For example, you might see a prompt like this one:

```
warning: /mnt/removable/updates-0b4cc238d1aa4ffe/packages/example-package.fc<new>.
↳x86_64.rpm: Header V3 RSA/SHA256 Signature, key ID XXXXXXXX: NOKEY
Importing GPG key 0xXXXXXXXX:
Userid      : "Fedora <new> (<new>) <fedora-<new>@fedoraproject.org>"
Fingerprint: XXXX XXXX XXXX XXXX XXXX  XXXX XXXX XXXX XXXX XXXX
From        : /etc/pki/rpm-gpg/RPM-GPG-KEY-fedora-<new>-x86_64
Is this ok [y/N]: y
```

This key was already checked when it was installed (notice that the “From” line refers to a location on your local disk), so you can safely say yes to this prompt. **Note:** If you encounter no errors, proceed to step 4. If you do encounter errors, see the next two points first.

- If `dnf` reports that you do not have enough free disk space to proceed with the upgrade process, create an empty file in `dom0` to use as a cache and attach it to the template as a virtual disk.

```
[user@dom0 ~]$ truncate -s 5GB /var/tmp/template-upgrade-cache.img
[user@dom0 ~]$ dev=$(sudo losetup -f --show /var/tmp/template-upgrade-cache.img)
[user@dom0 ~]$ qvm-block attach fedora-<new> dom0:${dev}##*/}
```

Then reattempt the upgrade process, but this time use the virtual disk as a cache.

```
[user@fedora-<new> ~]$ sudo mkfs.ext4 /dev/xvdi
[user@fedora-<new> ~]$ sudo mount /dev/xvdi /mnt/removable
[user@fedora-<new> ~]$ sudo dnf clean all
[user@fedora-<new> ~]$ sudo dnf --releasever=<new> --setopt=cachedir=/mnt/
↳removable --best --alloweraseing distro-sync
```

If this attempt is successful, proceed to step 4.

- `dnf` may complain: `At least X MB more space needed on the / filesystem.` In this case, one option is to *resize the template's disk image* before reattempting the upgrade process. (See *Additional*



*Information* below for other options.)

4. Check that you are on the correct (new) Fedora release. Do this check only after completing the upgrade process. This is *not* a troubleshooting procedure for fixing download issues from the repository. This check simply verifies that your clone has successfully been upgraded.

```
[user@fedora-<new> ~]$ cat /etc/fedora-release
```

5. (Optional) Trim the new template. (This should *no longer be necessary*, but it does not hurt. Some users have *reported* that it makes a difference.)

```
[user@fedora-<new> ~]$ sudo fstrim -av
[user@dom0 ~]$ qvm-shutdown fedora-<new>
[user@dom0 ~]$ qvm-start fedora-<new>
[user@fedora-<new> ~]$ sudo fstrim -av
```

6. Shut down the new template.

```
[user@dom0 ~]$ qvm-shutdown fedora-<new>
```

7. Remove the cache file, if you created one.

```
[user@dom0 ~]$ sudo losetup -d $dev
[user@dom0 ~]$ rm /var/tmp/template-upgrade-cache.img
```

8. (Recommended) *Switch everything that was set to the old template to the new template.*

9. (Optional) Make the new template the global default.

```
[user@dom0 ~]$ qubes-prefs --set default_template fedora-<new>
```

10. (Optional) *Uninstall the old template.* Make sure that the template you're uninstalling is the old one, not the new one!

## Summary instructions for Fedora Minimal templates

**Note:** The prompt on each line indicates where each command should be entered: dom0, fedora-<old>, or fedora-<new>, where <old> is the Fedora version number *from* which you are upgrading, and <new> is the Fedora version number *to* which you are upgrading.

```
[user@dom0 ~]$ qvm-clone fedora-<old>-minimal fedora-<new>-minimal
[user@dom0 ~]$ qvm-run -u root -a fedora-<new>-minimal xterm
[root@fedora-<new>-minimal ~]# dnf clean all
[user@fedora-<new>-minimal ~]# dnf --releasever=<new> --best --allowdowngrading distro-sync
[user@fedora-<new>-minimal ~]# fstrim -v /
```

(Shut down template by any normal means.)

(If you encounter insufficient space issues, you may need to use the methods described for the standard template above.)



## Standalones

The procedure for upgrading a Fedora *standalone* is the same as for a template.

## Release-specific notes

See the [news](#) announcement for each specific template release for any important notices about that particular release.

## End-of-life (EOL) releases

We strongly recommend against using any Fedora release that has reached [end-of-life \(EOL\)](#). Also see [supported releases](#).

## Additional information

As mentioned above, you may encounter the following `dnf` error:

```
At least X MB more space needed on the / filesystem.
```

In this case, you have several options:

1. [Increase the template's disk image size](#). This is the solution mentioned in the main instructions above.
2. Delete files in order to free up space. One way to do this is by uninstalling packages. You may then reinstall them again after you finish the upgrade process, if desired). However, you may end up having to increase the disk image size anyway (see previous option).
3. Do the upgrade in parts, e.g., by using package groups. (First upgrade `@core` packages, then the rest.)
4. Do not perform an in-place upgrade, see [Upgrading Fedora templates](#).

## 1.12.27 Debian templates

The Debian *template* is an officially [supported](#) template in Qubes OS. This page is about the standard (or “full”) Debian template. For the minimal version, please see the [Minimal templates](#) page. There is also a [Qubes page on the Debian Wiki](#).

## Installing

To *install* a specific Debian template that is not currently installed in your system, use the following command in `dom0`:

```
$ sudo qubes-dom0-update qubes-template-debian-XX
```

(Replace `XX` with the Debian version number of the template you wish to install.)

To reinstall a Debian template that is already installed in your system, see [How to Reinstall a template](#).

### After Installing

After installing a fresh Debian template, we recommend performing the following steps:

1. *Update the template.*
2. *Switch any app qubes that are based on the old template to the new one.*
3. If desired, *uninstall the old template.*

### Installing software

See *How to Install Software*.

### Updating

For routine daily updates within a given release, see *How to Update*.

### Upgrading

There are two ways to upgrade your template to a new Debian release:

- **Recommended:** *Install a fresh template to replace the existing one.* **This option may be simpler for less experienced users.** After you install the new template, redo all desired template modifications and *switch everything that was set to the old template to the new template*. You may want to write down the modifications you make to your templates so that you remember what to redo on each fresh install. In the old Debian template, see `/var/log/dpkg.log` and `/var/log/apt/history.log` for logs of package manager actions.
- **Advanced:** *Perform an in-place upgrade of an existing Debian template.* This option will preserve any modifications you've made to the template, **but it may be more complicated for less experienced users.**

### Release-specific notes

This section contains notes about specific Debian releases.

#### Debian 10

Debian 10 (buster) - minimal:

```
[user@dom0 ~]$ sudo qubes-dom0-update --enablerepo=qubes-templates-itl qubes-template-  
↪debian-10-minimal
```

Debian 10 (buster) - stable:

```
[user@dom0 ~]$ sudo qubes-dom0-update --enablerepo=qubes-templates-itl qubes-template-  
↪debian-10
```

## Starting services

The Debian way (generally) is to start daemons if they are installed. This means that if you install (say) `ssh-server` in a template, *all* the qubes that use that template will run a `ssh` server when they start. (They will, naturally, all have the same server key.) This may not be what you want.

So be very careful when installing software in Templates - if the daemon spawns outbound connections then there is a serious security risk.

In general, a reasonable approach would be, (using `ssh` as example):

- Install the `ssh` service.
- `systemctl stop ssh`
- `systemctl disable ssh`
- `systemctl mask ssh`
- Close down template

Now the `ssh` service will **NOT** start in qubes based on this template.

Where you **DO** want the service to run, put this in `/rw/config/rc.local`:

```
systemctl unmask ssh
systemctl start ssh
```

Don't forget to make the file executable.

## Unattended Upgrades

Some users have noticed that on upgrading to Stretch, the `unattended-upgrade` package is installed.

This package is pulled in as part of a Recommend chain, and can be purged.

The lesson is that you should carefully look at what is being installed to your system, particularly if you run `dist-upgrade`.

## Package installation errors in Qubes 4.0

If some packages throw installation errors, see [this guide](#).

### 1.12.28 How to upgrade a Debian template in-place

**Danger:** Warning: This page is intended for advanced users only. Most users seeking to upgrade should instead install a new Debian template. Learn more about the two options [here](#).

This page provides instructions for performing an in-place upgrade of an installed *Debian Template*. If you wish to install a new, unmodified Debian template instead of upgrading a template that is already installed in your system, please see the *Debian Template* page instead. (*Learn more about the two options.*) In general, upgrading a Debian template follows the same process as [upgrading a native Debian system](#).

## Summary instructions for Debian templates

**Important:** The prompt on each line indicates where each command should be entered: `dom0`, `debian-<old>`, or `debian-<new>`, where `<old>` is the Debian version number *from* which you are upgrading, and `<new>` is the Debian version number *to* which you are upgrading. The instructions may differ for certain releases. See [release-specific notes](#) for any instructions specific to your particular release.

```
[user@dom0 ~]$ qvm-clone debian-<old> debian-<new>
[user@dom0 ~]$ qvm-run -a debian-<new> gnome-terminal
[user@debian-<new> ~]$ sudo sed -i 's/<old-name>/<new-name>/g' /etc/apt/sources.list
[user@debian-<new> ~]$ sudo sed -i 's/<old-name>/<new-name>/g' /etc/apt/sources.list.d/
↪qubes-r4.list
[user@debian-<new> ~]$ sudo apt update
[user@debian-<new> ~]$ sudo apt upgrade
[user@debian-<new> ~]$ sudo apt dist-upgrade
[user@dom0 ~]$ qvm-shutdown debian-<new>
```

**Recommended:** *Switch everything that was set to the old template to the new template.*

## Detailed instructions for Debian templates

These instructions will show you how to upgrade Debian templates. The same general procedure may be used to upgrade any template based on the standard Debian template.

**Important:** The prompt on each line indicates where each command should be entered: `dom0`, `debian-<old>`, or `debian-<new>`, where `<old>` is the Debian version number *from* which you are upgrading, and `<new>` is the Debian version number *to* which you are upgrading. The instructions may differ for certain releases. See [release-specific notes](#) for any instructions specific to your particular release.

1. Ensure the existing template is not running.

```
[user@dom0 ~]$ qvm-shutdown debian-<old>
```

2. Clone the existing template and start a terminal in the new template.

```
[user@dom0 ~]$ qvm-clone debian-<old> debian-<new>
[user@dom0 ~]$ qvm-run -a debian-<new> gnome-terminal
```

3. Update your apt repositories to use the new release's code name instead of the old release's code name. (This can be done manually with a text editor, but `sed` can be used to automatically update the files.)

```
[user@debian-<new> ~]$ sudo sed -i 's/<old-name>/<new-name>/g' /etc/apt/sources.list
[user@debian-<new> ~]$ sudo sed -i 's/<old-name>/<new-name>/g' /etc/apt/sources.
↪list.d/qubes-r4.list
```

4. Update the package lists and upgrade. During the process, it may prompt you to overwrite the file `qubes-r4.list`. You should overwrite this file.

```
[user@debian-<new> ~]$ sudo apt update
[user@debian-<new> ~]$ sudo apt upgrade
[user@debian-<new> ~]$ sudo apt dist-upgrade
```

5. (Optional) Remove unnecessary packages that were previously installed.

```
[user@debian-<new> ~]$ sudo apt-get autoremove
```

6. (Optional) Clean cached packages from `/var/cache/apt`.

```
[user@debian-<new> ~]$ sudo apt-get clean
```

7. (Optional) Trim the new template. (This should *no longer be necessary*, but it does not hurt. Some users have reported that it makes a difference.)

```
[user@debian-<new> ~]$ sudo fstrim -av
[user@dom0 ~]$ qvm-shutdown debian-<new>
[user@dom0 ~]$ qvm-start debian-<new>
[user@debian-<new> ~]$ sudo fstrim -av
```

8. Shut down the new template.

```
[user@dom0 ~]$ qvm-shutdown debian-<new>
```

9. (Recommended) *Switch everything that was set to the old template to the new template.*

10. (Optional) Make the new template the global default.

```
[user@dom0 ~]$ qubes-prefs --set default_template debian-<new>
```

11. (Optional) *Uninstall the old template.* Make sure that the template you're uninstalling is the old one, not the new one!

## Standalones

The procedure for upgrading a Debian *standalone* is the same as for a template.

## Release-specific notes

This section contains notes about upgrading to specific releases.

### Debian 11 (“Bullseye”)

Please see [Debian’s Bullseye upgrade instructions](#). In particular: for APT source lines referencing the security archive, the format has changed slightly along with the release name, going from `buster/updates` to `bullseye-security`; see [Section 5.1.2, “Changed security archive layout”](#).

This means that, when upgrading from Buster to Bullseye, an additional `sed` command is required:

```
[user@dom0 ~]$ qvm-clone debian-10 debian-11
[user@dom0 ~]$ qvm-run -a debian-11 gnome-terminal
[user@debian-<new> ~]$ sudo sed -i 's/buster/bullseye/g' /etc/apt/sources.list
[user@debian-<new> ~]$ sudo sed -i 's/debian-security bullseye\/updates/debian-security-
↪bullseye-security/g' /etc/apt/sources.list
[user@debian-<new> ~]$ sudo sed -i 's/buster/bullseye/g' /etc/apt/sources.list.d/qubes-
↪r4.list
[user@debian-<new> ~]$ sudo apt update
[user@debian-<new> ~]$ sudo apt upgrade
```

(continues on next page)

(continued from previous page)

```
[user@debian-<new> ~]$ sudo apt dist-upgrade
[user@dom0 ~]$ qvm-shutdown debian-11
```

### Debian 10 (“Buster”)

Please see [Debian’s Buster upgrade instructions](#).

### Debian 9 (“Stretch”)

- The upgrade process may prompt you to overwrite two files: `qubes-r4.list` and `pulse/client.conf`. `qubes-r4.list` can be overwritten, but `pulse/client.conf` must be left as the currently-installed version.
- If sound is not working, you may need to enable the Qubes testing repository to get the testing version of `qubes-gui-agent`. This can be done by editing the `/etc/apt/sources.list.d/qubes-r4.list` file and uncommenting the Qubes Updates Candidates repo.
- User-initiated updates/upgrades may not run when a template first starts. This is due to a new Debian config setting that attempts to update automatically; it should be disabled with `sudo systemctl disable apt-daily.{service,timer}`.

Relevant discussions:

- [Stretch Template Installation](#)
- [Stretch availability in 3.2](#)
- [Fixing sound in Debian Stretch](#)
- [User apt commands blocked on startup](#)

Also see [Debian’s Stretch upgrade instructions](#).

### Debian 8 (“Jessie”)

Please see [Debian’s Jessie upgrade instructions](#).

### End-of-life (EOL) releases

We strongly recommend against using any Debian release that has reached [end-of-life \(EOL\)](#).

### Additional information

- Please note that, if you installed packages from one of the [testing](#) repositories, you must make sure that the repository is enabled in `/etc/apt/sources.list.d/qubes-r4.list` before attempting the upgrade. Otherwise, your upgrade will [break](#).
- By default, Qubes uses code names in the apt sources files, although the templates are referred to by release number. Check the code names for the templates, and ensure you are aware of any changes you have made in the repository definitions.

### 1.12.29 Minimal templates

The minimal *templates* are lightweight versions of their standard template counterparts. They have only the most vital packages installed, including a minimal X and xterm installation. When properly configured and used, minimal templates can be less resource-intensive, reduce attack surface, and support more fine-grained compartmentalization. The sections below contain instructions for installing and configuring minimal templates, along with some examples of common use cases.

#### Important

1. The minimal templates are intended only for advanced users. Most things will *not* work out-of-the-box, and you *will* have to fix them yourself. If you are not prepared to do a lot of reading, searching, learning, and troubleshooting, then you should instead stick to the standard templates, which are perfectly suitable for the vast majority of users. In particular, if you are new to Qubes, you should not attempt to use minimal templates until you have gained more experience.
2. If something works with a standard template but not the minimal version, this is most likely due to user error (e.g., a missing package or misconfiguration) rather than a bug. In such cases, please do *not* file a bug report. Instead, please see [Help, Support, Mailing Lists, and Forum](#) for the appropriate place to ask for help. Once you have learned how to solve your problem, please contribute what you learned to the documentation.
3. The minimal templates are intentionally *minimal*. *Do not ask for your favorite package to be added to the minimal template by default.*
4. In order to reduce unnecessary risk, unused repositories have been disabled by default. If you wish to install or update any packages from those repositories, you must enable them.

#### List

Minimal templates of the following distros are available:

- Fedora
- Debian
- CentOS
- Gentoo

A list of all available templates can also be obtained with the *Template Manager* tool.

#### Installation

The minimal templates can be installed with the following type of command:

```
[user@dom0 ~]$ sudo qubes-dom0-update qubes-template-<DISTRO_NAME>-<RELEASE_NUMBER>-minimal
```

If your desired version is not found, it may still be in *testing*. You may wish to try again with the testing repository enabled:

```
[user@dom0 ~]$ sudo qubes-dom0-update --enablerepo=qubes-templates-itl-testing qubes-template-<DISTRO_NAME>-<RELEASE_NUMBER>-minimal
```

If you would like to install a community distribution, try the install command by enabling the community repository:

```
[user@dom0 ~]$ sudo qubes-dom0-update --enablerepo=qubes-templates-community qubes-  
↪template-<DISTRO_NAME>-<RELEASE_NUMBER>-minimal
```

The download may take a while depending on your connection speed.

### Passwordless root

It is an intentional design choice for *Passwordless Root Access in VMs* to be optional in minimal templates. Since the minimal templates are *minimal*, they are not configured for passwordless root by default. To update or install packages, execute the following command in dom0:

```
[user@dom0 ~]$ qvm-run -u root <DISTRO_NAME>-<RELEASE_NUMBER>-minimal xterm
```

This opens a root terminal in the minimal template, from which you can use execute root commands without `sudo`. You will have to do this every time if you choose not to enable passwordless root.

If you want to be able to use `sudo` inside a minimal template (or app qubes based on a minimal template), open a root terminal as just instructed, then install the `qubes-core-agent-passwordless-root` package.

Optionally, verify that passwordless root now works by opening a normal (non-root) xterm window in the minimal template, then issue the command `sudo -l`. This should give you output that includes the `NOPASSWD` keyword.

### Customization

You may wish to clone the original template and make any changes in the clone instead of the original template. You must start the clone in order to customize it.

Customizing the template for specific use cases normally only requires installing additional packages.

### Distro-specific notes

The following sections provide information that is specific to a particular minimal template distro.

#### Fedora

The following list provides an overview of which packages are needed for which purpose. As usual, the required packages are to be installed in the running template with the following command (replace `packages` with a space-delimited list of packages to be installed):

```
[user@your-new-clone ~]$ sudo dnf install packages
```

- Commonly used utilities: `pciutils vim-minimal less psmisc gnome-keyring`.
- Audio: `pulseaudio-qubes`.
- Networking: `qubes-core-agent-networking`, and whatever network tools you want. N.B. minimal templates do not include any browser.
- *FirewallVM*, such as the template for `sys-firewall`: at least `qubes-core-agent-networking` and `iproute`, and also `qubes-core-agent-dom0-updates` if you want to use it as the `UpdateVM` (which is normally `sys-firewall`).



- NetVM, such as the template for `sys-net`: `qubes-core-agent-networking` `qubes-core-agent-network-manager` `NetworkManager-wifi` `network-manager-applet` `wireless-tools` `notification-daemon` `gnome-keyring` `polkit` `@hardware-support`. If your network devices need extra packages for the template to work as a network VM, use the `lspci` command to identify the devices, then run `dnf search firmware` (replace `firmware` with the appropriate device identifier) to find the needed packages and then install them. If you need utilities for debugging and analyzing network connections, install `tcpdump` `telnet` `nmap` `nmap-ncat`.
- *USB qube*, such as the template for `sys-usb`: `qubes-usb-proxy` to provide USB devices to other Qubes and `qubes-input-proxy-sender` to provide keyboard or mouse input to `dom0`.
- *VPN qube*: Use the `dnf search "NetworkManager VPN plugin"` command to look up the VPN packages you need, based on the VPN technology you'll be using, and install them. Some GNOME related packages may be needed as well. After creation of a machine based on this template, follow the [VPN instructions](#) to configure it.
- `default-mgmt-dvm`: requires `qubes-core-agent-passwordless-root` and `qubes-mgmt-salt-vm-connector`.

In Qubes 4.0, additional packages from the `qubes-core-agent` suite may be needed to make the customized minimal template work properly. These packages are:

- `qubes-core-agent-nautilus`: This package provides integration with the Nautilus file manager (without it, items like “copy to VM/open in disposable” will not be shown in Nautilus).
- `qubes-core-agent-thunar`: This package provides integration with the thunar file manager (without it, items like “copy to VM/open in disposable” will not be shown in thunar).
- `qubes-core-agent-dom0-updates`: Script required to handle `dom0` updates. Any template on which the qube responsible for ‘`dom0`’ updates (e.g. `sys-firewall`) is based must contain this package.
- `qubes-menus`: Defines menu layout.
- `qubes-desktop-linux-common`: Contains icons and scripts to improve desktop experience.
- `qubes-core-agent-qrexec`: Qubes qrexec agent. Installed by default.
- `qubes-core-agent-systemd`: Qubes unit files for SystemD init style. Installed by default.
- `qubes-core-agent-passwordless-root`, `polkit`: By default, the Fedora minimal template doesn't have passwordless root. These two packages enable this feature.
- `qubes-core-agent-sysvinit`: Qubes unit files for SysV init style or upstart.

Also, there are packages to provide additional services:

- `qubes-gpg-split`: For implementing split GPG.
- `qubes-u2f`: For implementing secure forwarding of U2F messages.
- `qubes-pdf-converter`: For implementing safe conversion of PDFs.
- `qubes-img-converter`: For implementing safe conversion of images.
- `qubes-snapd-helper`: If you want to use snaps in qubes.
- `thunderbird-qubes`: Additional tools for use in thunderbird.
- `qubes-app-shutdown-idle`: If you want qubes to automatically shutdown when idle.
- `qubes-mgmt-salt-vm-connector`: If you want to use salt management on the template and qubes.

You may also wish to consider additional packages from the `qubes-core-agent` suite.

See [here](#) for further information on customizing `fedora-minimal`.

## Logging

The `rsyslog` logging service is not installed by default, as all logging is instead being handled by the `systemd` journal. Users requiring the `rsyslog` service should install it manually.

To access the `journal` log, use the `journalctl` command.

## Debian

The following list provides an overview of which packages are needed for which purpose. As usual, the required packages are to be installed in the running template with the following command (replace packages with a space-delimited list of packages to be installed):

```
[user@your-new-clone ~]$ sudo apt install packages
```

- Commonly used utilities: `pciutils vim-minimal less psmisc gnome-keyring`
- The `zenity` package is required for interactive dialogs, e.g., file selection ([#5202](#)) and for using the Nautilus menu option to copy some files to other qubes ([#6801](#)).
- Audio: `pulseaudio-qubes`
- Networking: `qubes-core-agent-networking`, and whatever network tools you want. N.B. minimal templates do not include any browser.
- [FirewallVM](#), such as the template for `sys-firewall`: at least `qubes-core-agent-networking`, and also `qubes-core-agent-dom0-updates` if you want to use it as the `UpdateVM` (which is normally `sys-firewall`).
- NetVM, such as the template for `sys-net`: `qubes-core-agent-networking` `qubes-core-agent-network-manager`. If your network devices need extra packages for a network VM, use the `lspci` command to identify the devices, then find the package that provides necessary firmware and install it. If you need utilities for debugging and analyzing network connections, install the following packages: `tcpdump telnet nmap ncat`.
- [USB qube](#), such as the template for `sys-usb`: `qubes-usb-proxy` to provide USB devices to other Qubes and `qubes-input-proxy-sender` to provide keyboard or mouse input to dom0.
- Qubes to which USB devices are attached: `libpam-systemd` (Until [#7689](#) is fixed, either pair it with `qubes-core-agent-passwordless-root` or manually activate the user session with `loginctl activate <USER_SESSION_ID>`.)
- [VPN qube](#): You may need to install network-manager VPN packages, depending on the VPN technology you'll be using. After creating a machine based on this template, follow the [VPN howto](#) to configure it.
- `default-mgmt-dvm`: requires `qubes-core-agent-passwordless-root` and `qubes-mgmt-salt-vm-connector`.
- [Yubikey](#): You may need to install `xserver-xorg-input-libinput` for 2FA responses to work in web browsers like Firefox.
- Thumbnails (e.g., file previews in Nautilus): `libgdk-pixbuf2.0-bin` (images), `ffmpegthumbnailer` (videos). (Try `apt search thumbnailer` for other file types.)

In Qubes 4.0, additional packages from the `qubes-core-agent` suite may be needed to make the customized minimal template work properly. These packages are:

- `qubes-core-agent-nautilus`: This package provides integration with the Nautilus file manager (without it, items like “copy to VM/open in disposable” will not be shown in Nautilus).

- `qubes-core-agent-thunar`: This package provides integration with the thunar file manager (without it, items like “copy to VM/open in disposable” will not be shown in thunar).
- `qubes-core-agent-dom0-updates`: Script required to handle dom0 updates. Any template on which the qube responsible for ‘dom0’ updates (e.g. `sys-firewall`) is based must contain this package.
- `qubes-menus`: Defines menu layout.
- `qubes-desktop-linux-common`: Contains icons and scripts to improve desktop experience.

Also, there are packages to provide additional services:

- `qubes-gpg-split`: For implementing split GPG.
- `qubes-u2f`: For implementing secure forwarding of U2F messages.
- `qubes-pdf-converter`: For implementing safe conversion of PDFs.
- `qubes-img-converter`: For implementing safe conversion of images.
- `qubes-snapd-helper`: If you want to use snaps in qubes.
- `qubes-thunderbird`: Additional tools for use in thunderbird.
- `qubes-app-shutdown-idle`: If you want qubes to automatically shutdown when idle.
- `qubes-mgmt-salt-vm-connector`: If you want to use salt management on the template and qubes.

Documentation on all of these can be found in the [docs](#).

You could, of course, use `qubes-vm-recommended` to automatically install many of these, but in that case you are well on the way to a standard Debian template.

## CentOS

The following list provides an overview of which packages are needed for which purpose. As usual, the required packages are to be installed in the running template with the following command (replace `packages` with a space-delimited list of packages to be installed):

```
[user@your-new-clone ~]$ sudo yum install packages
```

- Commonly used utilities: `pciutils vim-minimal less psmisc gnome-keyring`
- Audio: `pulseaudio-qubes`.
- Networking: `qubes-core-agent-networking`, and whatever network tools you want. N.B. minimal templates do not include any browser.
- *FirewallVM*, such as the template for `sys-firewall`: at least `qubes-core-agent-networking`, and also `qubes-core-agent-dom0-updates` if you want to use it as the UpdateVM (which is normally `sys-firewall`).
- NetVM, such as the template for `sys-net`: `qubes-core-agent-networking` `qubes-core-agent-network-manager` `NetworkManager-wifi` `network-manager-applet` `wireless-tools` `notification-daemon` `gnome-keyring`. If your network devices need extra packages for a network VM, use the `lspci` command to identify the devices, then find the package that provides necessary firmware and install it. If you need utilities for debugging and analyzing network connections, install the following packages: `tcpdump telnet nmap nmap-ncat`
- *USB qube*, such as the template for `sys-usb`: `qubes-usb-proxy` to provide USB devices to other Qubes and `qubes-input-proxy-sender` to provide keyboard or mouse input to dom0.
- *VPN qube*: You may need to install network-manager VPN packages, depending on the VPN technology you’ll be using. After creating a machine based on this template, follow the [VPN howto](#) to configure it.

- `default-mgmt-dvm:` requires `qubes-core-agent-passwordless-root` and `qubes-mgmt-salt-vm-connector`.

In Qubes 4.0, additional packages from the `qubes-core-agent` suite may be needed to make the customized minimal template work properly. These packages are:

- `qubes-core-agent-nautilus:` This package provides integration with the Nautilus file manager (without it, items like “copy to VM/open in disposable” will not be shown in Nautilus).
- `qubes-core-agent-thunar:` This package provides integration with the thunar file manager (without it, items like “copy to VM/open in disposable” will not be shown in thunar).
- `qubes-core-agent-dom0-updates:` Script required to handle dom0 updates. Any template on which the qube responsible for ‘dom0’ updates (e.g. `sys-firewall`) is based must contain this package.
- `qubes-menus:` Defines menu layout.
- `qubes-desktop-linux-common:` Contains icons and scripts to improve desktop experience.

Also, there are packages to provide additional services:

- `qubes-gpg-split:` For implementing split GPG.
- `qubes-pdf-converter:` For implementing safe conversion of PDFs.
- `qubes-img-converter:` For implementing safe conversion of images.
- `qubes-snapd-helper:` If you want to use snaps in qubes.
- `qubes-mgmt-salt-vm-connector:` If you want to use salt management on the template and qubes.

Documentation on all of these can be found in the [docs](#).

You could, of course, use `qubes-vm-recommended` to automatically install many of these, but in that case you are well on the way to a standard Debian template.

### 1.12.30 Xfce templates

If you would like to use Xfce (more lightweight compared to GNOME desktop environment) Linux distribution in your qubes, you can install one of the available Xfce templates for [Fedora](#), [Debian](#), [CentOS](#), or [Gentoo](#).

#### Installation

The Fedora Xfce templates can be installed with the following command (where `X` is your desired distro and version number):

```
[user@dom0 ~]$ sudo qubes-dom0-update qubes-template-X-xfce
```

If your desired version is not found, it may still be in [testing](#). You may wish to try again with the testing repository enabled:

```
[user@dom0 ~]$ sudo qubes-dom0-update --enablerepo=qubes-templates-itr-testing qubes-  
↪template-X-xfce
```

If you would like to install a community distribution, like CentOS or Gentoo, try the install command by enabling the community repository:

```
[user@dom0 ~]$ sudo qubes-dom0-update --enablerepo=qubes-templates-community qubes-  
↪template-X-xfce
```

If your desired version is not found, it may still be in *testing*. You may wish to try again with the testing repository enabled:

```
[user@dom0 ~]$ sudo qubes-dom0-update --enablerepo=qubes-templates-community-testing_
↪qubes-template-X-xfce
```

The download may take a while depending on your connection speed.

To reinstall a Xfce template that is already installed in your system, see *How to Reinstall a template*.

## 1.12.31 Installation troubleshooting

### “An unknown error has occurred” error during installation

Some people have encountered this error message when trying to install Qubes on drives that already have data on them. The solution is to exit the installer, wipe all data or delete all partitions, then restart the Qubes installation.

### Trouble installing from USB stick

If you are facing issues when booting using UEFI mode, see the *UEFI troubleshooting guide*.

There are a variety of other problems that could arise when using a USB installation medium, and some of the issues can be fixed by doing one or more of the following:

- **Use a different USB drive:** If possible, try several drives of different sizes and formats. This determines whether the problem stems from the flash drive or Qubes installer. Some laptops cannot read from an external boot device larger than 8GB. If you encounter a black screen when performing an installation from a USB stick, ensure you are using a USB drive less than 8GB, or a partition on that USB less than 8GB and of format FAT32. Note that the Qubes installation image is over 4GB, so it may not fit on a smaller USB. If a machine can not boot from a bigger USB, it may be too old to run Qubes.
- **Verify your Qubes ISO:** Errors will occur if the Qubes installer is corrupted. Ensure that the installer is correct and complete before writing it to a flash drive by *verifying the ISO*.
- **Change the method you used to write your ISO to a USB key :** Some people use the dd command (recommended), others use tools like Rufus, balenaEtcher or the GNOME Disk Utility. If installation fails after using one tool, try a different one. For example, if you are facing trouble installing Qubes after writing the ISO using Rufus, then try using other tools like balenaEtcher or the dd command. In case the boot partition is not set to “active” after copying the ISO, you can use some other tool like gparted on a Linux system to activate it.

### “Warning: dracut-initqueue timeout - starting timeout scripts” during installation

This error message is related to the faulty creation of the USB installation medium. If you receive this error message during installation, please make sure you have followed the instructions on *how to write your ISO to a USB key*. Specifically, the dd command listed on that page has been verified to solve this issue on multiple Qubes installation versions.

```
$ sudo dd if=Qubes-RX-x86_64.iso of=/dev/sdY status=progress bs=1048576 && sync
```

See [here](#) for a discussion of this error message.

## Boot screen does not appear / system does not detect your installation medium

If the boot screen does not appear, there are several options to troubleshoot. First, try rebooting your computer. If it still loads your currently installed operating system or does not detect your installation medium, make sure the boot order is set up appropriately.

The process to change the boot order varies depending on the currently installed system and the motherboard manufacturer.

If **Windows 10** is installed on your machine, you may need to follow specific instructions to change the boot order. This may require an [advanced reboot](#).

## “Not asking for VNC because we don’t have a network” / “X startup failed, aborting installation” / “Pane is dead” error during installation

The boot mode in use may be causing these error messages. Try to install after enabling both UEFI and legacy boot modes. If that doesn’t help, then disable one and try the other. Visit the [UEFI Troubleshooting guide](#) if other errors arise during UEFI booting.

These errors may also occur due to an incompatible Nvidia graphics card. If you have one, follow the following instructions: 1. Disable secure/fast boot and use legacy mode 2. Enter GRUB, move the selection to the first choice, and then press the Tab key. 3. Now, you are in edit mode. Move the text cursor with your arrow key and after `kernel=` line, add:

```
...
nouveau.modeset=0 rd.driver.blacklist=nouveau video=vesa:off
...

If the above code doesn't fix the problem, replace it with:

...
noexitboot=1 modprobe.blacklist=nouveau rd.driver.blacklist=nouveau --- intitrd.img
...
```

For more information, look at the [Nvidia Troubleshooting guide](#).

## Installation freezes at “Setting up Networking”

If you are facing this problem on an Apple computer, check out the [Macbook Troubleshooting guide](#).

If you are installing Qubes 4.0 on an external storage device, you may have forgotten to disable `sys-usb` during the [initial setup](#), which is generally required for that setup to work.

This issue occurs due to the network card, which may be missing some drivers or is incompatible with Qubes.

First, install all available drivers for the card. You can install the drivers without internet access by first downloading them on another machine, then transferring them over to the current machine (e.g., with a USB drive).

If installing the available drivers does not help, disable the network card in the BIOS and perform the installation before re-enabling the card. If this solves the issue, it confirms the PCI card is incompatible with Qubes. In this case, you may want to consider replacing it with a network card of a different brand. Broadcom cards are notoriously problematic with Qubes.

## “Unsupported Hardware Detected” error

During Qubes installation, you may come across the error message which reads “Unsupported Hardware Detected. Missing features: IOMMU/VT-d/AMD-Vi, Interrupt Remapping. Without these features, Qubes OS will not function normally”.

This error message indicates that IOMMU-virtualization hasn’t been activated in the BIOS. Return to the [hardware requirements](#) section to learn how to activate it. If the setting is not configured correctly, it means that your hardware won’t be able to leverage some Qubes security features, such as a strict isolation of the networking and USB hardware.

In Qubes 4.0, the default installation won’t function properly without IOMMU, as default sys-net and sys-usb qubes require IOMMU. It is possible to configure them to reduce isolation and not use IOMMU by changing virtualization mode of these two VMs to “PV”.

In Qubes 4.1, the default sys-net and sys-usb qubes need additional configuration to be usable without an IOMMU. Otherwise they will fail to start with this error message:

```
Start failed: internal error: libxenlight failed to create new domain 'sys-net', see /
↳var/log/libvirt/libxl/libxl-driver.log for details
```

To confirm that a missing IOMMU is causing this problem, check for the following error message in `/var/log/libvirt/libxl/libxl-driver.log`:

```
2022-03-01 13:27:17.117+0000: libxl: libxl_create.c:1146:libxl__domain_config_
↳setdefault: passthrough not supported on this platform
```

Here are the steps to fix this. Note that this allows sys-net and sys-usb to take complete control of the system, as described in the [FAQ here](#):

1. Change the virtualization mode of sys-net and sys-usb to “PV”
2. Add `qubes.enable_insecure_pv_passthrough` to `GRUB_CMDLINE_LINUX` in `/etc/default/grub`
3. Run `sudo grub2-mkconfig -o /boot/efi/EFI/qubes/grub.cfg`. If you are using a non-UEFI BIOS (where `/boot/efi/EFI` doesn’t exist), use the command `sudo grub-mkconfig -o /boot/grub2/grub.cfg` instead.
4. Reboot

## 1.12.32 Update troubleshooting

### “Failed to synchronize cache for repo” errors when updating Fedora templates

This is general Fedora issue, not a Qubes-specific issue. Usually, this is due to network problems (especially if downloading updates over Tor) or problems with the download mirrors. Often, the problem can be resolved by trying again on a different connection (a different Tor circuit, if using Tor) or waiting and trying again later. Here are some examples of non-Qubes reports about this problem:

- <https://ask.fedoraproject.org/en/question/88086/error-failed-to-synchronize-cache-for-repo-fedora/>
- <https://unix.stackexchange.com/questions/390805/repos-not-working-on-fedora-error-failed-to-synchronize-cache-for-repo-upda>
- [https://www.reddit.com/r/Fedora/comments/74nldq/fedora\\_26\\_dnf\\_error\\_failed\\_to\\_synchronize\\_cache/](https://www.reddit.com/r/Fedora/comments/74nldq/fedora_26_dnf_error_failed_to_synchronize_cache/)
- [https://bugzilla.redhat.com/show\\_bug.cgi?id=1494178](https://bugzilla.redhat.com/show_bug.cgi?id=1494178)
- <https://stackoverflow.com/questions/45318256/error-failed-to-synchronize-cache-for-repo-updates>

More examples can be found by searching for “Failed to synchronize cache for repo” (with quotation marks) on your preferred search engine.



## Lost internet access after a template update

In earlier versions of Qubes, there were situations where qubes lost internet access after a template update. The following fix should be applied in recent versions of Qubes.

Run `systemctl enable NetworkManager-dispatcher.service` in the template upon which your NetVM is based. You may have to reboot afterward for the change to take effect. (Note: This is an upstream problem. See [this Redhat ticket](#)). For details, see the qubes-users mailing list threads [here](#) and [here](#).)

## Windows update is stuck

This has nothing to do with Qubes. It's a longstanding Windows bug. More information about this issue and solutions can be found [here](#).

## Dom0 and/or template update stalls when updating via the GUI tool

This can usually be fixed by updating via the command line.

In dom0, open a terminal and run `sudo qubes-dom0-update`.

Depending on your operating system, open a terminal in the templates and run: \* Fedora: `sudo dnf upgrade` \* Debian: `apt-get update && apt-get dist-upgrade`

### 1.12.33 Updating Debian and Whonix

Despite Qubes shipping with *Debian Templates*, most of Qubes core components run on Fedora and thus our documentation has better coverage for Fedora. However, Qubes has been working closely with the [Whonix](#) project which is based on Debian.

This troubleshooting guide is collection of tips about updating Whonix that also pertain to updating the normal Debian package manager. If you plan to use Debian heavily, **we highly recommend you install the Whonix templates and use them to update your normal Debian template.**

*Note: some of the links on this page go to documentation on Whonix's website*

## Updating Error Messages

After running the commands to update Debian or Whonix, hopefully everything will complete perfectly.

```
sudo apt-get update && sudo apt-get dist-upgrade
```

However, if you see something like the following, then something went wrong.

```
W: Failed to fetch http://ftp.us.debian.org/debian/dist/jessie/contrib/binary-i386/
↳Packages 404 Not Found

W: Failed to fetch http://ftp.us.debian.org/debian/dist/jessie/non-free/binary-i386/
↳Packages 404 Not Found

E: Some index files failed to download. They have been ignored, or old ones used instead.

Err http://ftp.us.debian.org jessie Release.gpg
  Could not resolve 'ftp.us.debian.org'
```

(continues on next page)



(continued from previous page)

```

Err http://deb.torproject.org jessie Release.gpg
  Could not resolve 'deb.torproject.org'
Err http://security.debian.org jessie/updates Release.gpg
  Could not resolve 'security.debian.org'
Reading package lists... Done
W: Failed to fetch http://security.debian.org/dists/jessie/updates/Release.gpg  Could
↳not resolve 'security.debian.org'

W: Failed to fetch http://ftp.us.debian.org/debian/dists/jessie/Release.gpg  Could not
↳resolve 'ftp.us.debian.org'

W: Failed to fetch http://deb.torproject.org/torproject.org/dists/jessie/Release.gpg  ↳
↳Could not resolve 'deb.torproject.org'

W: Some index files failed to download. They have been ignored, or old ones used instead.

```

This could be a temporary Tor exit relay or server failure that should fix itself. Here are some simple things to try:

- Check if your network connection is functional
- Try to [change your Tor circuit](#), then try again
- Running [whonixcheck](#) might also help diagnose the problem

Sometimes if you see a message such as:

```
Could not resolve 'security.debian.org'
```

It helps to run the following command:

```
nslookup security.debian.org
```

And then trying running the update and upgrade commands again.

```
sudo apt-get update && sudo apt-get dist-upgrade
```

*Please note: if you disabled the [Whonix APT Repository](#) you'll have to manually check for new Whonix releases and manually install them from source code .*

## Never Install Unsigned Packages

If you see something like this:

```

WARNING: The following packages cannot be authenticated!
  icedove
Install these packages without verification [y/N]?

```

Don't proceed! Press N and <enter>. Running `apt-get update` again should fix it. If not, something is broken or it's a [Man in the middle attack](#), which isn't that unlikely, since we are updating over Tor exit relays and some of them are malicious. Try to [change your Tor circuit](#).

## Signature Verification Warnings

There should be none at the moment. If there was such a warning, it would look like this:

```
W: A error occurred during the signature verification. The repository is not updated and
↳ the previous index files will be used. GPG error: http://deb.torproject.org stable
↳ Release: The following signatures were invalid: KEYEXPIRED 1409325681 KEYEXPIRED
↳ 1409325681 KEYEXPIRED 1409325681 KEYEXPIRED 1409325681
```

Even though, `apt-get` will automatically ignore repositories with expired keys or signatures, you will not receive upgrades from that repository. Unless the issue is already known/documented, it should be reported so it can be further investigated.

There are two possible reasons why this could happen, either there is an issue with the repository that the maintainers have to fix, or you are victim of a [Man-in-the-middle\\_attacks](#). The latter would not be a big issue and might go away after a while automatically or try to [change your Tor circuit](#)

In past various apt repositories were signed with expired key. If you want to see how the documentation looked at that point, please click on expand on the right.

The Tor Project's apt repository key was expired. You saw the following warning.

```
W: A error occurred during the signature verification. The repository is not updated and
↳ the previous index files will be used. GPG error: http://deb.torproject.org stable
↳ Release: The following signatures were invalid: KEYEXPIRED 1409325681 KEYEXPIRED
↳ 1409325681 KEYEXPIRED 1409325681 KEYEXPIRED 1409325681
```

```
W: Failed to fetch http://deb.torproject.org/torproject.org/dists/stable/Release
W: Some index files failed to download. They have been ignored, or old ones used instead.
```

It had already been [reported](#). There was no immediate danger. You could have just ignored it. Just make sure, you never install unsigned packages as explained above.

If you were to see other signature verification errors, those should be reported, but it shouldn't happen at this time.

## Changed Configuration Files

If you see something like the following.

```
Setting up ifupdown ...
Configuration file /etc/network/interfaces
==> Modified (by you or by a script) since installation.
==> Package distributor has shipped an updated version.
What would you like to do about it ? Your options are:
  Y or I : install the package maintainer's version
  N or O : keep your currently-installed version
  D      : show the differences between the versions
  Z      : background this process to examine the situation
The default action is to keep your current version.
*** interfaces (Y/I/N/O/D/Z) [default=N] ? N
```

Be careful. If the updated file isn't coming from Whonix specific package (some are called `whonix-...`), then press `n`. Otherwise anonymity/privacy/security settings deployed with Whonix might get lost. If you are an advanced user and know better, you can of course manually check the difference and merge them.

How could you find out if the file is coming from a Whonix specific package or not?

- Whonix specific packages are sometimes called `whonix-...`. In the example above it's saying `Setting up ifupdown ...`, so the file isn't coming from a Whonix specific package. In this case, you should press `n` as advised in the paragraph above.
- If the package name does include `whonix-...`, it's a Whonix specific package. In that case, your safest bet should be pressing `y`, but then you would lose your customized settings. You can re-add them afterwards. Such conflicts will hopefully rarely happen, if you use [Whonix modular flexible .d style configuration folders](#).

### 1.12.34 Hardware troubleshooting

#### Audio doesn't work / Troubleshooting newer hardware

By default, the kernel that is installed in `dom0` comes from the `kernel` package, which is an older Linux LTS kernel. For most cases this works fine since the Linux kernel developers backport fixes to this kernel, but for some newer hardware, you may run into issues. For example, the audio might not work if the sound card is too new for the LTS kernel. To fix this, you can try the `kernel-latest` package – though be aware that it's less tested! (See [here](#) for more information about upgrading kernels in `dom0`). In `dom0`:

```
sudo qubes-dom0-update kernel-latest
```

Reboot when it's done installing. You can double-check that the boot used the newer kernel with `uname -r`, which prints the version of the currently-running kernel. Compare this with the output of `rpm -q kernel`. If the start of `uname -r` matches one of the versions printed by `rpm`, then you're still using the Linux LTS kernel, and you'll probably need to manually fix your boot settings. If `uname -r` reports a higher version number, then you've successfully booted with the kernel shipped by `kernel-latest`.

#### “Unsupported Hardware Detected” error

See [Installation Troubleshooting](#).

#### Keyboard layout settings not behaving correctly

The best approach is to choose the right keyboard layout during the installation process. But if you want to change things afterwards, you can try this workaround.

Assuming XFCE desktop: in `Q` → `System Tools` → `Keyboard` → `Layout`, leave the checkbox “Use system defaults” checked. Do not customize the keyboard layout here.

Set the system-wide layout and options for `xorg` with the `localectl` command in `dom0`. You can use `localectl --help` as a starting point.

Example: `localectl set-x11-keymap us dell ,qwerty compose:caps`.

This generates the appropriate configuration in `/etc/X11/xorg.conf.d/00-keyboard.conf`. This file is auto-generated. Do not edit it by hand, unless you know what you are doing.

Restarting `xorg` is required. The most straightforward way is to reboot the system.

More information in [this discussion](#) and [this GitHub issue](#).

### 1.12.35 UEFI troubleshooting

#### Successfully installed in legacy mode, but had to change some kernel parameters

If you've installed successfully in legacy mode but had to change some kernel parameters for it to work, you should try installing in UEFI mode with the same parameters.

##### Change the xen configuration on a USB media

1. Attach the usb disk, mount the EFI partition (second partition available on the disk)
2. Open a terminal and enter the command `sudo su -`. Use your preferred text editor (e.g nano) to edit your xen config (EFI/BOOT/BOOTX64.cfg):

```
nano EFI/BOOT/BOOTX64.cfg
```

3. Change the `kernel` key to add your kernel parameters on the boot entry of your choice
4. Install using your modified boot entry

**Change xen configuration directly in an iso image** 01. Set up a loop device (replacing X with your ISO's version name): `losetup -P /dev/loop0 Qubes-RX-x86_64.iso` 02. Mount the loop device: `sudo mount /dev/loop0p2 /mnt` 03. Edit EFI/BOOT/BOOTX64.cfg to add your params to the `kernel` configuration key 04. Save your changes, unmount and dd to usb device

#### Installation freezes before displaying installer

If you have an Nvidia card, see also [Nvidia Troubleshooting](#).

#### Removing noexitboot and mapbs

Some systems can freeze with the default UEFI install options. You can try the following to remove `noexitboot` and `mapbs`.

1. Follow the [steps here](#) to edit the `[qubes-verbose]` section of your installer's `BOOTX64.cfg`. You want to comment out the `mapbs` and `noexitboot` lines. The end result should look like this:

```
[qubes-verbose]
options=console=vga efi=attr=uc
# noexitboot=1
# mapbs=1
kernel=vmlinuz inst.stage2=hd:LABEL=Qubes-R4.0-x86_64 i915.alpha_support=1
ramdisk=initrd.img
```

2. Boot the installer and continue to install as normal, but don't reboot the system at the end when prompted.
3. Go to `tty2` (Ctrl-Alt-F2).
4. Use your preferred text editor (nano works) to edit `/mnt/sysimage/boot/efi/EFI/qubes/xen.cfg`, verifying the `noexitboot` and `mapbs` lines are not present. This is also a good time to make permanent any other changes needed to get the installer to work, such as `nouveau.modeset=0`. For example:

```
[4.14.18-1.pvops.qubes.x86_64]
options=loglvl=all dom0_mem=min:1024M dom0_mem=max:4096M iommu=no-igfx ucode=scan_
↪efi=attr=uc
```

5. Go back to `tty6` (Ctrl-Alt-F6) and click Reboot.

6. Continue with setting up default templates and logging in to Qubes.

### Changing `options=console=` parameter to none

If removing `noexitboot` and `mapbs` did not help, you can try changing the `options=console=` parameter to `none`. The detailed solution can be found in the comments of [this GitHub issue](#)

1. Follow the [steps here](#) to edit the `[qubes-verbose]` section of your installer's `BOOTX64.cfg`. You want to change `options=console=vga` to `options=console=none`. The end result should look like this:

```
[qubes-verbose]
options=console=none efi=attr=uc
noexitboot=1
mapbs=1
kernel=vmlinuz inst.stage2=hd:LABEL=Qubes-R4.0-x86_64 i915.alpha_support=1
ramdisk=initrd.img
```

2. Boot the installer and continue to install as normal

### Disable EFI runtime services

On some early, buggy UEFI implementations, you may need to disable EFI under Qubes completely. This can sometimes be done by switching to legacy mode in your BIOS/UEFI configuration. If that's not an option there, or legacy mode does not work either, you can try the following to add `efi=no-rs`. Consider this approach as a last resort, because it will make every Xen update a manual process.

1. Follow the [steps here](#) to edit the `[qubes-verbose]` section of your installer's `xen.cfg`. You want to modify the `efi=attr=uc` setting and comment out the `mapbs` and `noexitboot` lines. The end result should look like this:

```
[qubes-verbose]
options=console=vga efi=no-rs
# noexitboot=1
# mapbs=1
kernel=vmlinuz inst.stage2=hd:LABEL=Qubes-R4.0-x86_64 i915.alpha_support=1
ramdisk=initrd.img
```

2. Boot the installer and continue to install as normal, until towards the end when you will receive a warning about being unable to create the EFI boot entry. Click continue, but don't reboot the system at the end when prompted.
3. Go to `tty2` (Ctrl-Alt-F2).
4. Use your preferred text editor (nano works) to edit `/mnt/sysimage/boot/efi/EFI/qubes/xen.cfg`, adding the `efi=no-rs` option to the end of the `options=` line. For example:

```
[4.14.18-1.pvops.qubes.x86_64]
options=loglvl=all dom0_mem=min:1024M dom0_mem=max:4096M iommu=no-igfx ucode=scan_
↪efi=no-rs
```

5. Execute the following commands:

```
cp -R /mnt/sysimage/boot/efi/EFI/qubes /mnt/sysimage/boot/efi/EFI/BOOT
mv /mnt/sysimage/boot/efi/EFI/BOOT/xen-*.efi /mnt/sysimage/boot/efi/EFI/BOOT/
↪BOOTX64.efi
mv /mnt/sysimage/boot/efi/EFI/BOOT/xen.cfg /mnt/sysimage/boot/efi/EFI/BOOT/BOOTX64.
↪cfg
```

6. Go back to `tty6` (Ctrl-Alt-F6) and click Reboot.
7. Continue with setting up default templates and logging in to Qubes.

Whenever there is a kernel or Xen update for Qubes, you will need to follow [these steps](#) because your system is using the fallback UEFI bootloader in `[...]/EFI/BOOT` instead of directly booting to the Qubes entry under `[...]/EFI/qubes`.

### Installation from USB stick hangs on black screen

Some laptops cannot read from an external boot device larger than 8GB. If you encounter a black screen when performing an installation from a USB stick, ensure you are using a USB drive less than 8GB, or a partition on that USB lesser than 8GB and of format FAT32.

### Installation completes successfully but then boot loops or hangs on black screen

There is a [common bug in UEFI implementation](#) affecting mostly Lenovo systems, but probably some others too. While some systems need `mapbs` and/or `noexitboot` disabled to boot, others require them enabled at all times. Although these are enabled by default in the installer, they are disabled after the first stage of a successful install. You can re-enable them either as part of the install process:

1. Perform installation normally, but don't reboot the system at the end yet.
2. Go to `tty2` (Ctrl-Alt-F2).
3. Enable `mapbs` and/or `noexitboot` on the just installed system. Edit `/mnt/sysimage/boot/efi/EFI/qubes/xen.cfg` (you can use `vi` or `nano` editor) and add to every kernel section:

```
mapbs=1
noexitboot=1
```

**Note:** You must add these parameters on two separate new lines (one parameter on each line) at the end of each section that includes a kernel line (i.e., all sections except the first one, since it doesn't have a kernel line).

4. Go back to `tty6` (Ctrl-Alt-F6) and click Reboot.
5. Continue with setting up default templates and logging in to Qubes.

Or if you have already rebooted after the first stage install and have encountered this issue, by:

1. Boot into *rescue mode*.
2. Enable `mapbs` and/or `noexitboot` on the just installed system. Edit `/mnt/sysimage/boot/efi/EFI/qubes/xen.cfg` (you can use `vi` or `nano` editor) and add to every kernel section:

```
mapbs=1
noexitboot=1
```

**Note:** You must add these parameters on two separate new lines (one parameter on each line) at the end of each section that includes a kernel line (i.e., all sections except the first one, since it doesn't have a kernel line).

3. Type `reboot`.

4. Continue with setting up default templates and logging in to Qubes.

### Installation completes successfully but then system crash/restarts on next boot

Some Dell systems and probably others have [another bug in UEFI firmware](#). These systems need `efi=attr=uc` enabled at all times. Although this is enabled by default in the installer, it is disabled after the first stage of a successful install. You can re-enable it either as part of the install process:

1. Perform installation normally, but don't reboot the system at the end yet.
2. Go to `tty2` (Ctrl-Alt-F2).
3. Execute:

```
sed -i -e 's/^options=.*\/\0 efi=attr=uc/' /mnt/sysimage/boot/efi/EFI/qubes/xen.cfg
```

4. Go back to `tty6` (Ctrl-Alt-F6) and click Reboot.
5. Continue with setting up default templates and logging in to Qubes.

Or if you have already rebooted after the first stage install and have encountered this issue, by:

1. Boot into *rescue mode*.
2. Execute:

```
sed -i -e 's/^options=.*\/\0 efi=attr=uc/' /mnt/sysimage/boot/efi/EFI/qubes/xen.cfg
```

3. Type `reboot`.
4. Continue with setting up default templates and logging in to Qubes.

### Boot device not recognized after installing

Some firmware will not recognize the default Qubes EFI configuration. As such, it will have to be manually edited to be bootable. This will need to be done after every kernel and Xen update to ensure you use the most recently installed versions.

1. Copy the `/boot/efi/EFI/qubes/` directory to `/boot/efi/EFI/BOOT/` (the contents of `/boot/efi/EFI/BOOT` should be identical to `/boot/efi/EFI/qubes` besides what is described in steps 2 and 3):

```
cp -r /boot/efi/EFI/qubes/. /boot/efi/EFI/BOOT
```

2. Rename `/boot/efi/EFI/BOOT/xen.cfg` to `/boot/efi/EFI/BOOT/BOOTX64.cfg`:

```
mv /boot/efi/EFI/BOOT/xen.cfg /boot/efi/EFI/BOOT/BOOTX64.cfg
```

3. Copy `/boot/efi/EFI/qubes/xen-*.efi` to `/boot/efi/EFI/qubes/xen.efi` and `/boot/efi/EFI/BOOT/BOOTX64.efi`. For example, with Xen 4.8.3 (you may need to confirm file overwrite):

```
cp /boot/efi/EFI/qubes/xen-4.8.3.efi /boot/efi/EFI/qubes/xen.efi
cp /boot/efi/EFI/qubes/xen-4.8.3.efi /boot/efi/EFI/BOOT/BOOTX64.efi
```

## Installation finished but “Qubes” boot option is missing and xen.cfg is empty / Installation fails with “failed to set new efi boot target”

In some cases installer fails to finish EFI setup and leave the system without a Qubes-specific EFI configuration. In such a case you need to finish those parts manually. You can do that just after installation (switch to `ttty2` with `Ctrl+Alt+F2`), or by booting from installation media in *rescue mode*.

1. Examine `/boot/efi/EFI/qubes` (if using Qubes installation media, it's in `/mnt/sysimage/boot/efi/EFI/qubes`). You should see 4 files there:
  - `xen.cfg` (empty, size 0)
  - `xen-(xen-version).efi`
  - `vmlinuz-(kernel-version)`
  - `initramfs-(kernel-version).img`
2. Copy `xen-(xen-version).efi` to `xen.efi`:

```
cd /mnt/sysimage/boot/efi/EFI/qubes
cp xen-*.efi xen.efi
```

3. Create `xen.cfg` with this content (adjust kernel version, and filesystem locations, below values are based on default installation of Qubes 3.2):

```
[global]
default=4.4.14-11.pvops.qubes.x86_64

[4.4.14-11.pvops.qubes.x86_64]
options=loglvl=all dom0_mem=min:1024M dom0_mem=max:4096M
kernel=vmlinuz-4.4.14-11.pvops.qubes.x86_64 root=/dev/mapper/qubes_dom0-root rd.lvm.
↪lv=qubes_dom0/root rd.lvm.lv=qubes_dom0/swap i915.preliminary_hw_support=1 rhgb
↪quiet
ramdisk=initramfs-4.4.14-11.pvops.qubes.x86_64.img
```

4. Create boot entry in EFI firmware (replace `/dev/sda` with your disk name and `-p 1` with `/boot/efi` partition number):

```
efibootmgr -v -c -u -L Qubes -l /EFI/qubes/xen.efi -d /dev/sda -p 1 "placeholder /
↪mapbs /noexitboot"
```

## Accessing installer Rescue mode on UEFI

In UEFI mode, the installer does not have a boot menu, but boots directly into the installation wizard. To get into Rescue mode, you need to switch to `ttty2` (`Ctrl+Alt+F2`) and then execute:

```
kill -9 anaconda
anaconda --rescue
```



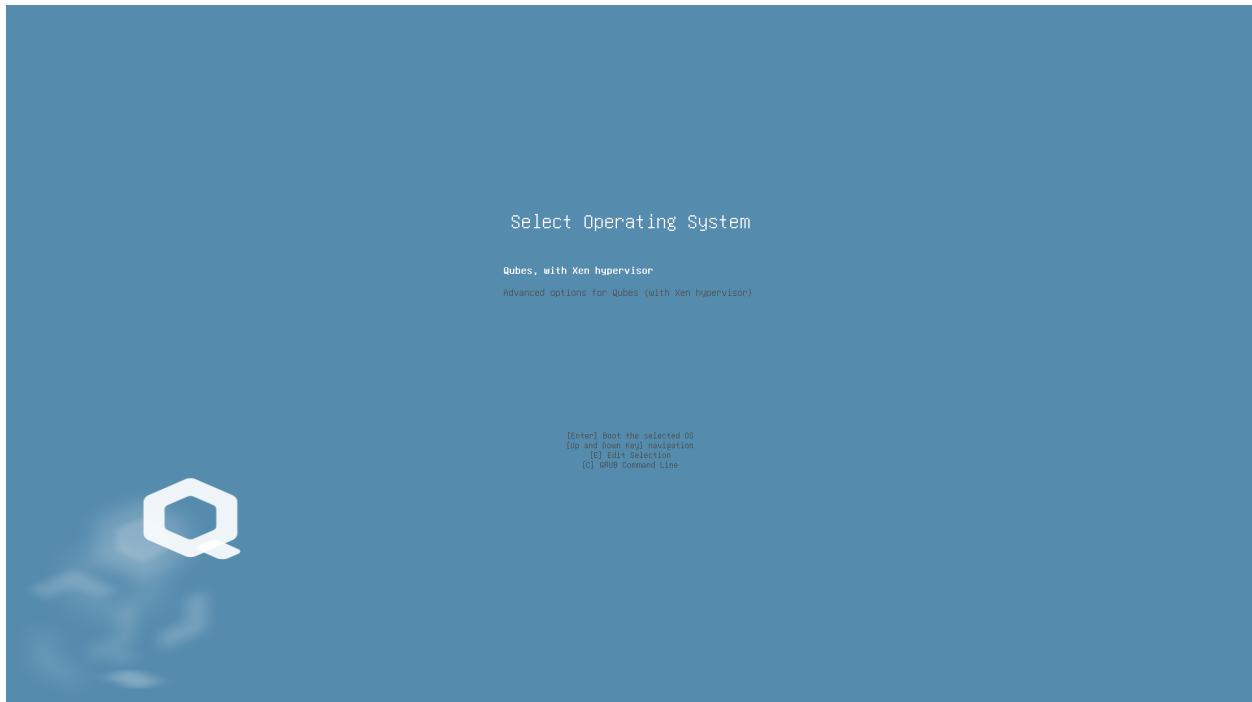
### 1.12.36 Autostart troubleshooting

The following instructions are valid for **Qubes OS R4.0 legacy mode** and **Qubes OS R4.1 legacy and UEFI modes**. For **Qubes OS R4.0 in UEFI mode**, there is no GRUB, so manual boot from another operating system is needed.

In several cases, there is a need to prevent `autostart=True` for qubes on boot. For example:

- `sys-usb` was enabled, but the only keyboard is attached via USB, and the `qubes.InputKeyboard` service is disabled.
- A PCI device assigned to an autostarting qube crashes the system (e.g., a GPU or RAID controller card).

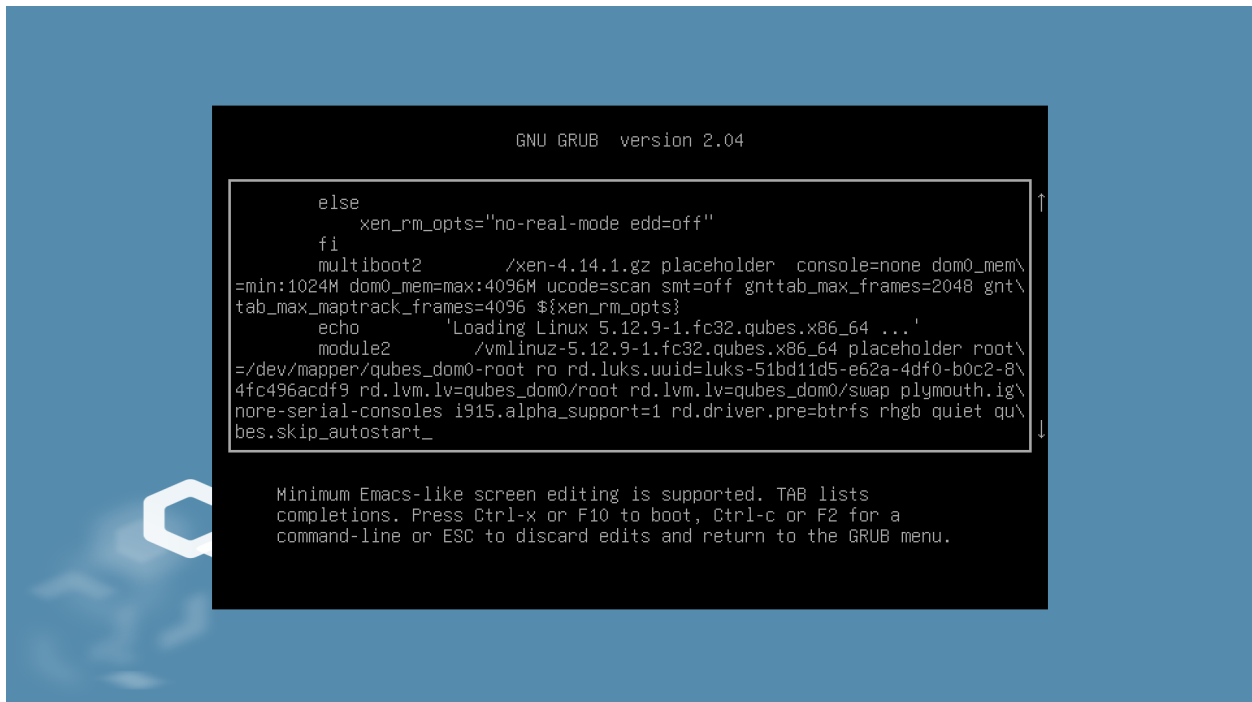
To address this, there is a `qubes.skip_autostart` option for the kernel command line. You can use it at the grub boot menu.



Press the E key on the first prompt (or your custom prompt). Then, press the down arrow key multiple times to reach the line starting with `module2`.



Append `qubes.skip_autostart` to the end of this line (generally after the `rhgb quiet` options).



Press **Ctrl+X** to boot with the edited GRUB entry. The boot will proceed as usual from here, except that no qube will be autostarted.

### 1.12.37 Suspend/resume troubleshooting

First check if there are any remarks for similar devices in the Hardware Compatibility List (HCL).

#### Network-Manager says “Device not ready” on suspend/resume

These instructions may help with suspend/resume issues for more devices than just wireless cards, that is just the (unfortunately not uncommon) example used here.

If your wireless card works, but after suspending and resuming your computer, the Network-Manager applet just says “Device not ready”, then try un-loading and re-loading the driver.

#### Determining your wireless card driver

First, determine which kernel module corresponds to your wireless card. There are several ways to do this.

The easiest is via the output of `lspci -k` in your sys-net VM:

```
[user@sys-net ~]$ lspci -k
00:00.0 Network controller: Intel Corporation Wireless 8260 (rev 3a)
    Subsystem: Intel Corporation Device 0130
    Kernel driver in use: iwlwifi
    Kernel modules: iwlwifi
```

Here we see that the machine in question has an Intel wireless card, being used by the `iwlwifi` kernel module.

#### Checking logs for relevant messages

View the output of `dmesg` in sys-net, and check if you see a bunch of wireless related errors. Depending on your hardware, they may look like the following (or not):

```
iwlwifi 0000:00:00.0: loaded firmware version 16.242414.0 op_mode iwlvm
iwlwifi 0000:00:00.0: Detected Intel(R) Dual Band Wireless AC 8260, REV=0x208
...
IPv6: ADDRCONF(NETDEV_UP): wlp0s0: link is not ready
iwlwifi 0000:00:00.0: L1 Enabled - LTR Enabled
iwlwifi 0000:00:00.0: L1 Enabled - LTR Enabled
iwlwifi 0000:00:00.0: Failed to load firmware chunk!
iwlwifi 0000:00:00.0: Could not load the [0] uCode section
iwlwifi 0000:00:00.0: Failed to start INIT ucode: -110
iwlwifi 0000:00:00.0: Failed to run INIT ucode: -110
...
iwlwifi 0000:00:00.0: Direct firmware load for iwlwifi-8000C-18.ucode failed with error -
↪2
```

## Seeing what modules you have loaded

You can check which drivers are currently loaded with `lsmod`, and view details about a module with `modinfo <module_name>`.

For example, we list what modules we have loaded:

```
[user@sys-net ~]$ lsmod
Module                Size  Used by
iwlvmv                315392  0
iwlwifi               155648  1 iwlvmv
mac80211              708608  1 iwlvmv
cfg80211              557056  3 iwlwifi,mac80211,iwlvmv
...
```

and check one:

```
[user@sys-net ~]$ modinfo iwlvmv | grep -E '^(description|author|depends):'
author:      Copyright(c) 2003- 2015 Intel Corporation <ilw@linux.intel.com>
description: The new Intel(R) wireless AGN driver for Linux
depends:      iwlwifi,mac80211,cfg80211
```

Hey, it's our wireless driver!

Now, check if reloading the module makes wireless work again:

```
[user@sys-net ~]$ sudo rmmod iwlvmv
[user@sys-net ~]$ sudo modprobe iwlvmv
```

and try reconnecting to a network that is known to work.

If that is successful, see below about having Qubes automatically reload the driver for you. If not, try also reloading some dependent modules, in our example we must also reload `iwlwifi`:

```
[user@sys-net ~]$ modinfo iwlwifi | grep -E '^(description|author|depends):'
author:      Copyright(c) 2003- 2015 Intel Corporation <ilw@linux.intel.com>
description: Intel(R) Wireless WiFi driver for Linux
depends:      cfg80211
```

```
[user@sys-net ~]$ sudo rmmod iwlvmv
[user@sys-net ~]$ sudo rmmod iwlwifi
[user@sys-net ~]$ sudo modprobe iwlwifi # note the reverse order of loading/unloading
[user@sys-net ~]$ sudo modprobe iwlvmv
```

## Drivers do not reload automatically on suspend/resume

If reloading the driver (which resets the hardware into a known-state) resolves your issue when done manually, you can have Qubes automatically un/reload them on suspend & resume by listing the relevant modules in `/rw/config/suspend-module-blacklist`.

In the above example, it would look like this:

```
[user@sys-net config]$ cat /rw/config/suspend-module-blacklist
# You can list here modules you want to be unloaded before going to sleep. This
```

(continues on next page)

(continued from previous page)

```
# file is used only if the VM has any PCI device assigned. Modules will be
# automatically loaded after resume.
iwlvm
iwlwifi
```

## Power consumption increases on suspend/resume

This problem is related to the software method used to disable sibling threads and how it interacts with suspend/resume. To solve the problem, disable hyper-threading in the BIOS. This [external guide](#) explains how to disable hyper-threading. Since Qubes does disable hyperthreading by default (by not using secondary threads), you won't pay any performance cost.

## Attached devices in Windows HVM stop working on suspend/resume

After the whole system gets suspended into S3 sleep and subsequently resumed, some attached devices may stop working. To make the devices work, they should be restarted within the VM. This can be achieved under a Windows HVM by opening the Device Manager, selecting the actual device (such as a USB controller), 'Disabling' the device, and then 'Enabling' the device again. This is illustrated on the screenshot below:

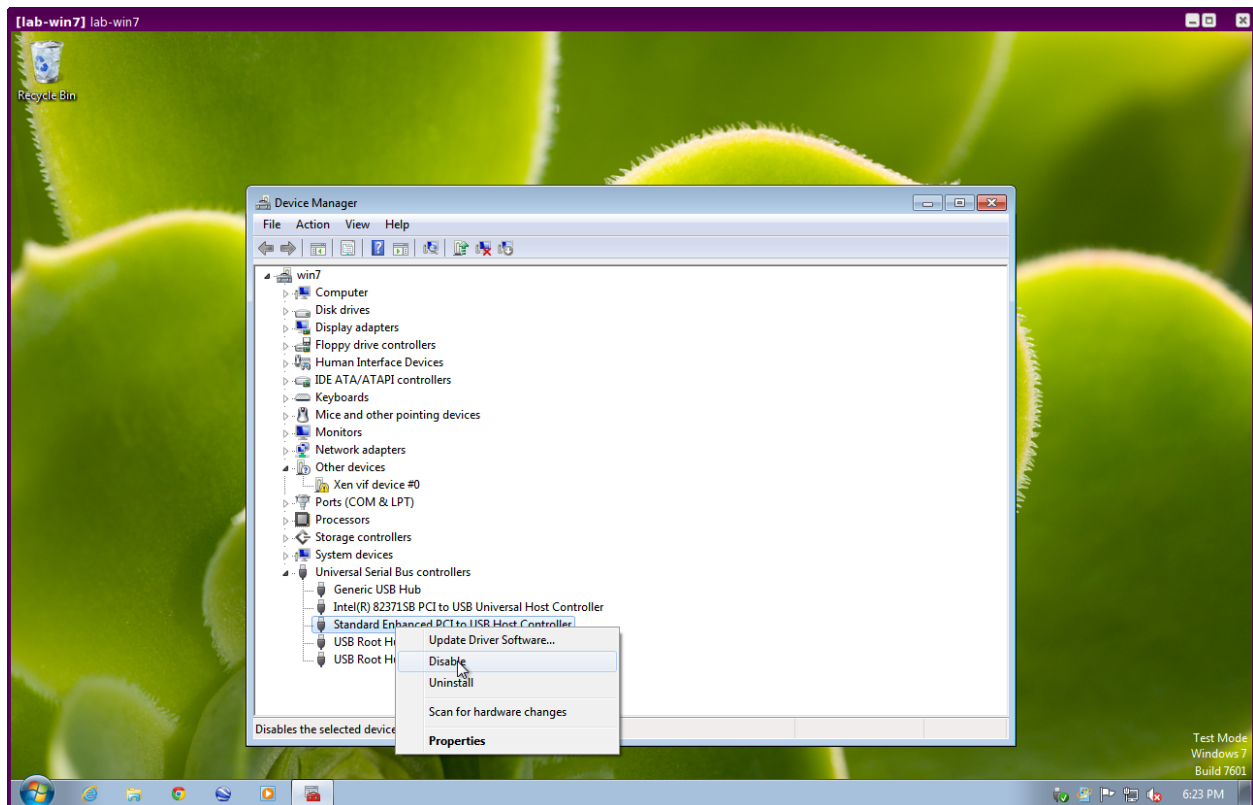


Fig. 6: r2b1-win7-usb-disable.png

## Suspend turns off the screen and gets stuck

On some devices suspend-to-RAM does not work and a hard power-off is needed to recover, because the system does not go into deep sleep. To get suspend to work, you need to add `mem_sleep_default=deep` kernel option. For legacy boot, or UEFI/legacy in R4.1+, add it to the `GRUB_CMDLINE_LINUX` setting in `/etc/default/grub`, update the grub config, and reboot. In R4.0 with UEFI boot, you need to add it to every `kernel=` line in `/boot/efi/EFI/qubes/xen.cfg` and reboot.

### 1.12.38 App menu shortcut troubleshooting

For ease of use Qubes aggregates shortcuts to applications that are installed in app qubes and shows them in one application menu (aka “app menu” or “start menu”) in dom0. Clicking on such shortcut runs the assigned application in its app qube.

To make applications newly installed via the OS’s package manager show up in the menu, use the `qvm-sync-appmenus` command (Linux VMs do this automatically):

```
qvm-sync-appmenus vmname
```

After that, select the *Add more shortcuts* entry in the VM’s submenu to customize which applications are shown:

The above image shows that Windows HVMs are also supported (provided that Qubes Tools are installed).

### What if my application has not been automatically included in the list of available apps?

Some times applications may not have included a `.desktop` file and may not be detected by `qvm-sync-appmenus`. Other times, you may want to make a web shortcut available from the Qubes start menu.

You can manually create new entries in the “available applications” list of shortcuts for all app qubes based on a template. To do this:

1. Open a terminal window to the template.
2. Create a custom `.desktop` file in `/usr/share/applications` (you may need to first create the subdirectory). Look in `/usr/share/applications` for existing examples, or see the full [file specification](#). It will be something like:

```
[Desktop Entry]
Version=1.0
Type=Application
Terminal=false
Icon=/usr/share/icons/Adwaita/256x256/devices/scanner.png
Name=VueScan
GenericName=Scanner
Comment=Scan Documents
Categories=Office;Scanning;
Exec=vuescan
```

3. In dom0, run `qvm-sync-appmenus <templateName>`.
4. Go to VM Settings of the app qube(s) to which you want to add the new shortcut, then the Applications tab. Move the newly created shortcut to the right under selected.

If you only want to create a shortcut for a single app qube, you can create a custom menu entry instead:

1. Open a terminal window to Dom0.

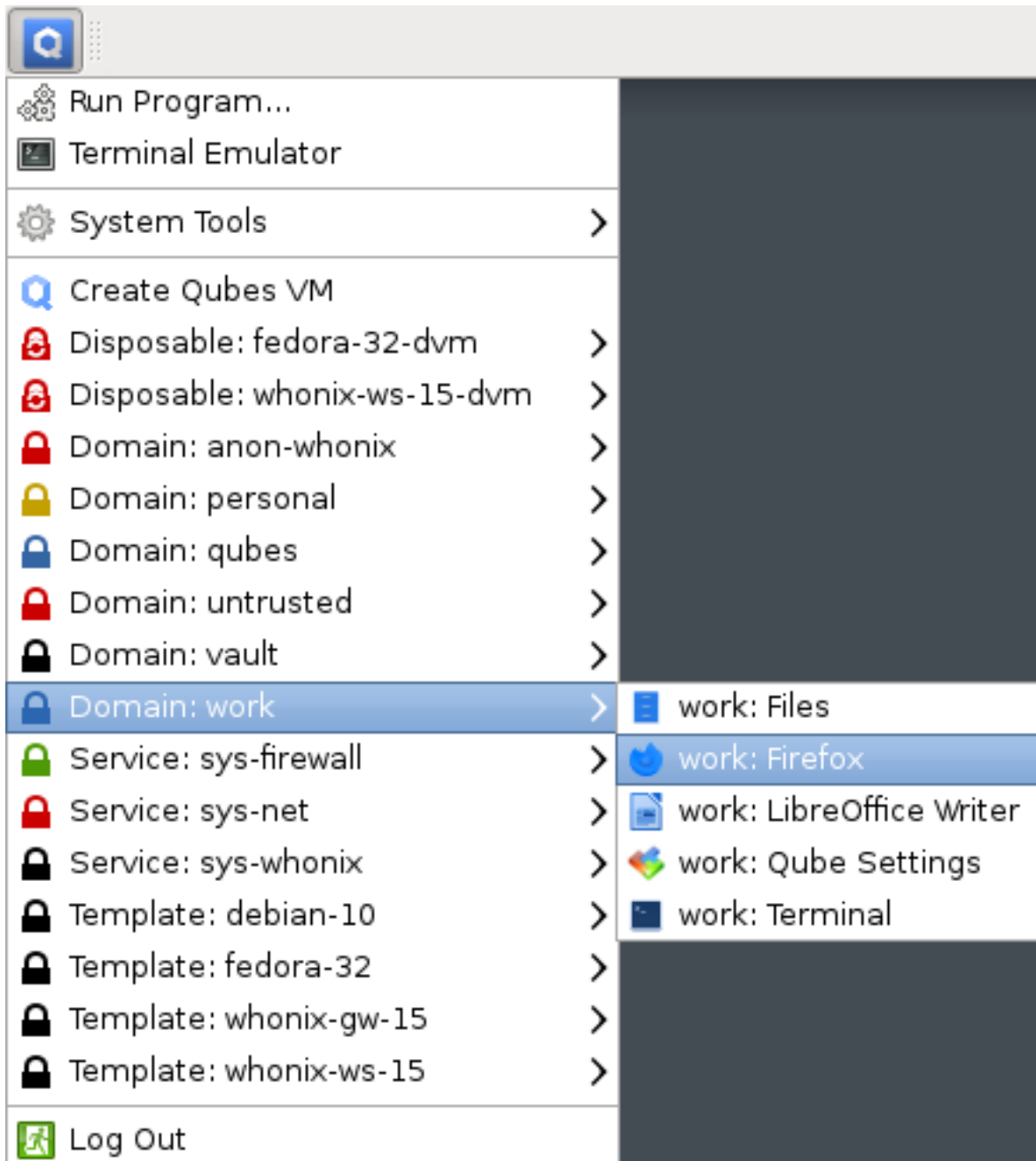


Fig. 7: dom0-menu.png

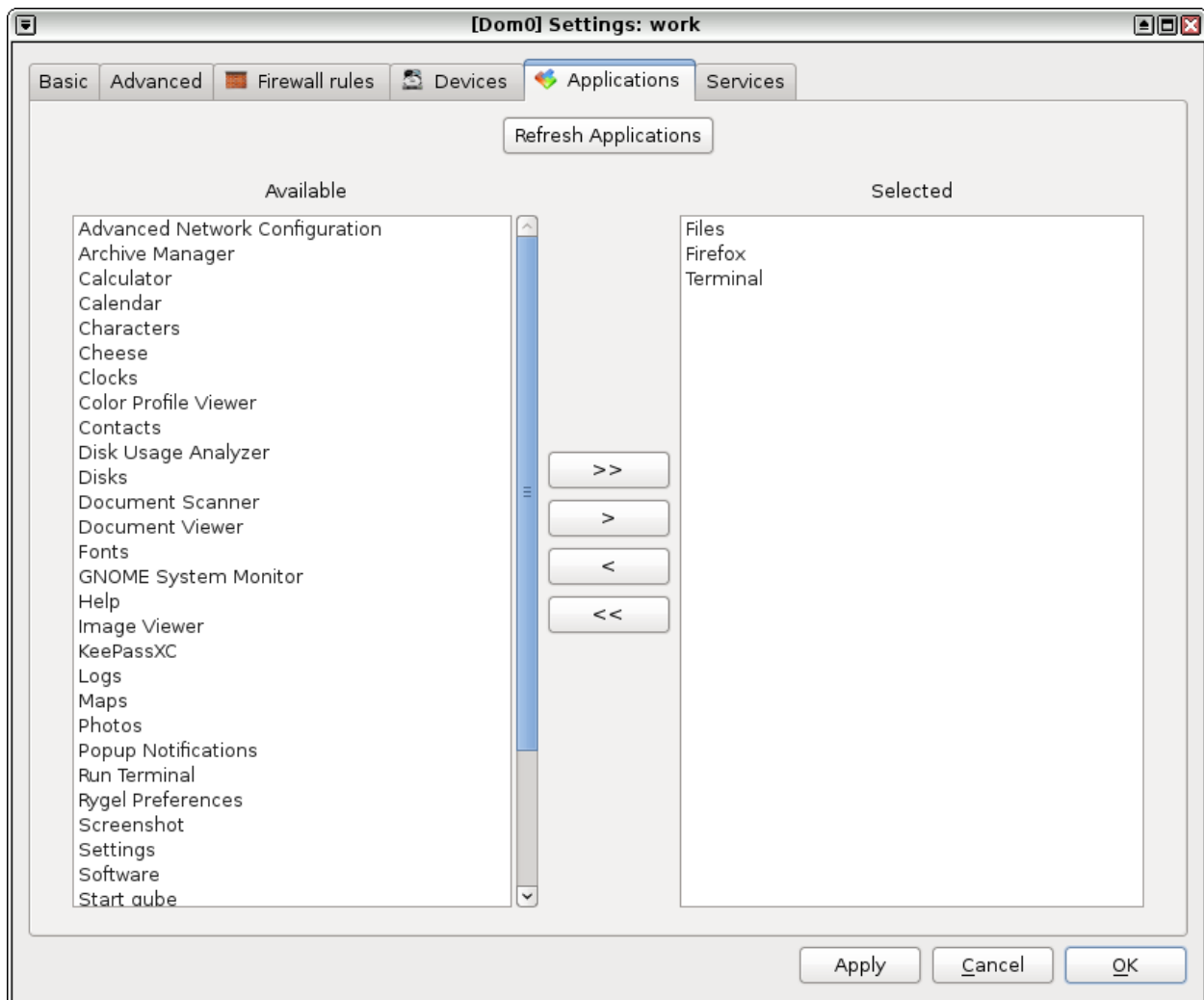


Fig. 8: dom0-appmenu-select.png



2. Create a custom `.desktop` file in `~/.local/share/applications`. Look in the same directory for existing examples, or see the full [file specification](#). You may use `qvm-run` inside the `.desktop` file; see [Behind the scenes](#) for more details.
3. Edit the `~/.config/menus/applications-merged/<vmname>-vm.menu` file for the app qube.
4. Add a custom menu entry referring to your newly created `.desktop` file.

```
<Menu>
  <Name>Webmail</Name>
  <Include>
    <Filename>custom.desktop</Filename>
  </Include>
</Menu>
```

## What about applications in DispVMs?

*See [here](#).*

## Fixing shortcuts

First, try this in dom0:

```
$ qvm-appmenus --update --force <vm_name>
```

If that doesn't work, you can manually modify the files in `~/.local/share/applications/` or `/usr/local/share/applications/`.

For example, suppose you've deleted `my-old-vm`, but there is a leftover Application Menu shortcut, and you find a related file in `~/.local/share/applications/`. In dom0:

```
$ rm -i ~/.local/share/applications/my-old-vm-*
```

## Behind the scenes

`qvm-sync-appmenus` works by invoking `GetAppMenus` [Qubes service](#) in the target domain. This service enumerates installed applications and sends formatted info back to the dom0 script (`/usr/libexec/qubes-appmenus/qubes-receive-appmenus`) which creates `.desktop` files in the app qube/template directory.

For Linux VMs the service script is in `/etc/qubes-rpc/qubes.GetAppMenus`. In Windows it's a PowerShell script located in `c:\Program Files\Invisible Things Lab\Qubes OS Windows Tools\qubes-rpc-services\get-appmenus.ps1` by default.

The list of installed applications for each app qube is stored in dom0's `~/.local/share/qubes-appmenus/<vmname>/apps.templates`. Each menu entry is a file that follows the [.desktop file format](#) with some wildcards (`%VMNAME%`, `%VMDIR%`). Applications selected to appear in the menu are stored in `~/.local/share/qubes-appmenus/<vmname>/apps`.

Actual command lines for the menu shortcuts involve `qvm-run` command which starts a process in another domain. Examples: `qvm-run -q -a --service -- %VMNAME% qubes.StartApp+7-Zip-7-Zip_File_Manager` or `qvm-run -q -a --service -- %VMNAME% qubes.StartApp+firefox`

Note that you can create a shortcut that points to a `.desktop` file in your app qube with e.g. `qvm-run -q -a --service -- personal qubes.StartApp+firefox`.

## 1.12.39 Qube troubleshooting

### VM Kernel troubleshooting

This troubleshoot applies to the non-default kernel choice described in the *Managing VM docs*.

In case of problems, you can access the VM console using `qvm-console-dispvm VMNAME` in dom0, then access the GRUB menu. You need to call it just after starting the VM (until `GRUB_TIMEOUT` expires); for example, in a separate dom0 terminal window.

In any case you can later access the VM's logs (especially the VM console log `/var/log/xen/console/guest-VMNAME.log`).

You can always set the kernel back to some dom0-provided value to fix a VM kernel installation.

### Qubes starts, but no VMs load

First, try to start a particular VM, check any failure message and direct further steps based on that.

This issue has been seen to occur if a dom0 update is interrupted halfway through and/or a hard power off is done without shutting down Qubes, which results in files getting corrupted.

### Can not uninstall a VM / “ERROR: VM installed by package manager: template-vm-name”

All of the following commands should be executed in a dom0 terminal.

When a template is marked as ‘installed by package manager’, but cannot be uninstalled there, trying to uninstall manually will result in the error “ERROR: VM installed by package manager: template-vm-name”. Do as follows to be able to uninstall the template:

1. Check the state of `installed_by_rpm`

```
$ qvm-prefs template-vm-name
```

2. If `installed_by_rpm - True`, mark the template as not installed by package manager

```
$ qvm-prefs template-vm-name installed_by_rpm false
```

3. Re-check the state of `installed_by_rpm`

- If `installed_by_rpm - False`, remove the template like you would a regular qube:

```
$ qvm-remove template-vm-name
```

- If `installed_by_rpm` remains `True`, reboot your computer to bring `qubes.xml` in sync with `qubesd`, and try again to remove the template.

## Fixing package installation errors

By default, templates in 4.0 only have a loopback interface.

Some packages will throw an error on installation in this situation. For example, Samba expects to be configured using a network interface post installation.

One solution is to add a dummy interface to allow the package to install correctly:

```
ip link add d0 type dummy
ip addr add 192.168.0.1/24 dev d0
ip link set d0 up
```

### “Cannot connect to qrexec agent” error

If you face this error when starting a VM, look into the VM logs at `/var/log/xen/console/guest-VMNAME.log`. Common reasons that may be revealed are: too low memory, corrupted files or a VM crash on startup.

If the error occurs as a result of too little initial memory, increase the initial memory from 200MB to 400MB by navigating to VM settings » Advanced » Initial memory.

### “No match found” when trying to install a template

For example:

```
[user@dom0 ~]$ sudo qubes-dom0-update --enablerepo=qubes-templates-itl qubes-template-
↪debian-10
Using sys-whonix as UpdateVM to download updates for Dom0; this may take some time...
No Match for argument qubes-template-debian-10
Nothing to download
```

This normally means you already have the template installed. It may be that you have the matching package installed, but you removed or renamed the template. Check `rpm -q qubes-template-<name>`. If it lists the package, but you don't really have the template present (`qvm-ls` doesn't list it), you need to clean up leftovers of the package with `rpm -e --noscripts qubes-template-<name>`, then install it normally.

## 1.12.40 HVM troubleshooting

### HVM pauses on boot, followed by kernel error

The HVM may pause on boot, showing a fixed cursor. After a while a series of warnings may be shown similar to this:

```
BUG: soft lockup - CPU#0 stuck for 23s! [systemd-udevd:244]
```

To fix this:

1. Kill the HVM.
2. Start the HVM
3. Press “e” at the grub screen to edit the boot parameters
4. Find the `vmlinuz` line, and edit it to replace “`rhgb`” with “`modprobe.blacklist=bochs_drm`”
5. Press “Ctrl-x” to start the HVM

If this solves the problem then you will want to make the change permanent:

1. Edit the file `/etc/default/grub`.
2. Find the line which starts:

```
GRUB_CMDLINE_LINUX=
```

3. Remove this text from that line:

```
rhgb
```

4. Add this text to that line:

```
modprobe.blacklist=bochs_drm
```

5. Run this command:

```
grub2-mkconfig --output=/boot/grub2/grub.cfg
```

The HVM should now start normally.

### Can't start an OS in an HVM / “Probing EDD (edd=off to disable!... ok” message

If you see a screen popup with SeaBios and 4 lines, last one being Probing EDD (edd=off to disable!... ok, then enter the following command from a `dom0` prompt:

```
qvm-prefs <HVMname> kernel ""
```

### HVM crashes when booting from ISO

If your HVM crashes when trying to boot an ISO, first ensure that `qvm-prefs <HVMname> kernel` is empty, as shown above. If this doesn't help, then disable memory balancing and set the minimum memory to 2GB.

You can disable memory-balancing in the settings, under the “Advanced” tab.

To give the VM a RAM of 2GB, open a terminal in `dom0` and enter:

```
qvm-prefs <HVMname> memory 2000
```

### Attached devices in Windows HVM stop working on suspend/resume

After the whole system gets suspended into S3 sleep and subsequently resumed, some attached devices may stop working. To know how to make the devices work, see [Suspend/resume Troubleshooting](#).

## 1.12.41 Disk troubleshooting

### “Out of disk space” error

If the disk is completely full, you will get an `Out of disk space` error that may crash your system because Dom0 does not have enough disk space to work. So it's good practice to regularly check disk space usage. Running the `df -h` command in dom0 terminal will show some information, but not include all the relevant details. The Qubes user interface provides a disk space widget. If you are unable to access the interface, the command line version is running `sudo lvs | head` and looking at top entry for LVM pool. For example:

LV	VG	Attr	LSize	Pool	
↪ Origin		Data% Meta% Move Log Cpy%Sync Convert			
pool00	qubes_dom0	twi-aotz--	453.17g		↪
↪	89.95 69.78				
root	qubes_dom0	Vwi-aotz--	453.17g	pool00	↪
↪	5.87				
swap	qubes_dom0	-wi-ao----	7.57g		

If you run `df -h`, it only shows the information in the `root` line (which is already included in the `pool00` line). As you can see, the `sudo lvs | head` command includes additional important columns `Data%` and `Meta%`, shown in the above example to have the values 89% and 69% respectively.

If your system is able to boot, but cannot load a desktop environment, it is possible to login to dom0 terminal with `Alt + Ctrl + F2`.

If this does not work, check the size of `/var/lib/qubes/qubes.xml`. If it is zero, you'll need to use one of the file backup (stored in `/var/lib/qubes/backup`), hopefully you have the current data there. Find the most recent one and place in `/var/lib/qubes/qubes.xml` instead of the empty file.

In any case you'll need some disk space to start the VM. Check `df -h` output if you have some. If not, here are some hints how to free some disk space:

1. Clean yum cache.

```
sudo dnf clean all
```

2. Delete `.img` files of a less important VM, which can be found in `/var/lib/qubes/appvms/`. Then, when the system is working again, clean up the rest.

```
qvm-remove <VMname>
```

With this method, you lose the data of one VM, but it'll work more reliably.

3. Decrease the filesystem safety margin (5% by default).

```
sudo tune2fs -m 4 /dev/mapper/vg_dom0-lv_root
```

4. Remove some unneeded files in dom0 home (if you have any, most likely not). Also look for unneeded files in `/var/log` in dom0, and `/var/log/qubes`.

The above steps applies to old VM disks format. These steps may work on Qubes 4.0, but are not default anymore. By default, Qubes 4.0 now uses LVM. The equivalent steps are:

1. Get a list of VM disks using `sudo lvs`.
2. Use `sudo lvremove qubes_dom0/<name>` to remove backup copies of some less important VMs – entries with `-back` in their name.

3. If that isn't enough, remove actual disks of less important VMs. NOTE: You will lose the data of that VM, but your system will resume working.

For example:

```
$ sudo lvs
LV                               VG      Attr      LSize   Pool   ↵
↪Origin                          Data%  Meta%  Move Log Cpy%Sync Convert
pool00                          qubes_dom0 twi-aotz-- 453.17g ↵
↪                               89.95  69.78
root                            qubes_dom0 Vwi-aotz-- 453.17g pool00 ↵
↪                               5.87
swap                            qubes_dom0 -wi-ao---- 7.57g
(...)
vm-d10test-private              qubes_dom0 Vwi-a-tz-- 2.00g pool00 vm-
↪d10test-private-1600961860-back 29.27
vm-d10test-private-1600961860-back qubes_dom0 Vwi-a-tz-- 2.00g pool00 ↵
↪                               4.87
vm-d10test-standalone-private   qubes_dom0 Vwi-a-tz-- 2.00g pool00 vm-
↪d10test-standalone-private-1580772439-back 4.90
vm-d10test-standalone-private-1580772439-back qubes_dom0 Vwi-a-tz-- 2.00g pool00 ↵
↪                               4.87
vm-d10test-standalone-root      qubes_dom0 Vwi-a-tz-- 10.00g pool00 vm-
↪d10test-standalone-root-1580772439-back 43.37
vm-d10test-standalone-root-1580772439-back qubes_dom0 Vwi-a-tz-- 10.00g pool00 ↵
↪                               42.05
vm-debian-10-my-private         qubes_dom0 Vwi-a-tz-- 2.00g pool00 ↵
↪                               4.96
vm-debian-10-my-root            qubes_dom0 Vwi-a-tz-- 10.00g pool00 vm-
↪debian-10-my-root-1565013689-back 57.99
vm-debian-10-my-root-1565013689-back qubes_dom0 Vwi-a-tz-- 10.00g pool00 ↵
↪                               56.55
vm-debian-10-private            qubes_dom0 Vwi-a-tz-- 2.00g pool00 ↵
↪                               4.94
vm-debian-10-root               qubes_dom0 Vwi-a-tz-- 10.00g pool00 vm-
↪debian-10-root-1601126126-back 93.44
vm-debian-10-root-1601126126-back qubes_dom0 Vwi-a-tz-- 10.00g pool00 ↵
↪                               88.75
(...)
$ sudo lvremove qubes_dom0/vm-d10test-standalone-root-1580772439-back
Do you really want to remove and DISCARD active logical volume qubes_dom0/vm-d10test-
↪standalone-root-1580772439-back? [y/n]: y
Logical volume "vm-d10test-standalone-root-1580772439-back" successfully removed
```

After freeing some initial space, it may be possible to recover more space by deleting files in a userVM after connecting to the userVM terminal:

```
qvm-start <VMname>
qvm-console-dispvm <VMname>
```

Since `qvm-console-dispvm` requires working graphical user interface login, you must first free enough space to be able to start a VM and login to graphical UI.

## Can't resize VM storage / “resize2fs: Permission denied” error

*Resizing a volume* in the Qubes interface should be a straightforward process. But sometimes, an attempt to resize will look like it worked, when it in fact fails silently. If you then try the same operation in the dom0 console using the `qvm-volume extend` command, it fails with the error message: `resize2fs: Permission denied to resize filesystem`. This error indicates that a `resize2fs` will not work, unless `fsck` is run first. Qubes OS utilities cannot yet handle this case.

To fix this issue:

1. In the dom0 terminal get a root console on the vm (eg. sys-usb) with:

```
qvm-console-dispvm sys-usb
```

2. Unmount everything mounted on the private volume `/dev/xvdb` partition. There are typically several mounts listed in `/etc/mtab`.
3. When you attempt to unmount the `/home` directory using the `umount /home` command, you will encounter an error because there are processes using the `/home` directory. You can view a list of these processes with the `fuser` command:

```
fuser -m /home
```

Kill these process until they are all gone using `kill <process ID>`.

4. Finally, run:

```
umount /home
fsck /dev/xvdb
resize2fs /dev/xvdb
```

After restarting your VM, everything should now work as expected. The private volume size shown externally in the VM's settings interface is the same as that seen within the VM.

## 1.12.42 PCI troubleshooting

### DMA errors

VMs with attached PCI devices in Qubes have allocated a small buffer for DMA operations (called `swiotlb`). By default, it is 2MB, but some devices (such as the [Realtek RTL8111DL Gigabit Ethernet Controller](#)) need a larger DMA buffer size. Without a larger buffer, you will face DMA errors such as `Failed to map TX DMA`.

To change this allocation, edit VM's kernel parameters (this is expressed in 512B chunks) by running the following in a dom0 terminal:

```
# qvm-prefs netvm |grep kernelopts
kernelopts      : iommu=soft swiotlb=2048 (default)
# qvm-prefs -s netvm kernelopts "iommu=soft swiotlb=8192"
```

The 8192 value is the default value and some devices may require a larger value (like 16384).

## PCI Passthrough Issues

Sometimes the PCI arbitrator is too strict, which may cause errors such as `Unable to reset PCI device` and other PCI-related errors. There is a way to enable permissive mode for it. See also: [this thread](#) and the Xen wiki's [PCI passthrough](#) page. Other times, you may instead need to disable the FLR requirement on a device.

Both can be achieved during attachment with `qvm-pci` as described [PCI Devices documentation](#).

### “Unable to reset PCI device” errors

**libvirt.libvirtError: internal error: Unable to reset PCI device [...]: internal error: Active [...] devices on bus with [...], not doing bus reset**

After running `qvm-start sys-net`, you may encounter an error message which begins with `libvirt.libvirtError: internal error: Unable to reset PCI device`.

This issue is likely to occur if you have the same device assigned to more than one VM. When you try to start `sys-net` with the `qvm-start sys-net` command, there is already a VM running (e.g., auto-starting) with one or more of the same devices as those assigned to `sys-net`.

To fix the error, remove the offending PCI device.

### Using the Qubes interface

From the “Selected” panel in `sys-net`, navigate to VM Settings, then Devices. There, you can remove the offending PCI device(s) and keep the desired PCI device.

### Using the command line

1. To see all the PCI available devices, enter the `lspci` command into the `dom0` terminal. Each device will be listed on a line, for example:

```
0000:03:00.0 Audio device: Intel Corporation Haswell-ULT HD Audio Controller (rev. 0b)
```

In the above output, the BDF (Bus Device Function) of the device is `0000:03:00.0`

2. Now that you can see all the PCI devices and their BDFs, you can decide which to remove and which to keep. Imagine we faced the following error message:

```
libvirt.libvirtError: internal error: Unable to reset PCI device 0000:03:00.1: internal error: Active 0000:03:00.0 devices on bus with 0000:03:00.1, not doing bus reset
```

In the above case, the device `0000:03:00.1` is the device which we want to use. But we are facing the `Unable to reset PCI device` error because another device, `0000:03:00.0`, is active. To fix this error and get device `0000:03:00.1` to work, we must first remove the offending device `0000:03:00.0`.

```
sudo su
echo -n "1" > /sys/bus/pci/devices/0000:03:00.0/remove
```

3. In order to make this change persistent, create a file `/etc/systemd/system/qubes-pre-netvm.service` and add the following:



```
[Unit]
Description=Netvm fixup
Before=qubes-netvm.service

[Service]
ExecStart=/bin/sh -c 'echo -n "1" > /sys/bus/pci/devices/0000:03:00.0/remove'
Type=oneshot
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

Finally, run `systemctl enable qubes-pre-netvm.service` and it will now persist between reboots.

### Domain [...] has failed to start: internal error: Unable to reset PCI device [...]: no FLR, PM reset or bus reset available

This is a *PCI passthrough issue*, which occurs when PCI arbitrator is too strict. There is a way to enable permissive mode for it. Sometimes, you may instead need to disable the FLR requirement on a device. Both can be achieved during attachment with `qvm-pci` as described below.

NOTE: The `permissive` flag increases attack surface and possibility of *side channel attacks*. While using the `no-strict-reset` flag, do not require PCI device to be reset before attaching it to another VM. This may leak usage data even without malicious intent. Both `permissive` and `no-strict-reset` options may not be necessary and you should try one first, then the other, before using both.

```
qvm-pci attach --persistent --option permissive=true --option no-strict-reset=true sys-
  ↳usb dom0:<BDF_OF_DEVICE>
```

Be sure to replace `<BDF_OF_DEVICE>` with the BDF of your PCI device, which can be obtained from running `qvm-pci`.

You can also configure strict reset directly from the Qubes interface by following these steps:

1. Go to the sys-net VM settings
2. Go to Devices
3. Make sure the device is in the right field
4. Click “Configure strict reset for PCI devices”
5. Select the device, click OK and apply

### Broadcom BCM43602 Wi-Fi card causes system freeze

You may face the problem where the BCM43602 Wi-Fi chip causes a system freeze whenever it is attached to a VM. To fix this problem on a Macbook, follow the steps in [Macbook Troubleshooting](#).

For other non-Macbook machines, it is advisable to replace the Broadcom BCM43602 with one known to work on Qubes, such as the Atheros AR9462.

Note that your computer manufacturer may have added a Wi-Fi card whitelist in your BIOS, which will prevent booting your computer if you have a non-listed wireless card. It is possible to bypass this limitation by removing the whitelist, disabling a check for it or modifying the whitelist to replace device ID of a whitelisted WiFi card with device ID of your new WiFi card.

### **Wireless card stops working after dom0 update**

There have been many instances where a Wi-Fi card stops working after a dom0 update. If you run `sudo dmesg` in `sys-net`, you may see errors beginning with `iwlwifi`. You can fix the problem by going to the `sys-net` VM's settings and changing the VM kernel to the previous version.

### **Attached devices in Windows HVM stop working on suspend/resume**

After the whole system gets suspended into S3 sleep and subsequently resumed, some attached devices may stop working. Refer to *Suspend/Resume Troubleshooting* for a solution.

### **PCI device not available in dom0 after being unassigned from a qube**

After you assign a PCI device to a qube, then unassign it/shut down the qube, the device is not available in dom0. This is an intended feature. A device which was previously assigned to a less trusted qube could attack dom0 if it were automatically reassigned there. Look at the *FAQs* to learn how to re-enable the device in dom0.

### **Network adapter does not work**

You may have an adapter (wired, wireless), that is not compatible with open-source drivers shipped by Qubes. You may need to install a binary blob, which provides drivers, from the `linux-firmware` package.

Open a terminal and run `sudo dnf install linux-firmware` in the template upon which your NetVM is based. You have to restart the NetVM after the template has been shut down.

## **1.12.43 USB troubleshooting**

### **disp-sys-usb does not start**

If the `disp-sys-usb` does not start, it could be due to a PCI passthrough problem. For more details on this issue along with possible solutions, look at *PCI passthrough issues*.

### **Can't attach a USB device / USB device not showing in qvm-usb**

To successfully attach a USB device, you require a VM dedicated to handling the USB input and output. For guidance setting up a USB qube, see the *USB documentation*.

Currently (until issue 1082 gets implemented), if you remove the device before detaching it from the qube, Qubes OS (more precisely, `libvirtd`) will think that the device is still attached to the qube and will not allow attaching further devices under the same name. This may be characterized by VM manager crashes and the error message: `Houston, we have a problem`. The easiest way to recover from such a situation is to reboot the qube to which the device was attached. If this isn't an option, you can manually recover from the situation by following the instructions at the *Block Devices documentation*

## “Device attach failed” error

Upon trying to attach a USB device using the `qvm-usb -a vm-name device-vm-name:device` command, you may face the error `Device attach failed: no device info received, connection failed, check backend side for details`. This error mainly arises due to problems specific to the particular device, such as the device being incompatible with `qvm-usb` or a broken cable.

## Attaching device to a qube works, but the device disconnects or disappears upon usage

After attaching a device to a qube, upon attempting to use the device results in the device disappearing or disconnecting. This may be observed by the device no longer existing in the Devices widget or the application within the attached qube indicating the device is no longer found.

As a first line of defense, increase the amount of memory given to the USB VM (`sys-usb`). High-bandwidth devices such as webcams have been [observed](#) to need more memory in `sys-usb`. If increasing the amount of memory does not resolve the issue, check kernel logs within `sys-usb` as well as the attached qube for errors before filing a bug report.

## USB VM does not boot after creating and assigning USB controllers to it

This is probably because one of the controllers does not support reset. In Qubes R2 any such errors were ignored. In Qubes R3.x they are not. In R4.x, devices that are automatically added to `sys-net` and `sys-usb` on install but do not support FLR will be attached with the `no-strict-reset` option, but see the related warning in the last sentence in this answer.

A device that does not support reset is not ideal and generally should not be assigned to a VM.

Most likely the offending controller is a USB 3.0 device. You can remove this controller from the USB VM, and see if this allows the VM to boot. Alternatively you may be able to disable USB 3.0 in the BIOS. If the BIOS does not have the option to disable USB 3.0, try running the following command in `dom0` to force USB 2.0 modes for the USB ports:

```
lspci -nn | grep USB | cut -d '[' -f3 | cut -d ']' -f1 | xargs -I@ setpci -H1 -d @ d0.1=0
```

Errors suggesting this issue:

- in `x1` `dmesg` output:

```
(XEN) [VT-D] It's disallowed to assign 0000:00:1a.0 with shared RMRR at dbe9a000_
↪ for Dom19.
(XEN) XEN_DOMCTL_assign_device: assign 0000:00:1a.0 to dom19 failed (-1)
```

- during `qvm-start sys-usb`:

```
internal error: Unable to reset PCI device [...] no FLR, PM reset or bus reset_
↪ available.
```

Another solution would be to set the `pci_strictreset` option in `dom0`:

- In Qubes R4.x, when attaching the PCI device to the VM (where `<BDF>` can be obtained from running `qvm-pci`):

```
qvm-pci attach --persistent --option no-strict-reset=true usbVM dom0:<BDF>
```

- In Qubes R3.x, by modifying the VM’s properties:

```
qvm-prefs usbVM -s pci_strictreset false
```

These options allow the VM to ignore the error and the VM will start. Please review the notes in the `qvm-prefs` man page and [here](#) and be aware of the potential risks.

### Can't use keyboard or mouse after creating sys-usb

You risk locking yourself out of your computer if you have a USB keyboard and use full disk encryption alongside sys-usb. On boot, the keyboard may be inactive, preventing you from entering your LUKS decryption password.

When you enable a USB qube, it hides all the USB controllers from dom0, even before it gets started. So, if your only keyboard is on USB, you should undo this hiding.

To solve the problem, disable the USB qube by not having it autostart, or unassigning your USB controller(s) from it. If you had created the USB qube by checking the box in the installer, then your USB controller(s) are probably hidden from dom0. To unhide them, reverse the procedure described in [how to hide USB controllers from dom0](#) (i.e., remove `rd.qubes.hide_all_usb` instead of adding it).

Note that this procedure will attach your USB controllers to dom0, so do this only with USB devices you trust.

If your computer has a PS/2 port, you may instead use a PS/2 keyboard to enter the LUKS password.

### “qubes-usb-proxy not installed in the VM” error

When trying to [create and use a USB qube](#) with the `qubes-usb-proxy` package, you may receive this error: `ERROR: qubes-usb-proxy not installed in the VM.`

If you encounter this error, you can install the `qubes-usb-proxy` with the package manager in the VM you want to attach the USB device to. Depending on your operating system, open a terminal in the template and enter one of the following commands:

- Fedora: `sudo dnf install qubes-usb-proxy`
- Debian/Ubuntu: `sudo apt-get install qubes-usb-proxy`

## 1.12.44 GUI troubleshooting

### Can't click on anything after connecting 4k external display

When you connect a 4K external display, you may be unable to click on anything but a small area in the upper-right corner.

When a qube starts, a fixed amount of RAM is allocated to the graphics buffer called video RAM. This buffer needs to be at least as big as the whole desktop, accounting for all displays that are or will be connected to the machine. By default, it is as much as needed for the current display and an additional full HD (FHD) display (1920×1080 8 bit/channel RGBA). This logic fails when the machine has primary display in FHD resolution and, after starting some qubes, a 4K display is connected. If the buffer is too small, and internal desktop resize fails.

The solution to this problem is to increase the minimum size of the video RAM buffer, as explained in [GUI Configuration](#).

## Screen blanks / Weird computer glitches like turning partially black or black boxes

You may encounter seemingly random screen blanking while using Qubes, where the screen will black and shows the logon screen, yet, only the active window will show when you move the mouse or use the keyboard. Sometimes, you will get random black screens or black boxes.

Similarly, while working, the XScreenSaver dialog may pop up (indicating the screen is locked) and the screen goes black. However, the screen is not locked, and you have to move a window to redraw the screen.

If you are experiencing any of the above symptoms, try disabling the window compositor:

Q → System Tools → Window Manager Tweaks → Compositor → uncheck “Enable display compositing”

## Post installation, screen goes black and freezes following LUKS decryption

After installing Qubes, you may experience a black screen after entering your LUKS decryption password. To fix the problem, use your preferred text editor (nano works) to edit `/mnt/sysimage/boot/efi/EFI/qubes/xen.cfg`, adding the `efi=no-rs` option to the end of the `options=` line. For example:

```
[4.14.18-1.pvops.qubes.x86_64]
options=loglvl=all dom0_mem=min:1024M dom0_mem=max:4096M iommu=no-igfx ucode=scan efi=no-
↪rs
```

Note that the `/mnt/sysimage/boot/efi/EFI/qubes/xen.cfg` path applies when running from the installer (either directly after installation, before the reboot, or by starting the installer again in recovery mode). On the actual installed system, the file to edit is `/boot/efi/EFI/qubes/xen.cfg` – but it may be hard to access directly when your system won't boot.

## Can start VM, but can't launch any applications

If you can start your VM, but can't launch any applications, then you need to fix the issues from the VM console, accessible from xen through:

```
qvm-start <VMname> # Make sure the VM is started
qvm-console-dispvm <VMname>
```

After launching a VM console using `qvm-console-dispvm`, you may look at the `qubes-gui-agent` service state with:

```
systemctl status -l qubes-gui-agent
```

If the service is in a failed state, you should see some messages on why it failed.

Another helpful place to look into is `/home/user/.xsession-errors`, which may also contain some hints what is wrong.

## Disable audited messages

During troubleshooting, you may be getting a lot of ‘audit’ messages which make the log very noisy. To disable audited messages, you need to edit your VM kernel parameters:

```
previous_kernel_parameters=$(qvm-prefs --get <VMname> kernelopts) # Get current kernel_
↪parameters
qvm-prefs --set <VMname> kernelopts "<previous_kernel_parameters> audit=0"
```

Then, restart your VM.

Once your troubleshooting is done, don’t forget to remove this kernel parameters, it makes troubleshooting VMs not starting easier.

## 1.12.45 Media troubleshooting

### Can’t play media videos in a VM due to missing codecs

If you’re having trouble playing a video file in a qube, you’re probably missing the required codecs. The easiest way to resolve this is to install VLC Media Player and use that to play your video files. You can do this in multiple different template distros by following the instructions [here](#).

## Video lagging

Playing videos may cause lags since software decoding uses a lot of CPU.

Depending on your video player, there are some settings that may smoothen video plays:

- If using VLC media player, go to Tools→ Preferences → Video → Output. By default, the Output is set to “Automatic”. Go through the list and try out other output options to see if any makes videos run smoother.
- If using mpv media player, you may be able to improve performance by entering `mpv --profile=sw-fast --vo=x11` in a terminal.
- For video lags when playing videos from a browser, disable hardware acceleration in the browser. If the problem arises when watching streams, it may be smoother to use `streamlink` to view streams in mpv instead of using the browser.

## 1.12.46 VPN troubleshooting

### Tips

- If using qubes-vpn, check the VPN service’s log in the VPN VM by running:

```
sudo journalctl -u qubes-vpn-handler
```

- Always test your basic VPN connection before adding scripts.
- Test DNS: Ping a familiar domain name from an appVM. It should print the IP address for the domain.
- Use `iptables -L -v` and `iptables -L -v -t nat` to check firewall rules. The latter shows the critical PR-QBS chain that enables DNS forwarding.

## VPN does not reconnect after suspend

This applies when using OpenVPN.

After suspend/resume, OpenVPN may not automatically reconnect. In order to get it to work, you must kill the OpenVPN process and restart it.

## VPN stuck at “Ready to start link”

After setting up OpenVPN and restarting the VM, you may be repeatedly getting the popup “Ready to start link”, but the VPN isn’t connected.

To figure out the root of the problem, check the VPN logs in `/var/logs/syslog`. The log may reveal issues like missing OpenVPN libraries, which you can then install.

## 1.12.47 Firewall

### Introduction

This page explains use of the firewall in Qubes 4.2, using `nftables`.

In Qubes 4.1, all firewall components used `iptables`. For details of that usage see [here](#)

### Understanding firewalling in Qubes

Every qube in Qubes is connected to the network via a FirewallVM, which is used to enforce network-level policies. By default there is one default FirewallVM, but the user is free to create more, if needed.

For more information, see the following:

- [https://groups.google.com/group/qubes-devel/browse\\_thread/thread/9e231b0e14bf9d62](https://groups.google.com/group/qubes-devel/browse_thread/thread/9e231b0e14bf9d62)
- <https://blog.invisiblethings.org/2011/09/28/playing-with-qubes-networking-for-fun.html>

### How to edit rules

In order to edit rules for a given qube, select it in the Qube Manager and press the “firewall” button.

If the qube is running, you can open Settings from the Qube Popup Menu.

ICMP and DNS are not accessible in the GUI, but can be changed via `qvm-firewall` described below. Connections to Updates Proxy are not made over a network so can not be allowed or blocked with firewall rules, but are controlled using the relevant policy file (see [R4.x Updates proxy](#) for more detail).

Note that if you specify a rule by DNS name it will be resolved to IP(s) *at the moment of applying the rules*, and not on the fly for each new connection. This means it will not work for servers using load balancing, and traffic to complex web sites which draw from many servers will be difficult to control.

Instead of using the firewall GUI, you can use the `qvm-firewall` command in Dom0 to edit the firewall rules by hand. This gives you greater control than by using the GUI.

The firewall rules for each qube are saved in an XML file in that qube’s directory in dom0:

```
/var/lib/qubes/appvms/<vm-name>/firewall.xml
```

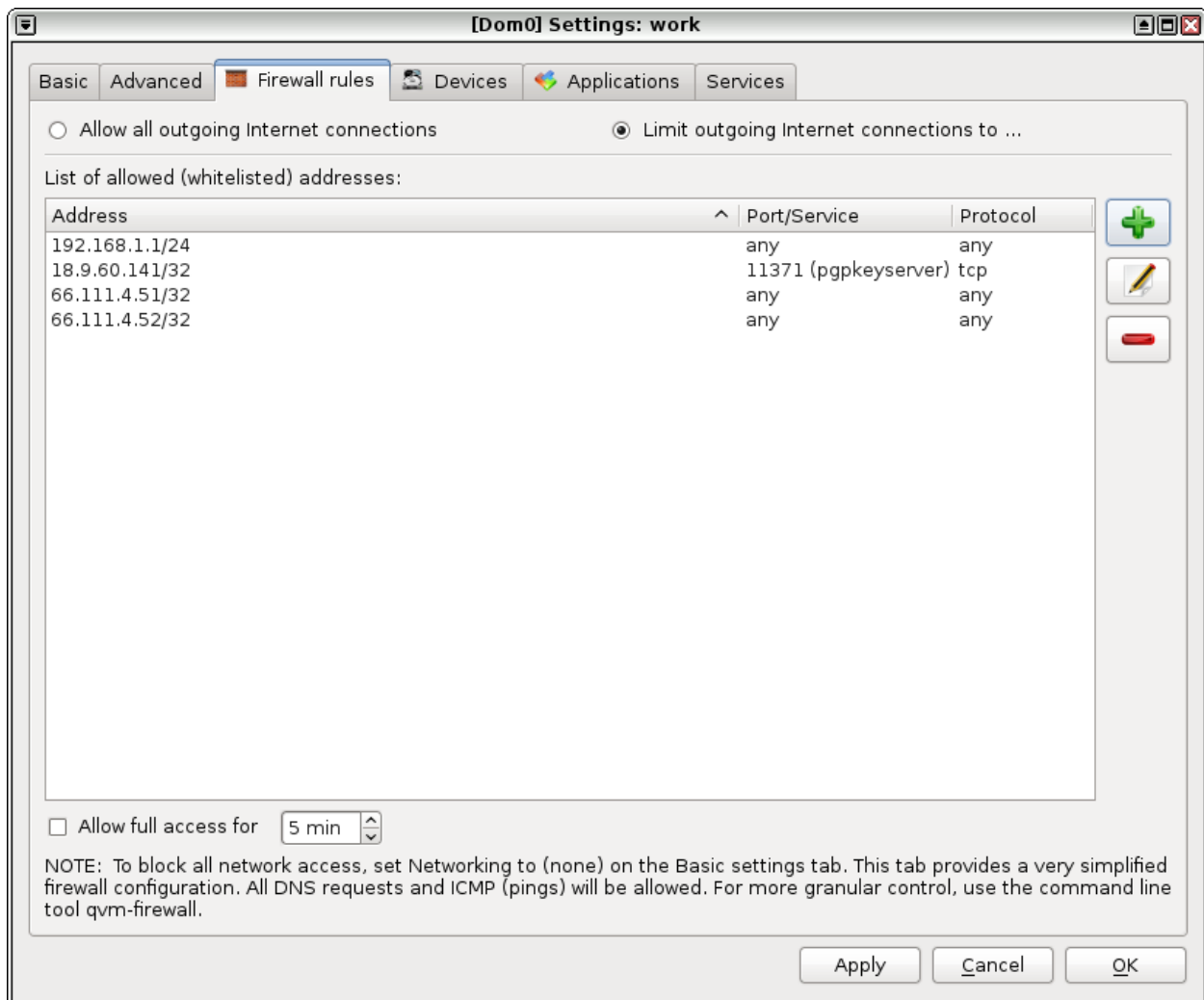


Fig. 9: r4.0-manager-firewall.png



Rules are implemented on the netvm.

You can also manually create rules in the qube itself using standard firewalling controls. See [Where to put firewall rules](#). In complex cases, it might be appropriate to load a ruleset using `nft -f /path/to/ruleset` called from `/rw/config/rc.local`, the ruleset file can be populated from the current ruleset using `nft list ruleset > /path/to/ruleset`, you should add `flush ruleset` at the top of the file to remove all existing rules before loading them. if you do this, be aware that `rc.local` is called *after* the network is up, so local rules should not be relied upon to block leaks.

## Reconnecting qubes after a NetVM reboot

Normally Qubes doesn't let the user stop a NetVM if there are other qubes running which use it as their own NetVM. But in case the NetVM stops for whatever reason (e.g. it crashes, or the user forces its shutdown via `qvm-kill` via terminal in Dom0), Qubes R4.x will often automatically repair the connection. If it does not, then there is an easy way to restore the connection to the NetVM by issuing in dom0:

```
qvm-prefs <vm> netvm <netvm>
```

Normally qubes do not connect directly to the actual NetVM (sys-net by default) which has networking devices, but rather to the default sys-firewall first, and in most cases it would be the NetVM that will crash, e.g. in response to S3 sleep/restore or other issues with WiFi drivers. In that case it is only necessary to issue the above command once, for the sys-firewall (this assumes default VM-naming used by the default Qubes installation):

```
qvm-prefs sys-firewall netvm sys-net
```

## Network service qubes

Qubes does not support running any networking services (e.g. VPN, local DNS server, IPS, ...) directly in a qube that is used to run the Qubes firewall service (usually sys-firewall) for good reasons. In particular, if you want to ensure proper functioning of the Qubes firewall, you should not tinker with nftables rules in such qubes.

Instead, you should deploy a network infrastructure such as

```
sys-net <--> sys-firewall-1 <--> network service qube <--> sys-firewall-2 <--> [client_
↔ qubes]
```

Thereby sys-firewall-1 is only needed if you have other client qubes connected there, or you want to manage the traffic of the local network service qube. The sys-firewall-2 proxy ensures that:

1. Firewall changes done in the network service qube cannot render the Qubes firewall ineffective.
2. Changes to the Qubes firewall by the Qubes maintainers cannot lead to unwanted information leakage in combination with user rules deployed in the network service qube.
3. A compromise of the network service qube does not compromise the Qubes firewall.

If you adopt this model, you should be aware that all traffic will arrive at the `network service qube` appearing to originate from the IP address of `sys-firewall-2`.

For the VPN service please also look at the [VPN documentation](#).

## Enabling networking between two qubes

Normally any networking traffic between qubes is prohibited for security reasons. However, in special situations, you might want to selectively allow specific qubes to establish networking connectivity between each other. For example, this might be useful in some development work, when you want to test networking code, or to allow file exchange between HVM domains (which do not have Qubes tools installed) via SMB/SSH/NFS protocols.

In order to allow networking from qube A (client) to qube B (server) follow these steps:

- Make sure both A and B are connected to the same firewall vm (by default all VMs use the same firewall VM).
- Note the Qubes IP addresses assigned to both qubes. This can be done using the `qvm-ls -n` command, or via the Qubes Manager using the IP column.
- Start both qubes, and also open a terminal in the firewall VM
- In the firewall VM's terminal enter the following nftables rule:

```
sudo nft add rule ip qubes custom-forward ip saddr <IP address of A> ip daddr <IP_
↪address of B> ct state new,established,related counter accept
```

- In qube B's terminal enter the following nftables rule:

```
sudo nft add rule qubes custom-input ip saddr <IP address of A> ct state new,established,
↪related counter accept
```

- Now you should be able to reach B from A – test it using e.g. ping issued from A. Note however, that this doesn't allow you to reach A from B – for this you would need two more rules, with A and B swapped.
- If everything works as expected, then you should write the above nftables rules into firewallVM's `qubes-firewall-user-script` script. This script is run when the netvm starts up. You should also write relevant rules in A and B's `rc.local` script which is run when the qube is launched. Here's an example how to update the script:

```
[user@sys-firewall ~]$ sudo -i
[root@sys-firewall user]# echo "nft add rule ip qubes custom-forward ip saddr 10.137.2.
↪25 ip daddr 10.137.2.6 ct state new,established,related counter accept" >> /rw/config/
↪qubes-firewall-user-script
```

- Here is an example how to update `rc.local`:

```
[user@B ~]$ sudo -i
[root@B user]# echo "nft add rule qubes custom-input ip saddr 10.137.2.25 accept" >> /rw/
↪config/rc.local
```

## Opening a single TCP port to other network-isolated qube

In the case where a specific TCP port needs to be exposed from a qubes to another one, you do not need to enable networking between them but you can use the qubes RPC service `qubes.ConnectTCP`.

### 1. Simple port binding

Consider the following example. `mytcp-service` qube has a TCP service running on port 444 and `untrusted` qube needs to access this service.

- In `dom0`, add the following to `/etc/qubes/policy.d/30-user-networking.policy`: (it could be `another-other-name.policy` – just remember to keep it consistent)

```
qubes.ConnectTCP * untrusted @default allow target=mytcp-service
```

- In untrusted, use the Qubes tool `qvm-connect-tcp`:

```
[user@untrusted #]$ qvm-connect-tcp 444:@default:444
```

Note: The syntax is the same as SSH tunnel handler. The first ``444`` correspond to the localport destination of ``untrusted``, ``@default`` the remote machine and the second ``444`` to the remote machine port.

The service of `mytcp-service` running on port 444 is now accessible in `untrusted` as `localhost:444`.

Here `@default` is used to hide the destination qube which is specified in the Qubes RPC policy by `target=mytcp-service`. Equivalent call is to use the tool as follow:

```
[user@untrusted #]$ qvm-connect-tcp ::444
```

which means to use default local port of `untrusted` as the same of the remote port and unspecified destination qube is `@default` by default in `qrexec` call.

## 2. Binding remote port on another local port

Consider now the case where someone prefers to specify the destination qube and use another port in `untrusted`, for example `10044`. Instead of previous case, add

```
qubes.ConnectTCP * untrusted mytcp-service allow
```

in `/etc/qubes/policy.d/30-user-networking.policy` and in `untrusted`, use the tool as follow:

```
[user@untrusted #]$ qvm-connect-tcp 10444:mytcp-service:444
```

The service of `mytcp-service` running on port 444 is now accessible in `untrusted` as `localhost:10444`.

## 3. Binding to different qubes using RPC policies

One can go further than the previous examples by redirecting different ports to different qubes. For example, let assume that another qube `mytcp-service-bis` with a TCP service is running on port 445. If someone wants `untrusted` to be able to reach this service but port 445 is reserved to `mytcp-service-bis` then, in `dom0`, add the following to `/etc/qubes/policy.d/30-user-networking.policy`:

```
qubes.ConnectTCP +445 untrusted @default allow target=mytcp-service-bis
```

In that case, calling `qvm-connect-tcp` like previous examples, will still bind TCP port 444 of `mytcp-service` to `untrusted` but now, calling it with port 445

```
[user@untrusted #]$ qvm-connect-tcp ::445
```

will restrict the binding to only the corresponding TCP port of `mytcp-service-bis`.

## 4. Permanent port binding

For creating a permanent port bind between two qubes, `systemd` can be used. We use the case of the first example. In `/rw/config` (or any place you find suitable) of qube `untrusted`, create `my-tcp-service.socket` with content:

```
[Unit]
Description=my-tcp-service

[Socket]
ListenStream=127.0.0.1:444
Accept=true

[Install]
WantedBy=sockets.target
```

and `my-tcp-service@.service` with content:

```
[Unit]
Description=my-tcp-service

[Service]
ExecStart=qrexec-client-vm '' qubes.ConnectTCP+444
StandardInput=socket
StandardOutput=inherit
```

In `/rw/config/rc.local`, append the lines:

```
cp -r /rw/config/my-tcp-service.socket /rw/config/my-tcp-service@.service /lib/systemd/
↪systemd/
systemctl daemon-reload
systemctl start my-tcp-service.socket
```

When the qube `untrusted` has started (after a first reboot), you can directly access the service of `mytcp-service` running on port 444 as `localhost:444`.

## Port forwarding to a qube from the outside world

In order to allow a service present in a qube to be exposed to the outside world in the default setup (where the qube has `sys-firewall` as network VM, which in turn has `sys-net` as network VM) the following needs to be done:

- In the `sys-net` VM:
  - Route packets from the outside world to the `sys-firewall` VM
  - Allow packets through the `sys-net` VM firewall
- In the `sys-firewall` VM:
  - Route packets from the `sys-net` VM to the VM
  - Allow packets through the `sys-firewall` VM firewall
- In the qube `QubeDest`:
  - Allow packets through the qube firewall to reach the service

As an example we can take the use case of qube `QubeDest` running a web server listening on port 443 that we want to expose on our physical interface `ens6`, but only to our local network `192.168.x.y/24`.

Note: To have all interfaces available and configured, make sure the 3 qubes are up and running

Note: [Issue #4028](#) discusses adding a command to automate exposing the port.

## 1. Identify the IP addresses you will need to use for sys-net, sys-firewall and the destination qube.

You can get this information using various methods, but only the first one can be used for sys-net outside world IP:

- by running this command in each qube: `ip -4 -br a | grep UP`
- using `qvm-ls -n`
- in the Qubes Manager window using the column IP
- from the Settings Window for the qube

Note the IP addresses you will need, they will be required in the next steps.

Note: The `vifx.0` interface is the one used by qubes connected to this netvm so it is *not* an outside world interface.

## 2. Route packets from the outside world to the FirewallVM

For the following example, we assume that the physical interface `ens6` in `sys-net` is on the local network `192.168.x.y` with the IP `192.168.x.n`, and that the IP address of `sys-firewall` is `10.137.1.z`.

In the `sys-net` VM's Terminal, the first step is to define an `nftables` chain that will receive DNAT rules to relay the network traffic on a given port to the qube NetVM, we recommend to define a new chain for each destination qube to ease rules management:

```
nft add chain qubes custom-dnat-qubeDEST '{ type nat hook prerouting priority filter +1 ;
↳ policy accept; }'
```

**Note:** the name `custom-dnat-qubeDST` is arbitrary

**Note:** while we use a DNAT chain for a single qube, it's totally possible to have a single DNAT chain for multiple qubes

Second step, code a natting firewall rule to route traffic on the outside interface for the service to the `sys-firewall` VM

```
nft add rule qubes custom-dnat-qubeDEST iif == "ens6" ip saddr 192.168.x.y/24 tcp dport 443
↳ ct state new,established,related counter dnat 10.137.1.z
```

Third step, code the appropriate new filtering firewall rule to allow new connections for the service

```
nft add rule qubes custom-forward iif == "ens6" ip saddr 192.168.x.y/24 ip daddr 10.137.
↳ 1.z tcp dport 443 ct state new,established,related counter accept
```

**Note:** If you do not wish to limit the IP addresses connecting to the service, remove `ip saddr 192.168.x.y/24` from the rules

If you want to expose the service on multiple interfaces, repeat the steps 2 and 3 described above, for each interface.

Verify the rules on `sys-net` firewall correctly match the packets you want by looking at its counters, check for the counter lines in the chains `custom-forward` and `custom-dnat-qubeDEST`:

```
nft list table ip qubes-firewall
```

In this example, we can see 7 packets in the forward rule, and 3 packets in the dnat rule:

```
chain custom-forward {
    iif "ens6" ip saddr 192.168.x.y/24 ip daddr 10.137.1.z tcp dport 443 ct state new,
    ↪ established,related counter packets 7 bytes 448 accept
}

chain custom-dnat-qubeDEST {
    type nat hook prerouting priority filter + 1; policy accept;
    iif "ens6" ip saddr 192.168.x.y/24 tcp dport 443 ct state new,established,related
    ↪ counter packets 3 bytes 192 dnat to 10.138.33.59
}
```

(Optional) You can send a test packet by trying to connect to the service from an external device using the following command:

```
telnet 192.168.x.n 443
```

Once you have confirmed that the counters increase, store the commands used in the previous steps in `/rw/config/rc.local` so they get set on sys-net start-up:

```
[user@sys-net user]$ sudo -i
[root@sys-net user]# nano /rw/config/qubes-firewall-user-script
```

Content of `/rw/config/qubes-firewall-user-script` in sys-net:

```
#!/bin/sh

# create the dnat chain for qubeDEST if it doesn't already exist
if nft add chain qubes custom-dnat-qubeDEST '{ type nat hook prerouting priority filter
    ↪ +1 ; policy accept; }'
then
    # create the dnat rule
    nft add rule qubes custom-dnat-qubeDEST iif == "ens6" saddr 192.168.x.y/24 tcp dport
    ↪ 443 ct state new,established,related counter dnat 10.137.1.z

    # allow forwarded traffic
    nft add rule qubes custom-forward iif == "ens6" ip saddr 192.168.x.y/24 ip daddr 10.
    ↪ 137.1.z tcp dport 443 ct state new,established,related counter accept
fi
```

### 3. Route packets from the FirewallVM to the VM

For the following example, we use the fact that the physical interface of sys-firewall, facing sys-net, is `eth0`. Furthermore, we assume that the target VM running the web server has the IP address `10.137.0.xx` and that the IP address of sys-firewall is `10.137.1.z`.

In the sys-firewall VM's Terminal, add a DNAT chain that will contain routing rules:

```
nft add chain qubes custom-dnat-qubeDEST '{ type nat hook prerouting priority filter +1 ;
    ↪ policy accept; }'
```

Second step, code a natting firewall rule to route traffic on the outside interface for the service to the destination qube

```
nft add rule qubes custom-dnat-qubeDEST iif == "eth0" ip saddr 192.168.x.y/24 tcp dport
    ↪ 443 ct state new,established,related counter dnat 10.137.0.xx
```

Third step, code the appropriate new filtering firewall rule to allow new connections for the service

```
nft add rule qubes custom-forward iif == "eth0" ip saddr 192.168.x.y/24 ip daddr 10.137.
↳0.xx tcp dport 443 ct state new,established,related counter accept
```

**Note:** If you do not wish to limit the IP addresses connecting to the service, remove `ip saddr 192.168.x.y/24` from the rules

Once you have confirmed that the counters increase, store these commands in the script `/rw/config/qubes-firewall-user-script`

```
[user@sys-net user]$ sudo -i
[root@sys-net user]# nano /rw/config/qubes-firewall-user-script
```

Content of `/rw/config/qubes-firewall-user-script` in `sys-firewall`:

```
#!/bin/sh

# create the dnat chain for qubeDEST if it doesn't already exist
if nft add chain qubes custom-dnat-qubeDEST '{ type nat hook prerouting priority filter_
↳+1 ; policy accept; }'
then
  # create the dnat rule
  nft add rule qubes custom-dnat-qubeDEST iif == "eth0" tcp dport 443 ct state new,
↳established,related counter dnat 10.137.0.xx

  # allow forwarded traffic
  nft add rule qubes custom-forward iif == "eth0" ip saddr 192.168.x.y/24 ip daddr 10.
↳137.0.xx tcp dport 443 ct state new,established,related counter accept
fi
```

If the service should be available to other VMs on the same system, do not forget to specify the additional rules described earlier in this guide.

#### 4. Allow packets into the qube to reach the service

No routing is required in the destination qube, only filtering.

For the following example, we assume that the target VM running the web server has the IP address `10.137.0.xx`

The according rule to allow the traffic is:

```
nft add rule qubes custom-input tcp dport 443 ip daddr 10.137.0.xx ct state new,
↳established,related counter accept
```

To make it persistent, you need to add this command in the script `/rw/config/rc.local`:

```
[user@qubeDEST user]$ sudo -i
[root@qubeDEST user]# echo 'nft add rule qubes custom-input tcp dport 443 ip daddr 10.137.
↳0.xx ct state new,established,related counter accept' >> /rw/config/rc.local
```

This time testing should allow connectivity to the service as long `qubeDEST` is running and the service is up :-)

## Where to put firewall rules

Implicit in the above example *scripts*, but worth calling attention to: for all qubes *except* those supplying networking, nftables commands should be added to the `/rw/config/rc.local` script. For service qubes supplying networking (sys-firewall and sys-net inclusive), nftables commands should be added to `/rw/config/qubes-firewall-user-script`.

## Firewall troubleshooting

Firewall logs are stored in the systemd journal of the qube the firewall is running in (probably sys-firewall). You can view them by running `sudo journalctl -u qubes-firewall.service` in the relevant qube. Sometimes these logs can contain useful information about errors that are preventing the firewall from behaving as you would expect.

An effective console utility to troubleshoot network is `tcpdump`, it can be used to display network packets entering or leaving network interfaces.

For instance, if you want to check if your network interface `eth0` is receiving packets on port TCP 443 from the network 192.168.x.y, you can run this command:

```
tcpdump -i eth0 -nn dst port 443 and src net 192.168.x.y/24
```

This can be used effectively in a destination qube and its Network VM to see if forwarding / NAT rules are working.

## Nftables tips

A simple way to experiment changes with your ruleset can be achieved by saving the current working ruleset in two files, one for backup and the other for making changes.

By adding `flush ruleset` at the top of the file, you can achieve atomic update, which mean the new ruleset would replace the current one only if it fully succeed to load.

You can dump the ruleset in two files using the following command:

```
nft list ruleset | tee nft_backup | tee nft_new_ruleset
```

Then, edit `nft_new_ruleset`, add `flush ruleset` on top and make changes, load it with `nft -f nft_new_ruleset`.

You can revert to the original ruleset with the following commands:

```
nft flush ruleset && nft -f nft_backup
```

## 1.12.48 Data leaks

### The Role of the Firewall

*Firewalling in Qubes* is not intended to be a leak-prevention mechanism.

There are several reasons for this, which will be explained below. However, the main reason is that Qubes cannot prevent an attacker who has compromised one app qube with restrictive firewall rules from leaking data via cooperative covert channels through another compromised app qube with nonrestrictive firewall rules.

For example, suppose you have an `email` app qube. You have set the firewall rules for `email` such that it can communicate only with your email server. Now suppose that an attacker sends you a GPG-encrypted message which exploits a hypothetical bug in the GnuPG process. There are now multiple ways the attacker could proceed to leak data (such as



confidential email messages) from email. The most obvious way is by simply emailing the data to himself. Another possibility is that the attacker has also compromised another one of your app qubes, such as your netvm, which is normally assumed to be untrusted and has unrestricted access to the network. In this case, the attacker might move data from email to the netvm via a covert channel, such as the CPU cache. Such covert channels have been described and even implemented in some “lab environments” and might allow for bandwidths of even a few tens of bits/sec. It is unclear whether such channels could be implemented in a real world system, where multiple VMs are running at the same time, each handling tens or hundreds of processes, all using the same cache memory, but it is worth keeping in mind. Of course, this would require special malware written specifically to attack Qubes OS, and perhaps even a specific Qubes OS version and/or configuration. Nevertheless, it might be possible.

Note that physically air-gapped machines are not necessarily immune to this problem. Covert channels can potentially take many forms (e.g., sneakernet thumb drive, bluetooth, or even microphone and speakers).

For a further discussion of covert channels, see [this thread](#) and #817.

## Types of Data Leaks

In order to understand and attempt to prevent data leaks in Qubes, we must distinguish among three different types of relevant data leaks:

1. **Intentional leaks.** Malicious software which actively tries to leak data out of an app qube, perhaps via cooperative covert channels established with other malicious software in another app qube or on some server via networking, if networking, even limited, is allowed for the app qube.
2. **Intentional sniffing.** Malicious software trying to use side channels to, e.g., actively guess some key material used in another VM by some non-malicious software there (e.g., non-leak-proof GPG accidentally leaking out bits of the private key by generating some timing patterns when using this key for some crypto operation). Such attacks have been described in the academic literature, but it is doubtful that they would succeed in practice in a moderately busy general purpose system like Qubes OS where the attacker normally has no way to trigger the target crypto operation explicitly and it is normally required that the attacker trigger many such operations.
3. **Unintentional leaks.** Non-malicious software which is either buggy or doesn’t maintain the privacy of user data, whether by design or accident. For example, software which automatically sends error reports to a remote server, where these reports contain details about the system which the user did not want to share. Both Qubes firewall and an empty NetVM (i.e., setting the NetVM of an app qube to “none”) can fully protect against leaks of type 3. However, neither Qubes firewall nor an empty NetVM are guaranteed to protect against leaks of types 1 and 2. There are few effective, practical policy measures available to end-users today to stop the leaks of type 1. It is likely that the only way to fully protect against leaks of type 2 is to either pause or shut down all other VMs while performing sensitive operations in the target VM(s) (such as key generation).

For further discussion, see [this thread](#).

### 1.12.49 Passwordless root access in qubes

Background (/etc/sudoers.d/qubes in VM):

```
user ALL=(ALL) NOPASSWD: ALL

# WTF?! Have you lost your mind?!
#
# In Qubes VMs there is no point in isolating the root account from
# the user account. This is because all the user data is already
# accessible from the user account, so there is no direct benefit for
# the attacker if she could escalate to root (there is even no benefit
# in trying to install some persistent rootkits, as the VM's root
```

(continues on next page)

(continued from previous page)

```
# filesystem modifications are lost upon each start of a VM).
#
# One might argue that some hypothetical attacks against the
# hypervisor or the few daemons/backends in Dom0 (so VM escape
# attacks) most likely would require root access in the VM to trigger
# the attack.
#
# That's true, but mere existence of such a bug in the hypervisor or
# Dom0 that could be exploited by a malicious VM, no matter whether
# requiring user, root, or even kernel access in the VM, would be
# FATAL. In such situation (if there was such a bug in Xen) there
# really is no comforting that: "oh, but the mitigating factor was
# that the attacker needed root in VM!" We're not M$, and we're not
# gonna BS our users that there are mitigating factors in that case,
# and for sure, root/user isolation is not a mitigating factor.
#
# Because, really, if somebody could find and exploit a bug in the Xen
# hypervisor -- as of 2016, there have been only three publicly disclosed
# exploitable bugs in the Xen hypervisor from a VM -- then it would be
# highly unlikely if that person couldn't also found a user-to-root
# escalation in VM (which as we know from history of UNIX/Linux
# happens all the time).
#
# At the same time allowing for easy user-to-root escalation in a VM
# is simply convenient for users, especially for update installation.
#
# Currently this still doesn't work as expected, because some idiotic
# piece of software called PolKit uses own set of policies. We're
# planning to address this in Beta 2. (Why PolKit is an idiocy? Do a
# simple experiment: start 'xinput test' in one xterm, running as
# user, then open some app that uses PolKit and asks for root
# password, e.g. gpk-update-viewer -- observe how all the keystrokes
# with root password you enter into the "secure" PolKit dialog box can
# be seen by the xinput program...)
#
# joanna.
```

Below is a complete list of configuration made according to the above statement, with (not necessary complete) list of mechanisms depending on each of them:

1. sudo (/etc/sudoers.d/qubes):

```
user ALL=(ALL) NOPASSWD: ALL
(...)
```

- Easy user -> root access (main option for the user).
- qvm-usb (not really working, as of R2).

2. PolicyKit (/etc/polkit-1/rules.d/00-qubes-allow-all.rules):

```
//allow any action, detailed reasoning in sudoers.d/qubes
polkit.addRule(function(action,subject) { return polkit.Result.YES; });
```

and /etc/polkit-1/localauthority/50-local.d/qubes-allow-all.pkla:

```
[Qubes allow all]
Identity=*
Action=*
ResultAny=yes
ResultInactive=yes
ResultActive=yes
```

- NetworkManager configuration from normal user (`nm-applet`).
- Updates installation (`gpk-update-viewer`).
- User can use `psexec` just like `sudo` Note: above is needed mostly because Qubes user GUI session isn't treated by PolicyKit/logind as "local" session because of the way in which X server and session is started. Perhaps we will address this issue in the future, but this is really low priority. Patches welcomed anyway.

### 3. Empty root password:

- Used for access to 'root' account from text console (`qvm-console-dispvm`) - the only way to access the VM when GUI isn't working.
- Can be used for easy 'su -' from user to root.

## Replacing passwordless root access with Dom0 user prompt

While the Qubes developers support the statement above, some Qubes users may wish to enable user/root isolation in VMs anyway. We do not support this in any of our packages, but of course nothing is preventing a user from modifying his or her own system. A list of steps to do so is provided in the [Qubes community guide, Replacing passwordless root with a dom0 prompt](#) **without any guarantee of safety, accuracy, or completeness. Proceed at your own risk. Do not rely on this for extra security.**

## Dom0 passwordless root access

There is also passwordless user->root access in dom0. As stated in the comment in sudo configuration there (which is different from the one in individual qubes), there is really no point in user/root isolation, because all the user data (and VM management interface) is already accessible from dom0 user level, so there is nothing more to get from dom0 root account.

### 1.12.50 Device handling security

Any additional ability a VM gains is additional attack surface. It's a good idea to always attach the minimum entity required in a VM.

For example, attaching a full USB-device offers [more attack surface than attaching a single block device](#), while attaching a full block device (e.g. `sda`) again offers more attack surface than attaching a single partition (e.g. `sda1`), since the targetVM doesn't have to parse the partition-table. (Attaching a full block device offers the advantage that most file-managers will mount and display them correctly, whereas they don't expect single partitions to be added and therefore don't handle them correctly.)

## PCI Security

Attaching a PCI device to a qube has serious security implications. It exposes the device driver running in the qube to an external device (and sourceVM, which contains the device - e.g. `sys-usb`). In many cases a malicious device can choose what driver will be loaded (for example by manipulating device metadata like vendor and product identifiers) - even if the intended driver is sufficiently secure, the device may try to attack a different, less secure driver. Furthermore that VM has full control of the device and may be able to exploit bugs or malicious implementation of the hardware, as well as plain security problems the hardware may pose. (For example, if you attach a USB controller, all the security implications of USB passthrough apply as well.)

By default, Qubes requires any PCI device to be resettable from the outside (i.e. via the hypervisor), which completely reinitialises the device. This ensures that any device that was attached to a compromised VM, even if that VM was able to use bugs in the PCI device to inject malicious code, can be trusted again. (Or at least as trusted as it was when Qubes booted.)

Some devices do not implement a reset option. In these cases, Qubes by default does not allow attaching the device to any VM. If you decide to override this precaution, beware that the device may only be trusted when attached to the first VM. Afterwards, it should be **considered tainted** until the whole system is shut down. Even without malicious intent, usage data may be leaked.

In case device reset is disabled for any reason, detaching the device should be considered a risk. Ideally, devices for which the `no-strict-reset` option is set are attached once to a VM which isn't shut down until the system is shut down.

Additionally, Qubes restricts the config-space a VM may use to communicate with a PCI device. Only whitelisted registers are accessible. However, some devices or applications require full PCI access. In these cases, the whole config-space may be allowed. You're potentially weakening the device isolation, especially if your system is not equipped with a VT-d Interrupt Remapping unit. This increases the VM's ability to run a [side channel attack](#) and vulnerability to the same. See [Xen PCI Passthrough: PV guests and PCI quirks](#) and [Software Attacks on Intel VT-d](#) (page 7) for more details.

## USB Security

The connection of an untrusted USB device to dom0 is a security risk since the device can attack an arbitrary USB driver (which are included in the linux kernel), exploit bugs during partition-table-parsing or simply pretend to be a keyboard. There are many ready-to-use implementations of such attacks, e.g. a [USB Rubber Ducky](#). The whole USB stack is put to work to parse the data presented by the USB device in order to determine if it is a USB mass storage device, to read its configuration, etc. This happens even if the drive is then assigned and mounted in another qube.

To avoid this risk, use a [USB qube](#).

Attaching a USB device to a VM (USB passthrough) will **expose your target qube** to most of the [security issues](#) associated with the USB-stack. If possible, use a method specific for particular device type (for example, block devices described above), instead of this generic one.

## Security Warning On USB Input Devices

If you connect USB input devices (keyboard and mouse) to a VM, that VM will effectively have control over your system. Because of this, the benefits of using a [USB qube](#) entrusted with a keyboard or other interface device are much smaller than using a fully untrusted USB qube. In addition to having control over your system, such a VM can also sniff all the input you enter there (for example, passwords in the case of a USB keyboard).

There is no simple way to protect against sniffing, but you can make it harder to exploit control over input devices.

If you have only a USB mouse connected to a USB qube, but the keyboard is connected directly to dom0 (using a PS/2 connector, for example), you simply need to lock the screen when you are away from your computer (assuming you don't

use the virtual keyboard of your screen locker). You must do this every time you leave your computer unattended, even if there no risk of anyone else having direct physical access to your computer. This is because you are guarding the system not only against anyone with local access, but also against possible malicious input from a potentially compromised USB qube.

If your keyboard is also connected to a USB qube, things are much harder. Locking the screen (with a traditional password) does not solve the problem, because the USB qube can simply sniff this password and later easily unlock the screen. One possibility is to set up the screen locker to require an additional step to unlock (i.e., two-factor authentication). One way to achieve this is to use a [YubiKey](#), or some other hardware token, or even to manually enter a one-time password.

Support for [two factor authentication](#) was recently added, though there are [issues](#).

### 1.12.51 Anti evil maid (AEM)

#### Background

Please read [this blog article](#).

#### Requirements

The current package requires a TPM 1.2 interface and a working Intel TXT engine. If you cleaned your Intel Management Engine with e.g. [me\\_cleaner](#) while installing [CoreBoot](#) then you are out of luck. For now you have to choose between cleaning your BIOS and deploying Anti Evil Maid.

#### Discussion

#### Installing

In Dom0 install anti-evil-maid:

```
sudo qubes-dom0-update anti-evil-maid
```

For more information, see the [qubes-antievilmaid](#) repository, which includes a [README](#).

#### Security Considerations

[Qubes security guidelines](#) dictate that USB devices should never be attached directly to dom0, since this can result in the entire system being compromised. However, in its default configuration, installing and using AEM requires attaching a USB drive (i.e., [mass storage device](#)) directly to dom0. (The other option is to install AEM to an internal disk. However, this carries significant security implications, as explained [here](#).) This presents us with a classic security trade-off: each Qubes user must make a choice between protecting dom0 from a potentially malicious USB drive, on the one hand, and protecting the system from Evil Maid attacks, on the other hand. Given the practical feasibility of attacks like [BadUSB](#) and revelations regarding pervasive government hardware backdoors, this is no longer a straight-forward decision. New, factory-sealed USB drives cannot simply be assumed to be “clean” (e.g., to have non-malicious microcontroller firmware). Therefore, it is up to each individual Qubes user to evaluate the relative risk of each attack vector against his or her security model.

For example, a user who frequently travels with a Qubes laptop holding sensitive data may be at a much higher risk of Evil Maid attacks than a home user with a stationary Qubes desktop. If the frequent traveler judges her risk of an Evil Maid attack to be higher than the risk of a malicious USB device, she might reasonably opt to install and use AEM. On the other hand, the home user might deem the probability of an Evil Maid attack occurring in her own home to be so low that there is a higher probability that any USB drive she purchases is already compromised, in which case she

might reasonably opt never to attach any USB devices directly to dom0. (In either case, users can—and should—secure dom0 against further USB-related attacks through the use of a [USB VM](#).)

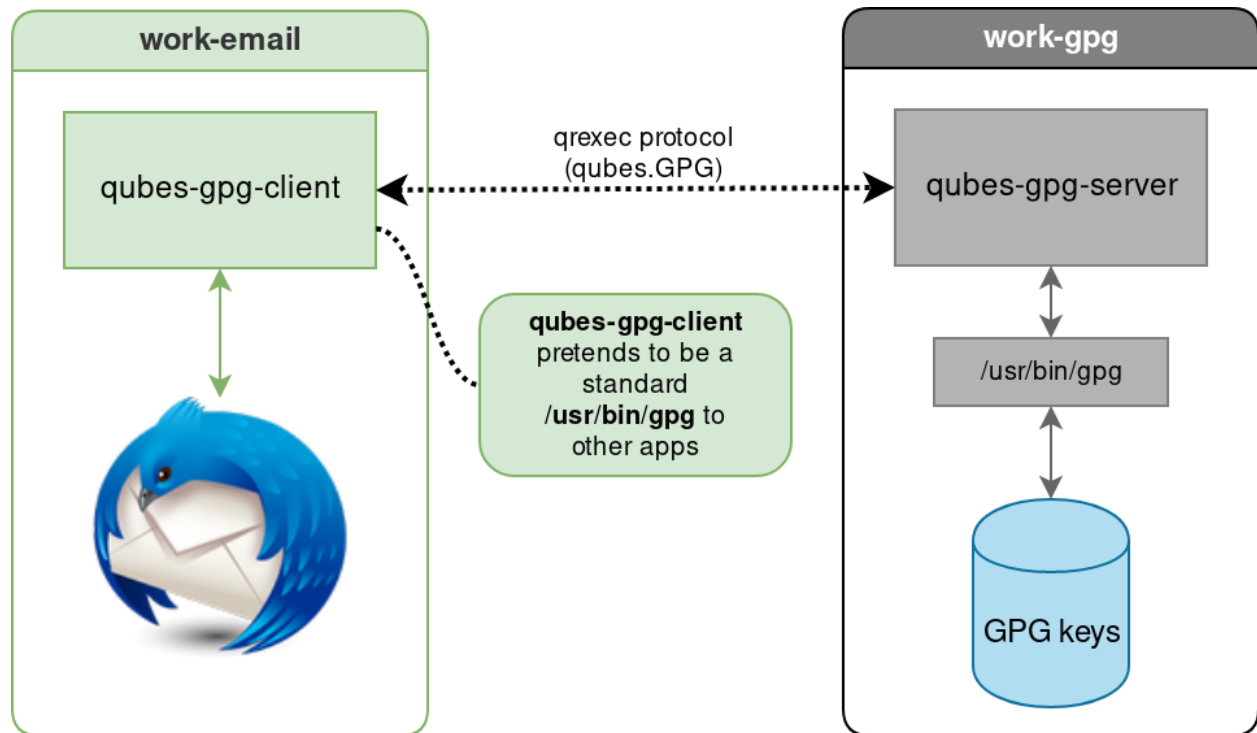
For more information, please see [this discussion thread](#).

### Known issues

- USB 3.0 isn't supported yet
- AEM is not compatible with having an SSD cache

### 1.12.52 Split GPG

Split GPG implements a concept similar to having a smart card with your private GPG keys, except that the role of the “smart card” is played by another Qubes app qube. This way one not-so-trusted domain, e.g. the one where Thunderbird is running, can delegate all crypto operations – such as encryption/decryption and signing – to another, more trusted, network-isolated domain. This way the compromise of your domain where Thunderbird or another client app is running – arguably a not-so-unthinkable scenario – does not allow the attacker to automatically also steal all your keys. (We should make a rather obvious comment here that the so-often-used passphrases on private keys are pretty meaningless because the attacker can easily set up a simple backdoor which would wait until the user enters the passphrase and steal the key then.)

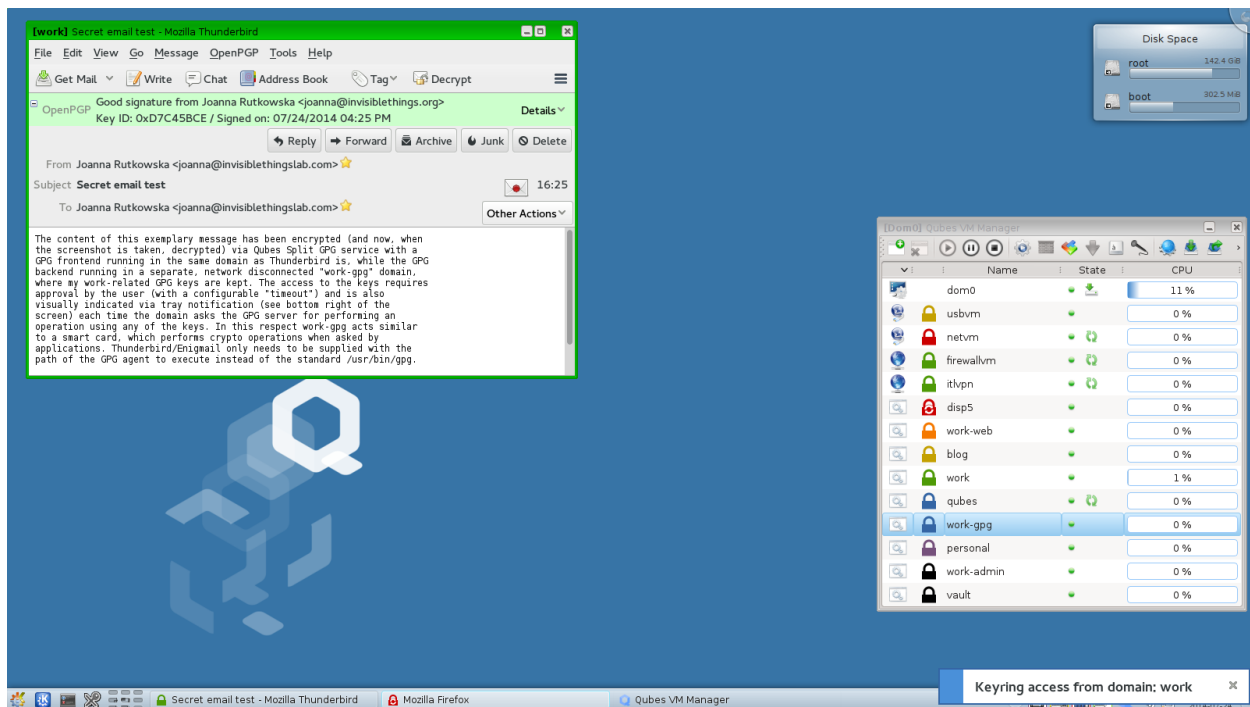


This diagram presents an overview of the Split GPG architecture.

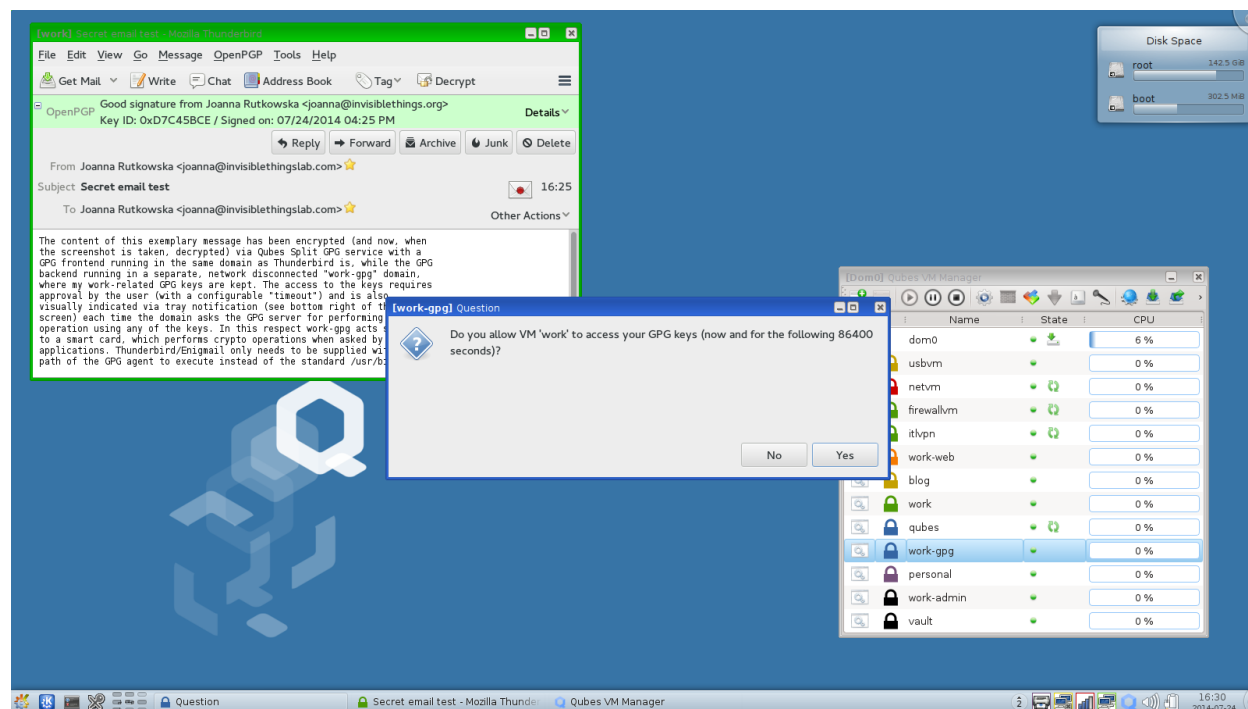
## Advantages of Split GPG vs. traditional GPG with a smart card

It is often thought that the use of smart cards for private key storage guarantees ultimate safety. While this might be true (unless the attacker can find a usually-very-expensive-and-requiring-physical-presence way to extract the key from the smart card) but only with regards to the safety of the private key itself. However, there is usually nothing that could stop the attacker from requesting the smart card to perform decryption of all the user documents the attacker has found or need to decrypt. In other words, while protecting the user's private key is an important task, we should not forget that ultimately it is the user data that are to be protected and that the smart card chip has no way of knowing the requests to decrypt documents are now coming from the attacker's script and not from the user sitting in front of the monitor. (Similarly the smart card doesn't make the process of digitally signing a document or a transaction in any way more secure – the user cannot know what the chip is really signing. Unfortunately this problem of signing reliability is not solvable by Split GPG)

With Qubes Split GPG this problem is drastically minimized, because each time the key is to be used the user is asked for consent (with a definable time out, 5 minutes by default), plus is always notified each time the key is used via a tray notification from the domain where GPG backend is running. This way it would be easy to spot unexpected requests to decrypt documents.







### Configuring Split GPG

In dom0, make sure the `qubes-gpg-split-dom0` package is installed.

```
[user@dom0 ~]$ sudo qubes-dom0-update qubes-gpg-split-dom0
```

Make sure you have the `qubes-gpg-split` package installed in the template you will use for the GPG domain.

For Debian or Whonix:

```
[user@debian-10 ~]$ sudo apt install qubes-gpg-split
```

For Fedora:

```
[user@fedora-32 ~]$ sudo dnf install qubes-gpg-split
```

### Setting up the GPG backend domain

First, create a dedicated app qube for storing your keys (we will be calling it the GPG backend domain). It is recommended that this domain be network disconnected (set its `netvm` to `none`) and only used for this one purpose. In later examples this app qube is named `work-gpg`, but of course it might have any other name.

Make sure that `gpg` is installed there. At this stage you can add the private keys you want to store there, or you can now set up Split GPG and add the keys later. To check which private keys are in your GPG keyring, use:

```
[user@work-gpg ~]$ gpg -K
/home/user/.gnupg/secring.gpg
-----
sec    4096R/3F48CB21 2012-11-15
uid    Qubes OS Security Team <security@qubes-os.org>
```

(continues on next page)



(continued from previous page)

```
ssb  4096R/30498E2A 2012-11-15
(...)
```

This is pretty much all that is required. However, you might want to modify the default timeout: this tells the backend for how long the user's approval for key access should be valid. (The default is 5 minutes.) You can change this via the QUBES\_GPG\_AUTOACCEPT environment variable. You can override it e.g. in ~/.profile:

```
[user@work-gpg ~]$ echo "export QUBES_GPG_AUTOACCEPT=86400" >> ~/.profile
```

Please note that previously, this parameter was set in ~/.bash\_profile. This will no longer work. If you have the parameter set in ~/.bash\_profile you *must* update your configuration.

Please be aware of the caveat regarding passphrase-protected keys in the [Current limitations](#) section.

## Configuring the client apps to use Split GPG backend

Normally it should be enough to set the QUBES\_GPG\_DOMAIN to the GPG backend domain name and use qubes-gpg-client in place of gpg, e.g.:

```
[user@work-email ~]$ export QUBES_GPG_DOMAIN=work-gpg
[user@work-email ~]$ gpg -K
[user@work-email ~]$ qubes-gpg-client -K
/home/user/.gnupg/secring.gpg
-----
sec  4096R/3F48CB21 2012-11-15
uid                               Qubes OS Security Team <security@qubes-os.org>
ssb  4096R/30498E2A 2012-11-15
(...)

[user@work-email ~]$ qubes-gpg-client secret_message.txt.asc
(...)
```

Note that running normal gpg -K in the demo above shows no private keys stored in this app qube.

A note on gpg and gpg2:

Throughout this guide, we refer to gpg, but note that Split GPG uses gpg2 under the hood for compatibility with programs like Enigmail (which now supports only gpg2). If you encounter trouble while trying to set up Split GPG, make sure you're using gpg2 for your configuration and testing, since keyring data may differ between the two installations.

## Advanced Configuration

The qubes-gpg-client-wrapper script sets the QUBES\_GPG\_DOMAIN variable automatically based on the content of the file /rw/config/gpg-split-domain, which should be set to the name of the GPG backend VM. This file survives the app qube reboot, of course.

```
[user@work-email ~]$ sudo bash
[root@work-email ~]$ echo "work-gpg" > /rw/config/gpg-split-domain
```

Split GPG's default qrexec policy requires the user to enter the name of the app qube containing GPG keys on each invocation. To improve usability for applications like Thunderbird with Enigmail, in dom0 place the following line at the top of the file /etc/qubes-rpc/policy/qubes.Gpg:

```
work-email work-gpg allow
```

where `work-email` is the Thunderbird + Enigmail app qube and `work-gpg` contains your GPG keys.

You may also edit the `qrexec` policy file for Split GPG in order to tell Qubes your default `gpg` vm (`qrexec` prompts will appear with the `gpg` vm preselected as the target, instead of the user needing to type a name in manually). To do this, append `,default_target=<vmname>` to ask in `/etc/qubes-rpc/policy/qubes.Gpg`. For the examples given on this page:

```
@anyvm @anyvm ask,default_target=work-gpg
```

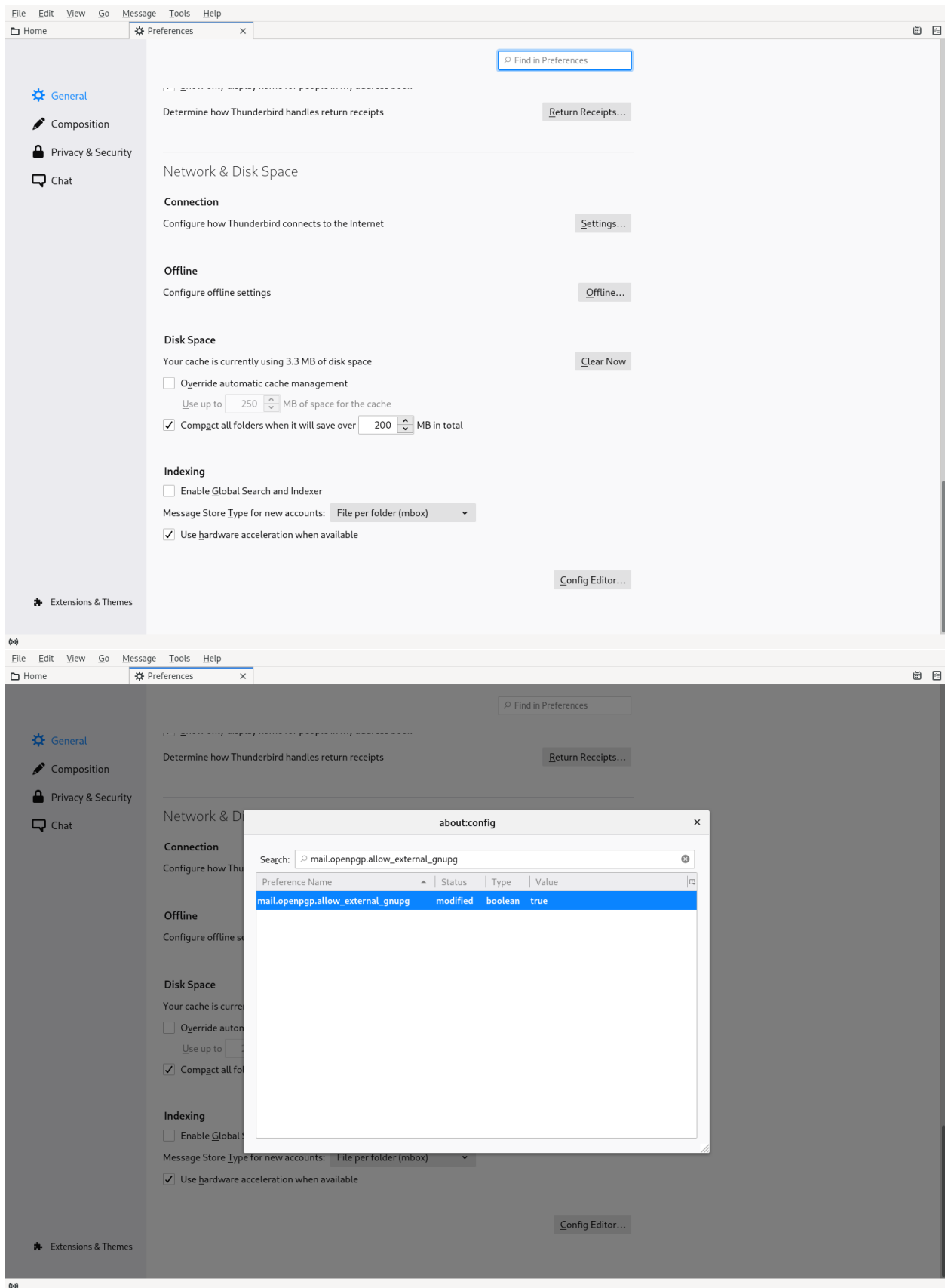
Note that, because this makes it easier to accept Split GPG's `qrexec` authorization prompts, it may decrease security if the user is not careful in reviewing presented prompts. This may also be inadvisable if there are multiple app qubes with Split GPG set up.

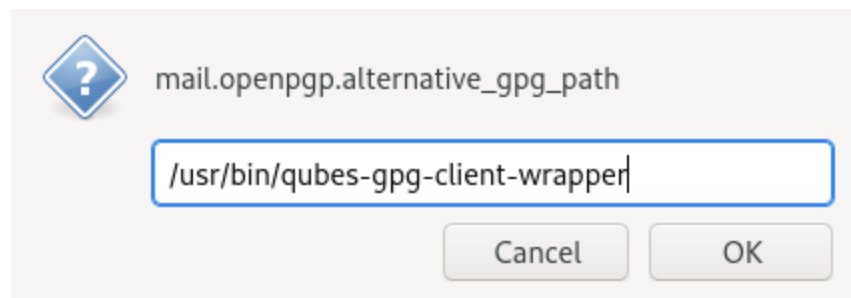
## Using Thunderbird

### Thunderbird 78 and higher

Starting with version 78, Thunderbird has a built-in PGP feature and no longer requires the Enigmail extension. For users coming from the Enigmail extension, the built-in functionality is more limited currently, including that **public keys must live in your ``work-email`` qube with Thunderbird rather than your offline ``work-gpg`` qube.**

In `work-email`, use the Thunderbird config editor (found at the bottom of preferences/options), and search for `mail.openpgp.allow_external_gnupg`. Switch the value to `true`. Still in config editor, search for `mail.openpgp.alternative_gpg_path`. Set its value to `/usr/bin/qubes-gpg-client-wrapper`. Restart Thunderbird after this change.



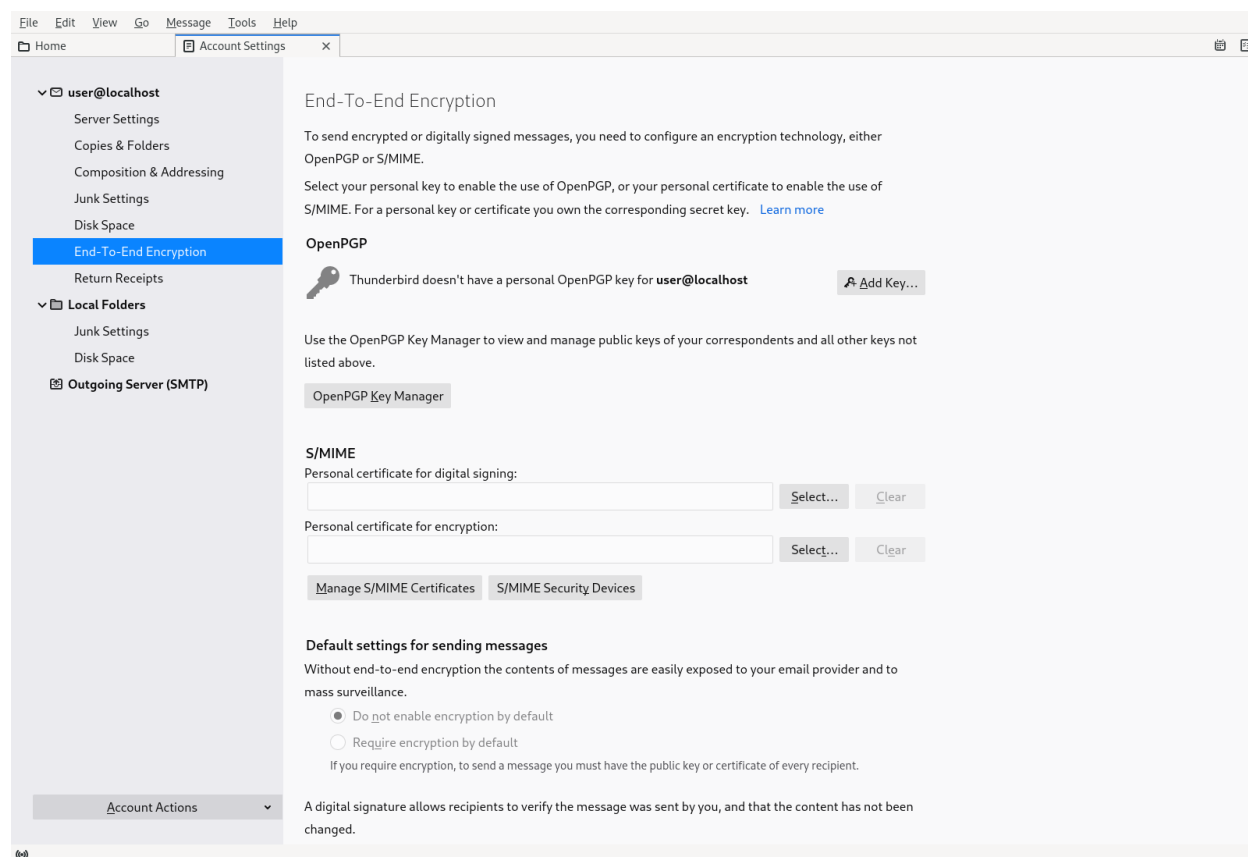


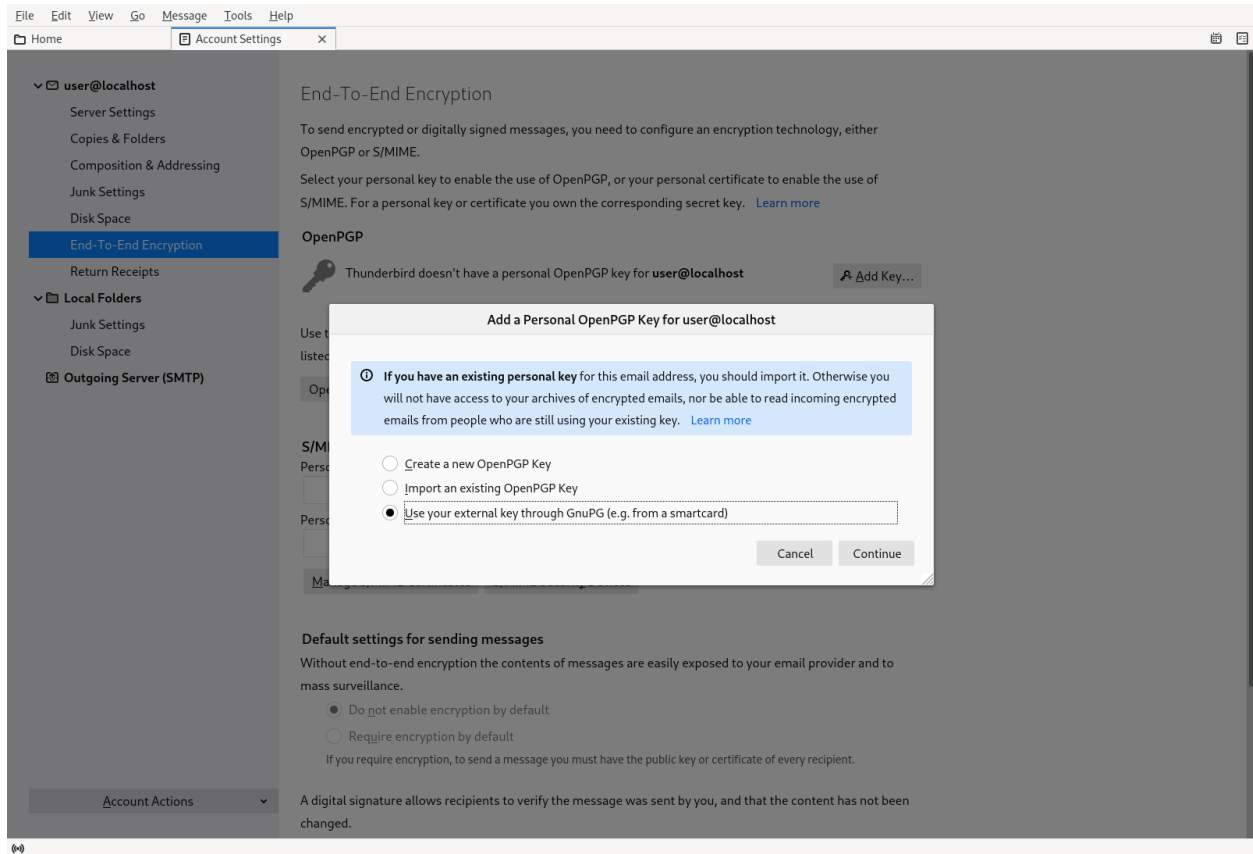
You need to obtain your key ID which should be **exactly 16 characters**. Enter the command `qubes-gpg-client-wrapper -K --keyid-format long`:

```
[user@work-email ~]$ qubes-gpg-client-wrapper -K --keyid-format long
/home/user/.gnupg/pubring.kbx
-----
sec   rsa2048/777402E6D301615C 2020-09-05 [SC] [expires: 2022-09-05]
      F7D2D4E922DFB7B2589AF3E9777402E6D301615C
uid           [ultimate] Qubes test <user@localhost>
ssb   rsa2048/370CE932085BA13B 2020-09-05 [E] [expires: 2022-09-05]
```

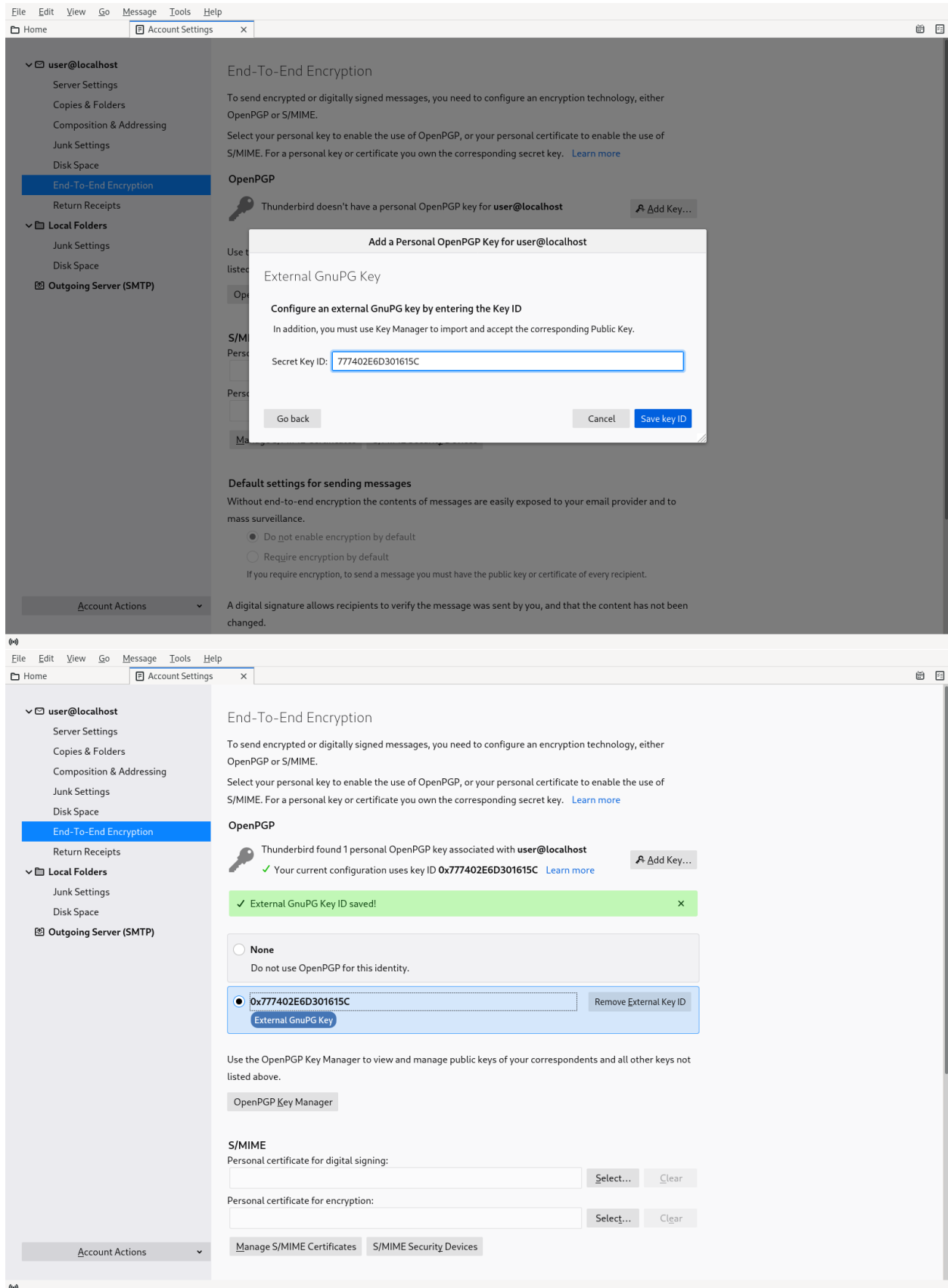
```
[user@work-email ~]$ qubes-gpg-client-wrapper --armor --export 777402E6D301615C >
↪ 777402E6D301615C.asc
```

Open the Account Settings and open the *End-to-End Encryption* tab of the respective email account. Click the *Add Key* button. You'll be offered the choice *Use your external key through GnuPG*. Select it and click Continue.

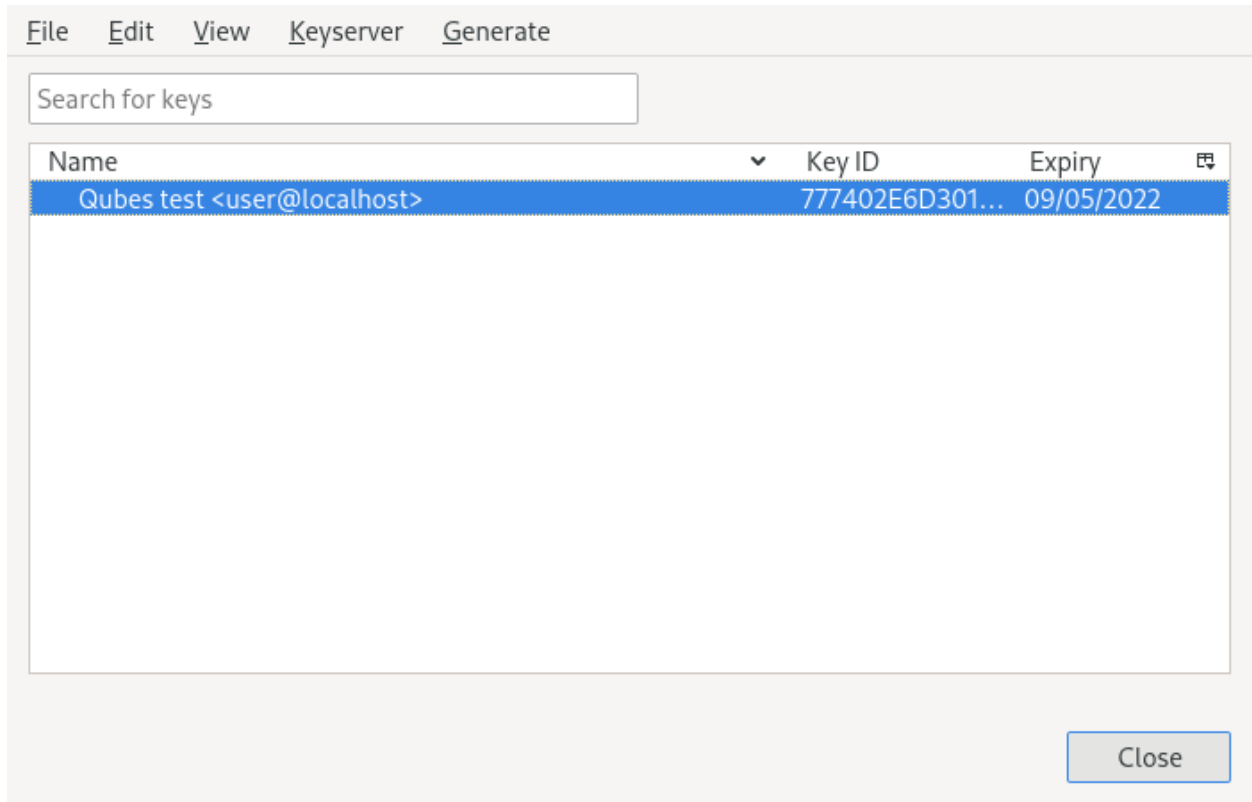




The key ID reference you would need here is 777402E6D301615C. Now paste or type the ID of the secret key that you would like to use. Be careful to enter it correctly, because your input isn't verified. Confirm to save this key ID. Now you can select the key ID to use.



This key ID will be used to digitally sign or send an encrypted message with your account. For this to work, Thunderbird needs a copy of your public key. At this time, Thunderbird doesn't fetch the public key from `/usr/bin/qubes-gpg-client-wrapper`, you must manually import it. Export the key as follow (assuming the key ID would be 777402E6D301615C):



Alleged Key Owner	Qubes test <user@localhost>
Type	public key
Fingerprint	F7D2 D4E9 22DF B7B2 589A F3E9 7774 02E6 D301 615C
Created	09/05/2020
Expiry	09/05/2022

Your Acceptance

Certifications

Structure

Do you accept this key for verifying digital signatures and for encrypting messages?

Avoid accepting a rogue key. Use a communication channel other than email to verify the fingerprint of your correspondent's key.

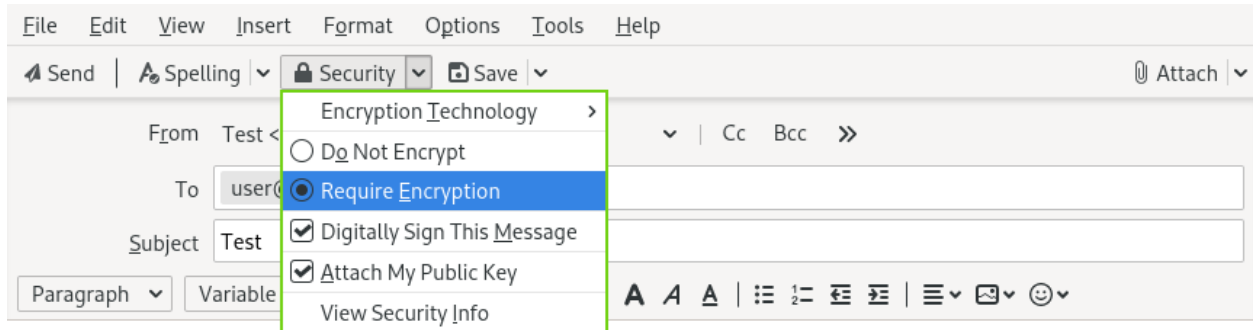
- ☐ No, reject this key.
- ☐ Not yet, maybe later.
- ☐ Yes, but I have not verified that it is the correct key.
- ☒ Yes, I've verified in person this key has the correct fingerprint.

OK

Use Thunderbird's Tools menu to open *OpenPGP Key Management*. In that window, use the File menu to access the *Import Public Key(s) From File* command. Open the file with your public key. After the import was successful, right click on the imported key in the list and select *Key Properties*. You must mark your own key as *Yes, I've verified in person this key has the correct fingerprint*.

Once this is done, you should be able to send an encrypted and signed email by selecting *Require Encryption* or *Digitally Sign This Message* in the compose menu *Options* or *Security* toolbar button. You can try it by sending an email to yourself.





This is a test message.

OpenPGP

For more details about using smart cards/Split GPG with Thunderbird PGP feature, please see [Thunderbird:OpenPGP:Smartcards](#) from which the above documentation is inspired.

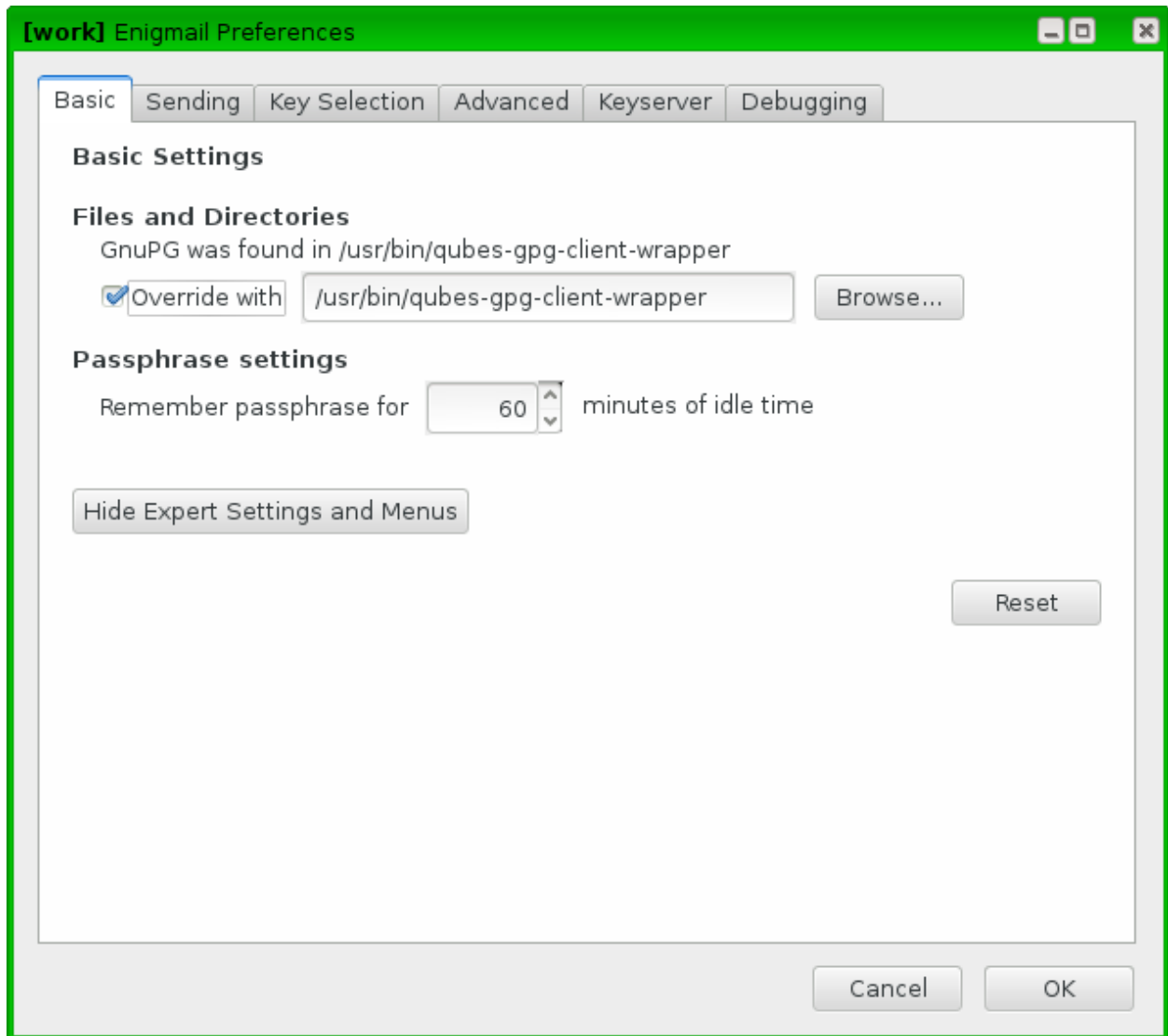
### Older Thunderbird versions

For Thunderbird versions below 78, the traditional Enigmail + Split GPG setup is required. It is recommended to set up and use `/usr/bin/qubes-gpg-client-wrapper`, as discussed above, in Thunderbird through the Enigmail addon.

**Warning:** Before adding any account, configuring Enigmail with `/usr/bin/qubes-gpg-client-wrapper` is **required**. By default, Enigmail will generate a default GPG key in `work-email` associated with the newly created Thunderbird account. Generally, it corresponds to the email used in `work-gpg` associated to your private key. In consequence, a new, separate private key will be stored in `work-email` but it *does not* correspond to your private key in `work-gpg`. Comparing the fingerprint or expiration date will show that they are not the same private key. In order to prevent Enigmail using this default generated local key in `work-email`, you can safely remove it.

On a fresh Enigmail install, your need to change the default Enigmail Junior Mode. Go to Thunderbird preferences and then privacy tab. Select Force using S/MIME and Enigmail. Then, in the preferences of Enigmail, make it

point to `/usr/bin/qubes-gpg-client-wrapper` instead of the standard GnuPG binary:



### Using Keybase with Split GPG

Keybase, a security focused messaging and file-sharing app with GPG integration, can be configured to use Split GPG.

The Keybase service does not preserve/pass the `QUBES_GPG_DOMAIN` environment variable through to underlying GPG processes, so it **must** be configured to use `/usr/bin/qubes-gpg-client-wrapper` (as discussed above) rather than `/usr/bin/qubes-gpg-client`.

The following command will configure Keybase to use `/usr/bin/qubes-gpg-client-wrapper` instead of its built-in GPG client:

```
$ keybase config set gpg.command /usr/bin/qubes-gpg-client-wrapper
```

Now that Keybase is configured to use `qubes-gpg-client-wrapper`, you will be able to use `keybase gpg select` to choose a GPG key from your backend GPG app qube and link that key to your Keybase identity.

## Using Git with Split GPG

Git can be configured to utilize Split GPG, something useful if you would like to contribute to the Qubes OS Project as every commit is required to be signed. The most basic `~/.gitconfig` file enabling Split GPG looks something like this.

```
[user]
    name = <YOUR_NAME>
    email = <YOUR_EMAIL_ADDRESS>
    signingKey = <YOUR_KEY_ID>

[pgp]
    program = qubes-gpg-client-wrapper
```

Your key id is the public id of your signing key, which can be found by running `qubes-gpg-client --list-keys`. In this instance, the key id is `E142F75A6B1B610E0E8F874FB45589245791CACB`.

```
[user@work-email ~]$ qubes-gpg-client --list-keys
/home/user/.gnupg/pubring.kbx
-----
pub   ed25519 2022-08-16 [C]
       E142F75A6B1B610E0E8F874FB45589245791CACB
uid           [ultimate] Qubes User <user@example.com>
sub   ed25519 2022-08-16 [S]
sub   cv25519 2022-08-16 [E]
sub   ed25519 2022-08-16 [A]
```

To sign commits, you now add the “-S” flag to your commit command, which should prompt for Split GPG usage. If you would like to automatically sign all commits, you can add the following snippet to `~/.gitconfig`.

```
[commit]
    gpgSign = true
```

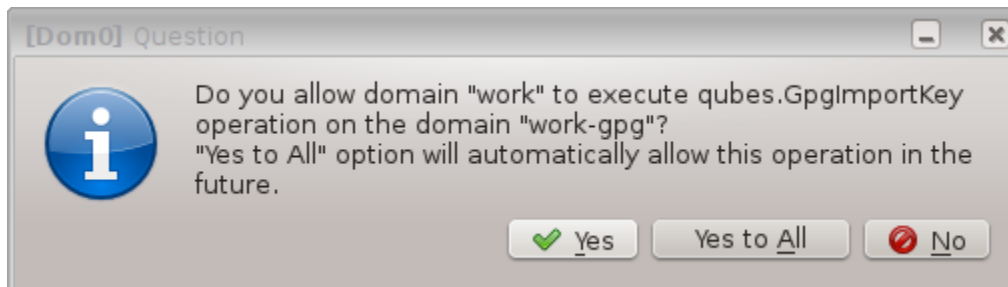
Lastly, if you would like to add aliases to sign and verify tags using the conventions the Qubes OS Project recommends, refer to the [code signing documentation](#).

## Importing public keys

Use `qubes-gpg-import-key` in the client app qube to import the key into the GPG backend VM.

```
[user@work-email ~]$ export QUBES_GPG_DOMAIN=work-gpg
[user@work-email ~]$ qubes-gpg-import-key ~/Downloads/marmarek.asc
```

A safe, unspoofable user consent dialog box is displayed.



Selecting “Yes to All” will add a line in the corresponding *RPC Policy* file.

## Advanced: Using Split GPG with Subkeys

Users with particularly high security requirements may wish to use Split GPG with [subkeys](#). However, this setup comes at a significant cost: It will be impossible to sign other people’s keys with the master secret key without breaking this security model. Nonetheless, if signing others’ keys is not required, then Split GPG with subkeys offers unparalleled security for one’s master secret key.

## Setup Description

In this example, the following keys are stored in the following locations (see below for definitions of these terms):

PGP Key(s)	VM Name
sec	vault
ssb	work-gpg
pub	work-email

- **sec** (master secret key) Depending on your needs, you may wish to create this as a **certify-only (C)** key, i.e., a key which is capable only of signing (a.k.a., “certifying”) other keys. This key may be created *without* an expiration date. This is for two reasons. First, the master secret key is never to leave the **vault** VM, so it is extremely unlikely ever to be obtained by an adversary (see below). Second, an adversary who *does* manage to obtain the master secret key either possesses the passphrase to unlock the key (if one is used) or does not. An adversary who *does* possess the passphrase can simply use it to legally extend the expiration date of the key (or remove it entirely). An adversary who does *not* possess the passphrase cannot use the key at all. In either case, an expiration date provides no additional benefit. By the same token, however, having a passphrase on the key is of little value. An adversary who is capable of stealing the key from your **vault** would almost certainly also be capable of stealing the passphrase as you enter it. An adversary who obtains the passphrase can then use it in order to change or remove the passphrase from the key. Therefore, using a passphrase at all should be considered optional. It is, however, recommended that a **revocation certificate** be created and safely stored in multiple locations so that the master keypair can be revoked in the (exceedingly unlikely) event that it is ever compromised.
- **ssb** (secret subkey) Depending on your needs, you may wish to create two different subkeys: one for **signing (S)** and one for **encryption (E)**. You may also wish to give these subkeys reasonable expiration dates (e.g., one year). Once these keys expire, it is up to you whether to *renew* these keys by extending the expiration dates or to create *new* subkeys when the existing set expires. On the one hand, an adversary who obtains any existing encryption subkey (for example) will be able to use it in order to decrypt all emails (for example) which were encrypted to that subkey. If the same subkey were to continue to be used—and its expiration date continually extended—only that one key would need to be stolen (e.g., as a result of the **work-gpg** VM being compromised; see below) in order to decrypt *all* of the user’s emails. If, on the other hand, each encryption subkey is used for at most approximately one year, then an adversary who obtains the secret subkey will be capable of decrypting at most approximately one year’s worth of emails. On the other hand, creating a new signing subkey each year without renewing (i.e., extending the expiration dates of) existing signing subkeys would mean that all of your old signatures would eventually read as “EXPIRED” whenever someone attempts to verify them. This can be problematic, since there is no consensus on how expired signatures should be handled. Generally, digital signatures are intended to last forever, so this is a strong reason against regularly retiring one’s signing subkeys.
- **pub** (public key) This is the complement of the master secret key. It can be uploaded to key servers (or otherwise publicly distributed) and may be signed by others.
- **vault** This is a network-isolated VM. The initial master keypair and subkeys are generated in this VM. The master secret key *never* leaves this VM under *any* circumstances. No files or text is *ever copied* or *pasted* into this VM under *any* circumstances.

- **work-gpg** This is a network-isolated VM. This VM is used *only* as the GPG backend for **work-email**. The secret subkeys (but *not* the master secret key) are *copied* from the **vault** VM to this VM. Files from less trusted VMs are *never copied* into this VM under *any* circumstances.
- **work-email** This VM has access to the mail server. It accesses the **work-gpg** VM via the Split GPG protocol. The public key may be stored in this VM so that it can be attached to emails and for other such purposes.

## Security Benefits

In the standard Split GPG setup, there are at least two ways in which the **work-gpg** VM might be compromised. First, an attacker who is capable of exploiting a hypothetical bug in **work-email**'s **MUA** could gain control of the **work-email** VM and send a malformed request which exploits a hypothetical bug in the GPG backend (running in the **work-gpg** VM), giving the attacker control of the **work-gpg** VM. Second, a malicious public key file which is imported into the **work-gpg** VM might exploit a hypothetical bug in the GPG backend which is running there, again giving the attacker control of the **work-gpg** VM. In either case, such an attacker might then be able to leak both the master secret key and its passphrase (if any is used, it would regularly be input in the **work-gpg** VM and therefore easily obtained by an attacker who controls this VM) back to the **work-email** VM or to another VM (e.g., the **netvm**, which is always untrusted by default) via the Split GPG protocol or other *covert channels*. Once the master secret key is in the **work-email** VM, the attacker could simply email it to himself (or to the world).

In the alternative setup described in this section (i.e., the subkey setup), even an attacker who manages to gain access to the **work-gpg** VM will not be able to obtain the user's master secret key since it is simply not there. Rather, the master secret key remains in the **vault** VM, which is extremely unlikely to be compromised, since nothing is ever copied or transferred into it. \* The attacker might nonetheless be able to leak the secret subkeys from the **work-gpg** VM in the manner described above, but even if this is successful, the secure master secret key can simply be used to revoke the compromised subkeys and to issue new subkeys in their place. (This is significantly less devastating than having to create a new *master* keypair.)

*In order to gain access to the ``vault`` VM, the attacker would require the use of, e.g., a general Xen VM escape exploit or a :ref:`signed, compromised package which is already installed in the template <user/templates/templates:trusting your templates>` upon which the ``vault`` VM is based.*

## Subkey Tutorials and Discussions

(Note: Although the tutorials below were not written with Qubes Split GPG in mind, they can be adapted with a few commonsense adjustments. As always, exercise caution and use your good judgment.)

- “OpenPGP in Qubes OS” on the [qubes-users mailing list](#)
- “Creating the Perfect GPG Keypair” by Alex Cabal
- “GPG Offline Master Key w/ smartcard” maintained by Abel Luck
- “Using GnuPG with QubesOS” by Alex

## Current limitations

- Current implementation requires importing of public keys to the vault domain. This opens up an avenue to attack the `gpg` running in the backend domain via a hypothetical bug in public key importing code. See ticket [#474](#) for more details and plans how to get around this problem, as well as the section on *using Split GPG with subkeys*.
- It doesn't solve the problem of allowing the user to know what is to be signed before the operation gets approved. Perhaps the GPG backend domain could start a disposable and have the to-be-signed document displayed there? To Be Determined.
- The Split GPG client will fail to sign or encrypt if the private key in the GnuPG backend is protected by a passphrase. It will give an `Inappropriate ioctl for device` error. Do not set passphrases for the private keys in the GPG backend domain. Doing so won't provide any extra security anyway, as explained in the introduction and in *using Split GPG with subkeys*. If you are generating a new key pair, or if you have a private key that already has a passphrase, you can use `gpg2 --edit-key <key_id>` then `passwd` to set an empty passphrase. Note that `pinentry` might show an error when you try to set an empty passphrase, but it will still make the change. (See [this StackExchange answer](#) for more information.) Note: The error shows only if you **do not** have graphical `pinentry` installed.

### 1.12.53 U2F proxy

The [Qubes U2F Proxy](#) is a secure proxy intended to make use of U2F two-factor authentication devices with web browsers without exposing the browser to the full USB stack, not unlike the *USB keyboard and mouse proxies* implemented in Qubes.

#### What is U2F?

U2F, which stands for “Universal 2nd Factor”, is a framework for authentication using hardware devices (U2F tokens) as “second factors”, i.e. *what you have* as opposed to *what you know*, like a passphrase. This additional control provides [good protection](#) in cases in which the passphrase is stolen (e.g. by phishing or keylogging). While passphrase compromise may not be obvious to the user, a physical device that cannot be duplicated must be stolen to be used outside of the owner's control. Nonetheless, it is important to note at the outset that U2F cannot guarantee security when the host system is compromised (e.g. a malware-infected operating system under an adversary's control).

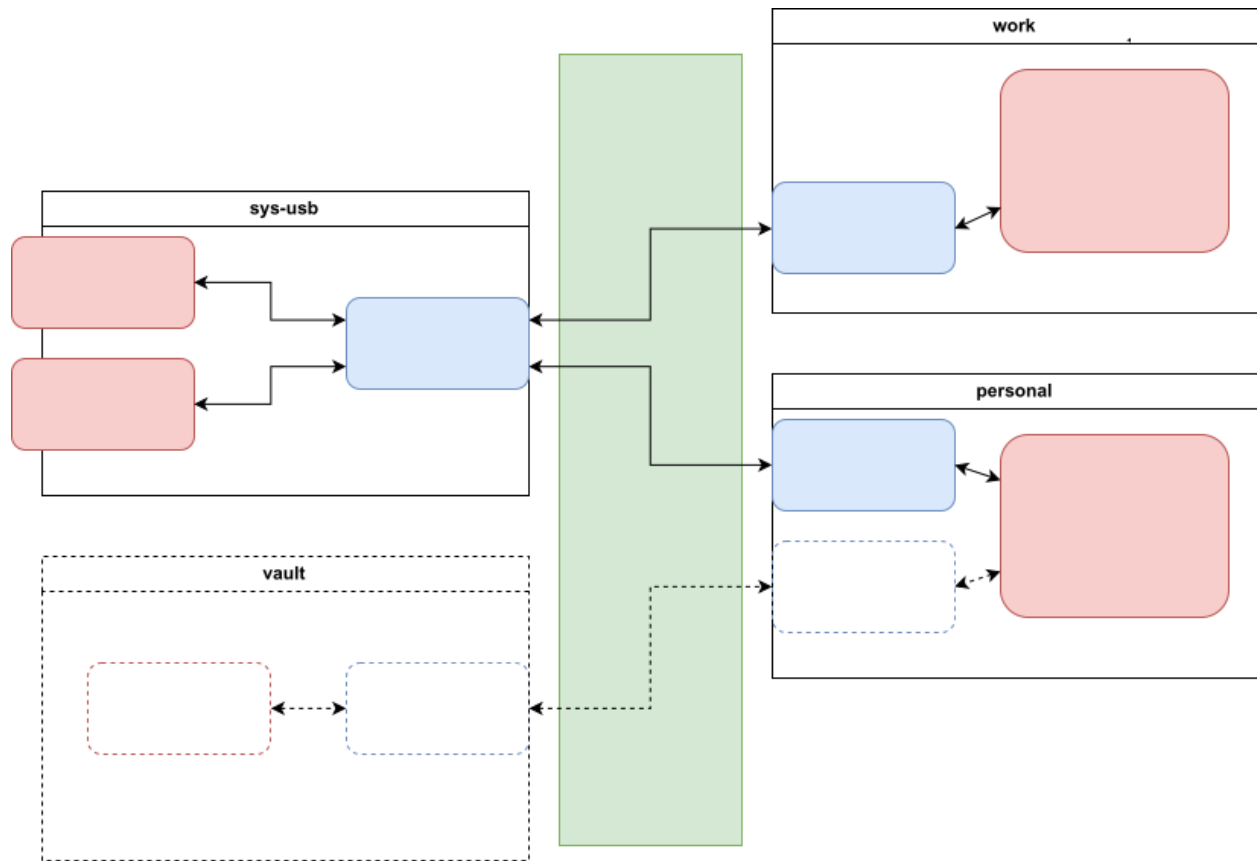
The U2F specification defines protocols for multiple layers from USB to the browser API, and the whole stack is intended to be used with web applications (most commonly websites) in browsers. In most cases, tokens are USB dongles. The protocol is very simple, allowing the devices to store very little state inside (so the tokens may be reasonably cheap) while simultaneously authenticating a virtually unlimited number of services (so each person needs only one token, not one token per application). The user interface is usually limited to a single LED and a button that is pressed to confirm each transaction, so the devices themselves are also easy to use.

Currently, the most common form of two-step authentication consists of a numeric code that the user manually types into a web application. These codes are typically generated by an app on the user's smartphone or sent via SMS. By now, it is well-known that this form of two-step authentication is vulnerable to phishing and man-in-the-middle attacks due to the fact that the application requesting the two-step authentication code is typically not itself authenticated by the user. (In other words, users can accidentally give their codes to attackers because they do not always know who is really requesting the code.) In the U2F model, by contrast, the browser ensures that the token receives valid information about the web application requesting authentication, so the token knows which application it is authenticating (for details, see [here](#)). Nonetheless, [some attacks are still possible](#) even with U2F (more on this below).

## The Qubes approach to U2F

In a conventional setup, web browsers and the USB stack (to which the U2F token is connected) are all running in the same monolithic OS. Since the U2F model assumes that the browser is trustworthy, any browser in the OS is able to access any key stored on the U2F token. The user has no way to know which keys have been accessed by which browsers for which services. If any of the browsers are compromised, it should be assumed that all of the token's keys have been compromised. (This problem can be mitigated, however, if the U2F device has a special display to show the user what's being authenticated.) Moreover, since the USB stack is in the same monolithic OS, the system is vulnerable to attacks like [BadUSB](#).

In Qubes OS, by contrast, it is possible to securely compartmentalise the browser in one qube and the USB stack in another so that they are always kept separate from each other. The Qubes U2F Proxy then allows the token connected to the USB stack in one qube to communicate with the browser in a separate qube. We operate under the assumption that the USB stack is untrusted from the point of view of the browser and also that the browser is not to be trusted blindly by the token. Therefore, the token is never in the same qube as the browser. Our proxy forwards only the data necessary to actually perform the authentication, leaving all unnecessary data out, so it won't become a vector of attack. This is depicted in the diagram below (click for full size).



The Qubes U2F Proxy has two parts: the frontend and the backend. The frontend runs in the same qube as the browser and presents a fake USB-like HID device using `uhid`. The backend runs in `sys-usb` and behaves like a browser. This is done using the `u2flib_host` reference library. All of our code was written in Python. The standard `qrexec` policy is responsible for directing calls to the appropriate domains.

The `vault` qube with a dashed line in the bottom portion of the diagram depicts future work in which we plan to implement the Qubes U2F Proxy with a software token in an isolated qube rather than a physical hardware token. This is similar to the manner in which [Split GPG](#) allows us to emulate the smart card model without physical smart cards.

One very important assumption of U2F is that the browser verifies every request sent to the U2F token — in particular,

that the web application sending an authentication request matches the application that would be authenticated by answering that request (in order to prevent, e.g., a phishing site from sending an authentication request for your bank's site). With the WebUSB feature in Chrome, however, a malicious website can [bypass](#) this safeguard by connecting directly to the token instead of using the browser's U2F API.

The Qubes U2F Proxy also prevents this class of attacks by implementing an additional verification layer. This verification layer allows you to enforce, for example, that the web browser in your `twitter` qube can only access the U2F key associated with `https://twitter.com`. This means that if anything in your `twitter` qube were compromised — the browser or even the OS itself — it would still not be able to access the U2F keys on your token for any other websites or services, like your email and bank accounts. This is another significant security advantage over monolithic systems. (For details and instructions, see the [Advanced usage](#) section below.)

For even more protection, you can combine this with the [Qubes firewall](#) to ensure, for example, that the browser in your banking qube accesses only one website (your bank's website). By configuring the Qubes firewall to prevent your banking qube from accessing any other websites, you reduce the risk of another website compromising the browser in an attempt to bypass U2F authentication.

## Installation

These instructions assume that there is a `sys-usb` qube that holds the USB stack, which is the default configuration in most Qubes OS installations.

In `dom0`:

```
$ sudo qubes-dom0-update qubes-u2f-dom0
$ qvm-service --enable work qubes-u2f-proxy
```

The above assumes a `work` qube in which you would like to enable `u2f`. Repeat the `qvm-service` command for all qubes that should have the proxy enabled. Alternatively, you can add `qubes-u2f-proxy` in VM settings -> Services in the Qube Manager of each qube you would like to enable the service.

In Fedora templates:

```
$ sudo dnf install qubes-u2f
```

In Debian templates:

```
$ sudo apt install qubes-u2f
```

As usual with software updates, shut down the templates after installation, then restart `sys-usb` and all qubes that use the proxy. After that, you may use your U2F token (but see [Browser support](#) below).

## Advanced usage: per-qube key access

If you are using Qubes 4.0, you can further compartmentalise your U2F keys by restricting each qube's access to specific keys. For example, you could make it so that your `twitter` qube (and, therefore, all web browsers in your `twitter` qube) can access only the key on your U2F token for `https://twitter.com`, regardless of whether any of the web browsers in your `twitter` qube or the `twitter` qube itself are compromised. If your `twitter` qube makes an authentication request for your bank website, it will be denied at the Qubes policy level.

To enable this, create a file in `dom0` named `/etc/qubes/policy.d/30-user-u2fproxy.policy` with the following content:

```
policy.RegisterArgument +u2f.Authenticate sys-usb @anyvm allow target=dom0
```



Next, empty the contents of `/etc/qubes-rpc/policy/u2f.Authenticate` so that it is a blank file. Do not delete the file itself. (If you do, the default file will be recreated the next time you update, so it will no longer be empty.) Finally, follow your web application's instructions to enroll your token and use it as usual. (This enrollment process depends on the web application and is in no way specific to Qubes U2F.)

The default model is to allow a qube to access all and only the keys that were enrolled by that qube. For example, if your `banking` qube enrolls your banking key, and your `twitter` qube enrolls your Twitter key, then your `banking` qube will have access to your banking key but not your Twitter key, and your `twitter` qube will have access to your Twitter key but not your banking key.

### Non-default USB qube name

If your USB qube is named differently than `sys-usb`, then do the following in the appropriate template(s):

```
systemctl enable qubes-u2fproxy@USB_QUBE.service
systemctl disable qubes-u2fproxy@sys-usb.service
```

Replace `USB_QUBE` with the actual USB qube name.

Do not forget to change the `sys-usb` qube name in the policy `/etc/qubes/policy.d/30-user-u2fproxy.policy`.

### Template and browser support

The large number of possible combinations of template (Fedora 27, 28; Debian 8, 9) and browser (multiple Google Chrome versions, multiple Chromium versions, multiple Firefox versions) made it impractical for us to test every combination that users are likely to attempt with the Qubes U2F Proxy. In some cases, you may be the first person to try a particular combination. Consequently (and as with any new feature), users will inevitably encounter bugs. We ask for your patience and understanding in this regard. As always, please *report any bugs you encounter*.

#### 1.12.54 YubiKey

“The YubiKey is a hardware authentication device manufactured by Yubico to protect access to computers, networks, and online services that supports one-time passwords (OTP), public-key cryptography, and authentication, and the Universal 2nd Factor (U2F) and FIDO2 protocols[1] developed by the FIDO Alliance.” ([Wikipedia](#))

### General usage in Qubes OS

Most use cases for the YubiKey can be achieved exactly as described by the manufacturer or other instructions found online. One usually just needs to attach the YubiKey to the corresponding app qube to get the same result (see the documentation on how to use *USB devices* in Qubes OS accordingly). The recommended way for using U2F in Qubes is described [here](#).

### Multi-factor login for Qubes OS

You can use a YubiKey to enhance the user authentication in Qubes. The following instructions explain how to setup the YubiKey as an *additional* way to login.

After setting it up, you can login by providing both - a password typed in via keyboard *and* the YubiKey plugged in. Someone eavesdropping your login attempt would not be able to login by only observing and remembering your password. Stealing your YubiKey would not suffice to login either. Only if an attacker has both, the password and the Yubikey, it would be possible to login (it is thus called *Multi-factor authentication*).

The following instructions keep your current login password untouched and recommends to define a new, additional password that is used in combination with the YubiKey only. This ensures that you a) do not accidentally lock yourself out during setup and b) you do not need to fear [shoulder surfing](#) so much (i.e. by not using your standard login password in public).

## Setup login with YubiKey

To use the YubiKey for multi-factor authentication you need to

- install software for the YubiKey,
- configure the YubiKey for the [Challenge-Response](#) mode,
- store the password for YubiKey Login and the Challenge-Response secret in dom0,
- enable YubiKey authentication for every service you want to use it for.

All these requirements are described below, step by step.

1. Install YubiKey software in the template on which your USB VM is based. Without this software the challenge-response mechanism is not working. For Fedora.

```
sudo dnf install ykpers
```

For Debian.

```
sudo apt-get install yubikey-personalization
```

Shut down your template. Then, either reboot your USB VM (so changes inside the template take effect in your USB app qube) or install the packages inside your USB VM as well if you would like to avoid rebooting it.

2. Install [qubes-app-yubikey](#) in dom0. This provides the program to authenticate with password and YubiKey.

```
sudo qubes-dom0-update qubes-yubikey-dom0
```

3. Configure your YubiKey for challenge-response HMAC-SHA1 mode. This can be done on any qube, e.g. a disposable (you need to [attach the YubiKey](#) to this app qube though) or directly on the sys-usb vm. You need to (temporarily) install the package “yubikey-personalization-gui” and run it by typing `yubikey-personalization-gui` in the command line.

- In the program go to Challenge-Response,
- select HMAC-SHA1,
- choose Configuration Slot 2,
- optional: enable Require user input (button press) (recommended),
- use fixed 64 bit input for HMAC-SHA1 mode,
- insert the YubiKey (if not done already) and make sure that it is attached to the vm,
- press Write Configuration once you are ready.

We will refer the Secret Key (20 bytes hex) as AESKEY.

- It is recommended to keep a backup of your AESKEY in an offline VM used as a vault.
- Consider keeping a backup of your AESKEY on paper and storing it in a safe place.
- If you have multiple YubiKeys for backup purposes (in case a yubikey gets lost, stolen or breaks) you can write the same settings into other YubiKeys. You can choose “Program multiple YubiKeys” in the program, make sure to select Same secret for all keys in this case.

4. Paste your AESKEY into `/etc/qubes/yk-keys/yk-secret-key.hex` in dom0.
5. As mentioned before, you need to define a new password that is only used in combination with the YubiKey. You can write this password in plain text into `/etc/qubes/yk-keys/yk-login-pass` in dom0. This is considered safe as dom0 is ultimately trusted anyway. However, if you prefer you can paste a hashed password instead into `/etc/qubes/yk-keys/yk-login-pass-hashed.hex` in dom0. You can calculate your hashed password using the following two commands. First run the following command to store your password in a temporary variable `password`. (This way your password will not leak to the terminal command history file.)

```
read -r password
```

Now run the following command to calculate your hashed password.

```
echo -n "$password" | openssl dgst -sha1 | cut -f2 -d ' '
```

6. To enable multi-factor authentication for a service, you need to add

```
auth include yubikey
```

to the corresponding service file in `/etc/pam.d/` in dom0. This means, if you want to enable the login via YubiKey for `xscreensaver` (the default screen lock program), you add the line at the beginning of `/etc/pam.d/xscreensaver`. If you want to use the login for a `tty` shell, add it to `/etc/pam.d/login`. Add it to `/etc/pam.d/lightdm` if you want to enable the login for the default display manager and so on. It is important, that `auth include yubikey` is added at the beginning of these files, otherwise it will most likely not work.

7. Adjust the USB VM name in case you are using something other than the default `sys-usb` by editing `/etc/qubes/yk-keys/yk-vm` in dom0.

## Usage

When you want to authenticate

1. plug your YubiKey into an USB slot,
2. enter the password associated with the YubiKey,
3. press Enter and
4. press the button of the YubiKey, if you configured the confirmation (it will blink).

When everything is ok, your screen will be unlocked.

In any case you can still use your normal login password, but do it in a secure location where no one can snoop your password.

## Optional: Enforce YubiKey Login

Edit `/etc/pam.d/yubikey` (or appropriate file if you are using other screen locker program) and remove `default=ignore` so the line looks like this.

```
auth [success=done] pam_exec.so expose_authtok quiet /usr/bin/yk-auth
```

## Optional: Locking the screen when YubiKey is removed

Look into it You can setup your system to automatically lock the screen when you unplug your YubiKey. This will require creating a simple qrexec service which will expose the ability to lock the screen to your USB VM, and then adding a udev hook to actually call that service.

In dom0:

1. First configure the qrexec service. Create `/etc/qubes-rpc/custom.LockScreen` with a simple command to lock the screen. In the case of `xscreensaver` (used in `Xfce`) it would be:

```
DISPLAY=:0 xscreensaver-command -lock
```

2. Then make `/etc/qubes-rpc/custom.LockScreen` executable.

```
sudo chmod +x /etc/qubes-rpc/custom.LockScreen
```

3. Allow your USB VM to call that service. Assuming that it's named `sys-usb` it would require creating `/etc/qubes-rpc/policy/custom.LockScreen` with:

```
sys-usb dom0 allow
```

In your USB VM:

3. Create udev hook. Store it in `/rw/config` to have it persist across VM restarts. For example name the file `/rw/config/yubikey.rules`. Add the following line:

```
ACTION=="remove", SUBSYSTEM=="usb", ENV{ID_SECURITY_TOKEN}=="1", RUN+="/usr/bin/
↳qrexec-client-vm dom0 custom.LockScreen"
```

4. Ensure that the udev hook is placed in the right place after VM restart. Append to `/rw/config/rc.local`:

```
ln -s /rw/config/yubikey.rules /etc/udev/rules.d/
udevadm control --reload
```

5. Then make `/rw/config/rc.local` executable.

```
sudo chmod +x /rw/config/rc.local
```

6. For changes to take effect, you need to call this script manually for the first time.

```
sudo /rw/config/rc.local
```

If you use KDE, the command(s) in first step would be different:

```
# In the case of USB VM being autostarted, it will not have direct access to D-Bus
# session bus, so find its address manually:
kde_pid=`pidof kdeinit4`
export `cat /proc/$kde_pid/environ|grep -ao 'DBUS_SESSION_BUS_ADDRESS=[[:graph:]]*`
qdbus org.freedesktop.ScreenSaver /ScreenSaver Lock
```

### 1.12.55 How to install software in dom0

**Warning:** Installing software in dom0 is for advanced users only. Doing so has the potential to compromise your entire Qubes OS installation. Exercise extreme caution.

#### Security

Since there is no networking in dom0, any bugs discovered in dom0 desktop components (e.g., the window manager) are unlikely to pose a problem for Qubes, since none of the third-party software running in dom0 is accessible from VMs or the network in any way. Nonetheless, since software running in dom0 can potentially exercise full control over the system, it is important to install only trusted software in dom0.

The install/update process is split into two phases: *resolve and download* and *verify and install*. The *resolve and download* phase is handled by the UpdateVM. (The role of UpdateVM can be assigned to any VM in the Qube Manager, and there are no significant security implications in this choice. By default, this role is assigned to the FirewallVM.) After the UpdateVM has successfully downloaded new packages, they are sent to dom0, where they are verified and installed. This separation of duties significantly reduces the attack surface, since all of the network and metadata processing code is removed from the TCB.

Although this update scheme is far more secure than directly downloading updates in dom0, it is not invulnerable. For example, there is nothing that the Qubes OS Project can feasibly do to prevent a malicious RPM from exploiting a hypothetical bug in the cryptographic signature verification operation. At best, we could switch to a different distro or package manager, but any of them could be vulnerable to the same (or a similar) attack. While we could, in theory, write a custom solution, it would only be effective if Qubes repos included all of the regular template distro's updates, and this would be far too costly for us to maintain.

#### How to update dom0

See [How to Update](#).

#### How to install a specific package

To install additional packages in dom0 (usually not recommended):

```
$ sudo qubes-dom0-update anti-evil-maid
```

You may also pass the `--enablerepo=` option in order to enable optional repositories (see yum configuration in dom0). However, this is only for advanced users who really understand what they are doing. You can also pass commands to dnf using `--action=...`

#### How to downgrade a specific package

**WARNING:** Downgrading a package can expose your system to security vulnerabilities.

1. Download an older version of the package:

```
sudo qubes-dom0-update package-version
```

Dnf will say that there is no update, but the package will nonetheless be downloaded to dom0.

2. Downgrade the package:

```
sudo dnf downgrade package-version
```

## How to re-install a package

You can re-install in a similar fashion to downgrading.

1. Download the package:

```
sudo qubes-dom0-update package
```

Dnf will say that there is no update, but the package will nonetheless be downloaded to dom0.

2. Re-install the package:

```
sudo dnf reinstall package
```

Note that `dnf` will only re-install if the installed and downloaded versions match. You can ensure they match by either updating the package to the latest version, or specifying the package version in the first step using the form `package-version`.

## How to uninstall a package

If you've installed a package such as `anti-evil-maid`, you can remove it with the following command:

```
sudo dnf remove anti-evil-maid
```

## Testing repositories

If you wish to install updates that are still in *testing*, you must enable the appropriate testing repositories.

**Note:** The following repos are in dom0. For template testing repos, see [here](#).

- `qubes-dom0-current-testing` – testing packages that will eventually land in the stable (`current`) repository
- `qubes-dom0-security-testing` – a subset of `qubes-dom0-current-testing` that contains packages that qualify as security fixes
- `qubes-dom0-unstable` – packages that are not intended to land in the stable (`qubes-dom0-current`) repository; mostly experimental debugging packages

To temporarily enable any of these repos, use the `--enablerepo=<repo-name>` option. Example commands:

```
sudo qubes-dom0-update --enablerepo=qubes-dom0-current-testing
sudo qubes-dom0-update --enablerepo=qubes-dom0-security-testing
sudo qubes-dom0-update --enablerepo=qubes-dom0-unstable
```

To enable or disable any of these repos permanently, change the corresponding `enabled` value to 1 in `/etc/yum.repos.d/qubes-dom0.repo`.

For testing new templates, please see [here](#).

## Contributed package repository

Please see *installing contributed packages*.

## Kernel upgrade

This section describes upgrading the kernel in dom0 and domUs.

### dom0

The packages `kernel` and `kernel-latest` are for dom0.

In the `current` repository:

- `kernel`: an older LTS kernel that has passed Qubes *testing* (the default dom0 kernel)
- `kernel-latest`: the latest release from kernel.org that has passed Qubes *testing* (useful for *troubleshooting newer hardware*)

In the `current-testing` repository:

- `kernel`: the latest LTS kernel from kernel.org at the time it was built.
- `kernel-latest`: the latest release from kernel.org at the time it was built.

### domU

The packages `kernel-qubes-vm` and `kernel-latest-qubes-vm` are for domUs. See *Managing VM kernel* for more information.

## Example

(Note that the following example enables the unstable repo.)

```
sudo qubes-dom0-update --enablerepo=qubes-dom0-unstable kernel kernel-qubes-vm
```

If the update process does not automatically do it (you should see it mentioned in the CLI output from the update command), you may need to manually rebuild the EFI or grub config depending on which your system uses.

## EFI

Replace the example version numbers with the one you are upgrading to.

```
sudo dracut -f /boot/efi/EFI/qubes/initramfs-4.14.35-1.pvops.qubes.x86_64.img 4.14.35-1.
↪pvops.qubes.x86_64
```

### Grub2

```
sudo grub2-mkconfig -o /boot/grub2/grub.cfg
```

Reboot required.

If you wish to upgrade to a kernel that is not available from the repos, then there is no easy way to do so, but it may still be possible if you're willing to do a lot of work yourself.

### Changing default kernel

This section describes changing the default kernel in dom0. It is sometimes needed if you have upgraded to a newer kernel and are having problems booting, for example. The procedure varies depending on if you are booting with UEFI or grub. On the next kernel update, the default will revert to the newest.

### EFI

```
sudo nano /boot/efi/EFI/qubes/xen.cfg
```

In the [global] section at the top, change the default= line to match one of the three boot entries listed below. For example:

```
default=4.19.67-1.pvops.qubes.x86_64
```

### Grub2

```
sudo nano /etc/default/grub
[update the following two lines, add if needed]
GRUB_DISABLE_SUBMENU=false
GRUB_SAVEDEFAULT=true
[save and exit nano]
sudo grub2-mkconfig -o /boot/grub2/grub.cfg
```

Then, reboot. Once the grub menu appears, choose “Advanced Options for Qubes (with Xen hypervisor)”. Next, the top menu item (for example, “Xen hypervisor, version 4.8.5-9.fc25”). Select the kernel you want as default, and it will be remembered for next boot.

### Updating over Tor

Requires installed [Whonix](#).

Go to Qubes VM Manager -> System -> Global Settings. See the UpdateVM setting. Choose your desired Whonix-Gateway ProxyVM from the list. For example: sys-whonix.

```
Qubes VM Manager -> System -> Global Settings -> UpdateVM -> sys-whonix
```



### 1.12.56 Volume backup and revert

With Qubes, it is possible to revert one of a VM's storage volumes to a previous state using the automatic snapshot that is normally saved every time a VM is shutdown. (Note that this is a different, lower level activity than the *Backup, Restoration, and Migration* process.)

In Qubes, when you create a new VM, it's volumes are stored in one of the system's *Storage Pools*. On pool creation, a `revisions_to_keep` default value is set for the entire pool. (For a pool creation example, see *Storing app qubes on Secondary Drives*.) Thereafter, each volume associated with a VM that is stored in this pool inherits the pool default `revisions_to_keep`.

For the private volume associated with a VM named `vmname`, you may inspect the value of `revisions_to_keep` from the `dom0` CLI as follows:

```
qvm-volume info vmname:private
```

The output of the above command will also display the “Available revisions (for revert)” at the bottom. For a very large volume in a small pool, `revisions_to_keep` should probably be set to the minimum value of 1 to minimize the possibility of the pool being accidentally filled up by snapshots. For a smaller volume for which you would like to have the future option of reverting, `revisions_to_keep` should probably be set to at least 2. To set `revisions_to_keep` for this same VM / volume example:

```
qvm-volume config vmname:private revisions_to_keep 2
```

With the VM stopped, you may revert to an older snapshot of the private volume from the the above list of “Available revisions (for revert)”, where the last item on the list with the largest integer is the most recent snapshot:

```
qvm-volume revert vmname:private <revision>
```

### 1.12.57 Standalones and HVMs

A *standalone* is a type of qube that is created by cloning a *template*. Unlike templates, however, standalones do not supply their root filesystems to other qubes. Examples of situations in which standalones can be useful include:

- Qubes used for development (dev environments often require a lot of specific packages and tools)
- Qubes used for installing untrusted packages. Normally, you install digitally signed software from Red Hat/Fedora repositories, and it's reasonable that such software has non malicious *installation* scripts (rpm pre/post scripts). However, when you would like to install some packages from less trusted sources, or unsigned, then using a dedicated (untrusted) standalone might be a better way.

Meanwhile, a *Hardware-assisted Virtual Machine (HVM)*, also known as a “Fully-Virtualized Virtual Machine,” utilizes the virtualization extensions of the host CPU. These are typically contrasted with Paravirtualized (PV) VMs.

HVMs allow you to create qubes based on any OS for which you have an installation ISO, so you can easily have qubes running Windows, \*BSD, or any Linux distribution. You can also use HVMs to run “live” distros.

By default, every qube runs in PVH mode (which has security advantages over both PV and HVM), except for those with attached PCI devices, which run in HVM mode. See [here](#) for a discussion of the switch from PV to HVM and [here](#) for the announcement about the change to using PVH as default.

The standalone/template distinction and the HVM/PV/PVH distinctions are orthogonal. The former is about root filesystem inheritance, whereas the latter is about the virtualization mode. In practice, however, it is most common for standalones to be HVMs and for HVMs to be standalones. Hence, this page covers both topics.

## Creating a standalone

You can create a standalone in the Qube Manager by selecting the “Type” of “Standalone qube copied from a template” or “Empty standalone qube (install your own OS).”

Alternatively, from the dom0 command line:

```
qvm-create --class StandaloneVM --label <YOUR_COLOR> --property virt_mode=hvm <NEW_  
↪STANDALONE_NAME>
```

(Note: Technically, `virt_mode=hvm` is not necessary for every standalone. However, it makes sense if you want to use a kernel from within the qube.)

## Updating standalones

When you create a standalone from a template, the standalone is a complete clone of the template, including the entire filesystem. After the moment of creation, the standalone becomes completely independent from the template. Therefore, the standalone will not be updated merely by updating the template from which it was originally cloned. Rather, it must be updated as an independent qube. See [How to Update](#).

## Creating an HVM

### Using the GUI

In Qube Manager, select “Create new qube” from the Qube menu, or select the “Create a new qube” button. In the “create new qube” dialog box set Type to “Empty standalone qube (install your own OS)”. If “install system from device” is selected (which it is by default), then `virt_mode` will be set to `hvm` automatically. Otherwise, open the newly-created qube’s Settings GUI and, in the “Advanced” tab, select HVM in the virtualization mode drop-down list. Also, make sure “Kernel” is set to `(none)` on the same tab.

### Command line

Qubes are template-based (i.e., *app qubes* by default, so you must set the `--class StandaloneVM` option to create a standalone. The name and label color used below are for illustration purposes.

```
qvm-create my-new-vm --class StandaloneVM --property virt_mode=hvm --property kernel='' -  
↪-label=green
```

If you receive an error like this one, then you must first enable VT-x in your BIOS:

```
libvirt.libvirtError: invalid argument: could not find capabilities for arch=x86_64
```

Make sure that you give the new qube adequate memory to install and run.

## Installing an OS in an HVM

You will have to boot the qube with the installation media “attached” to it. You may either use the GUI or use command line instructions. At the command line you can do this in three ways:

1. If you have the physical CD-ROM media and an optical disc drive:

```
qvm-start <YOUR_HVM> --cdrom=/dev/cdrom
```

2. If you have an ISO image of the installation media located in dom0:

```
qvm-start <YOUR_HVM> --cdrom=dom0:/usr/local/iso/<YOUR_INSTALLER.ISO>
```

3. If you have an ISO image of the installation media located in a qube (the qube where the media is located must be running):

```
qvm-start <YOUR_HVM> --cdrom=<YOUR_OTHER_QUBE>:/home/user/<YOUR_INSTALLER.ISO>
```

For security reasons, you should *never* copy untrusted data to dom0.

Next, the qube will start booting from the attached installation media, and you can start installation. Whenever the installer wants to “reboot the system” it actually shuts down the qube, and Qubes won’t automatically start it. You may have to restart the qube several times in order to complete installation (as is the case with Windows 7 installations). Several invocations of the `qvm-start` command (as shown above) might be needed.

## Setting up networking for HVMs

Just like standard app qubes, an HVM gets a fixed IP addresses centrally assigned by Qubes. Normally, Qubes agent scripts (or services on Windows) running within each app qube are responsible for setting up networking within the qube according to the configuration created by Qubes (through *keys* exposed by dom0 to the qube). Such centrally-managed networking infrastructure allows for *advanced networking configurations*.

A generic HVM such as a standard Windows or Ubuntu installation, however, has no Qubes agent scripts running inside it initially and thus requires manual configuration of networking so that it matches the values assigned by Qubes.

Even though we do have a small DHCP server that runs inside the HVM’s untrusted stub domain to make the manual network configuration unnecessary for many qubes, this won’t work for most modern Linux distributions, which contain Xen networking PV drivers (but not Qubes tools), which bypass the stub-domain networking. (Their net frontends connect directly to the net backend in the *net qube*.) In this instance, our DHCP server is not useful.

In order to manually configure networking in a qube, one should first find out the IP/netmask/gateway assigned to the particular qube by Qubes. This can be seen, e.g., in the Qube Manager in the qube’s properties:

Alternatively, one can use the `qvm-ls -n` command to obtain the same information (IP/netmask/gateway).

The DNS IP addresses are 10.139.1.1 and 10.139.1.2. There is *opt-in support* for IPv6 forwarding.

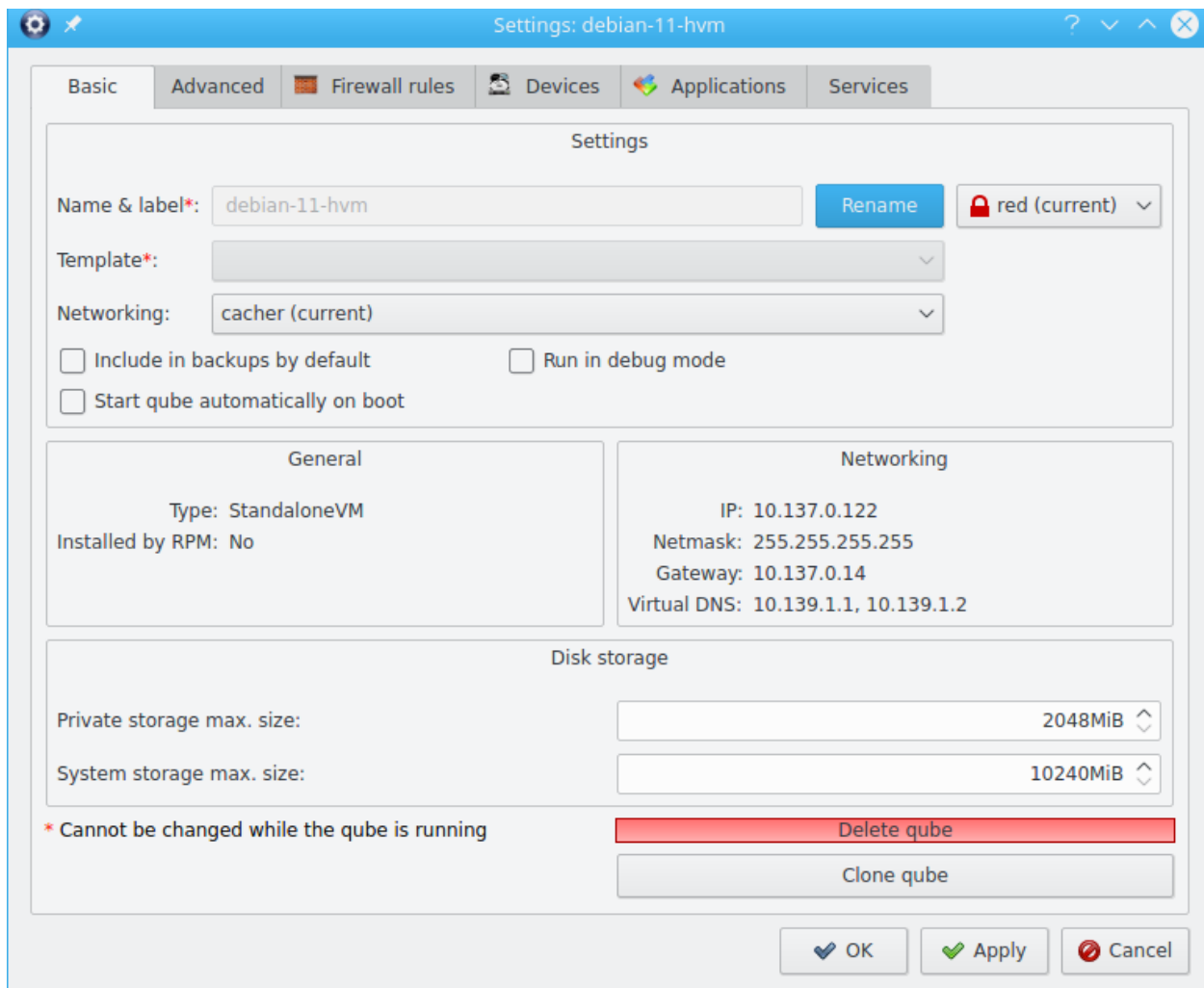


Fig. 10: r4.0-manager-networking-config.png

## Using template-based HVMs

Qubes allows HVMs to share a common root filesystem from a select template. This mode can be used for any HVM (e.g., FreeBSD running in an HVM).

In order to create an HVM template, you use the following command, suitably adapted:

```
qvm-create --class TemplateVM <YOUR_HVM_TEMPLATE_NAME> --property virt_mode=HVM --
↪property kernel='' -l <YOUR_COLOR>
```

Set memory as appropriate and install the OS into this template in the same way you would install it into a normal HVM. Generally, you should install in to the first “system” disk. (Resize it as needed before starting installation.)

You can then create a new qube using the new template. If you use this Template as is, then any HVMs based on it will effectively be disposables. All file system changes will be wiped when the HVM is shut down.

Please see [this page](#) for specific advice on installing and using Windows-based templates.

## Cloning HVMs

Just like normal app qubes, HVMs can also be cloned either using the command `qvm-clone` or via the Qube Manager’s “Clone VM” option in the right-click menu.

The cloned qube will get identical root and private images and will essentially be identical to the original qube, except that it will get a different MAC address for the networking interface:

```
[joanna@dom0 ~]$ qvm-prefs my-new-vm
autostart          D  False
backup_timestamp    U
debug              D  False
default_dispvm      D  None
default_user        D  user
gateway            D
gateway6           D
include_in_backups  -  False
installed_by_rpm    D  False
ip                 D  10.137.0.122
ip6                D  fd09:24ef:4179::a89:7a
kernel             -
kernelopts          D  nopat
klass               D  StandaloneVM
label              -  red
mac                D  00:16:3e:5e:6c:00
management_dispvm  D  default-mgmt-dvm
maxmem             D  0
memory             -  1000
name                -  my-new-vm
netvm               -  sys-firewall
provides_network    -  False
qid                -  122
qrexec_timeout      D  60
shutdown_timeout    D  60
start_time          D
stubdom_mem         U
stubdom_xid         D  -1
```

(continues on next page)

(continued from previous page)

```

updateable      D  True
uuid            -  54387f94-8617-46b0-8806-0c18bc387f34
vcpus           D  2
virt_mode       -  hvm
visible_gateway D  10.137.0.14
visible_gateway6 D  fd09:24ef:4179::a89:e
visible_ip       D  10.137.0.122
visible_ip6      D  fd09:24ef:4179::a89:7a
visible_netmask  D  255.255.255.255
xid             D  -1

```

```
[joanna@dom0 ~]$ qvm-clone my-new-vm my-new-vm-copy
```

```
/.../
```

```

[joanna@dom0 ~]$ qvm-prefs my-new-vm-copy
autostart      D  False
backup_timestamp U
debug          D  False
default_dispvm D  None
default_user   D  user
gateway        D
gateway6       D
include_in_backups - False
installed_by_rpm D  False
ip             D  10.137.0.137
ip6            D  fd09:24ef:4179::a89:89
kernel         -
kernelopts     D  nopat
klass          D  StandaloneVM
label          -  red
mac            D  00:16:3e:5e:6c:00
management_dispvm D  default-mgmt-dvm
maxmem         D  0
memory         -  1000
name           -  my-new-vm-copy
netvm          -  sys-firewall
provides_network - False
qid            -  137
qrexec_timeout D  60
shutdown_timeout D  60
start_time     D
stubdom_mem    U
stubdom_xid    D  -1
updateable     D  True
uuid           -  9ad109a9-d95a-4e03-b977-592f8424f42b
vcpus          D  2
virt_mode      -  hvm
visible_gateway D  10.137.0.14
visible_gateway6 D  fd09:24ef:4179::a89:e
visible_ip     D  10.137.0.137
visible_ip6    D  fd09:24ef:4179::a89:89

```

(continues on next page)

(continued from previous page)

```
visible_netmask    D 255.255.255.255
xid               D  -1
```

Note that the MAC addresses differ between those two otherwise identical qubes. The IP addresses assigned by Qubes will also be different, of course, to allow networking to function properly:

```
[joanna@dom0 ~]$ qvm-ls -n
```

NAME	STATE	NETVM	IP	IPBACK	GATEWAY
my-new-hvm	Halted	sys-firewall	10.137.0.122	-	10.137.0.14
my-new-hvm-clone	Halted	sys-firewall	10.137.0.137	-	10.137.0.14

If, for any reason, you would like to make sure that the two qubes have the same MAC address, you can use `qvm-prefs` to set a fixed MAC address:

```
[joanna@dom0 ~]$ qvm-prefs my-new-vm-copy -s mac 00:16:3E:5E:6C:05
[joanna@dom0 ~]$ qvm-prefs my-new-vm-copy
name           : my-new-vm-copy
label          : green
type           : HVM
netvm          : firewallvm
updateable?    : True
installed by RPM? : False
include in backups: False
dir            : /var/lib/qubes/appvms/my-new-vm-copy
config         : /var/lib/qubes/appvms/my-new-vm-copy/my-new-vm-copy.conf
pcidevs        : []
root img       : /var/lib/qubes/appvms/my-new-vm-copy/root.img
private img    : /var/lib/qubes/appvms/my-new-vm-copy/private.img
vcpus          : 4
memory         : 512
maxmem         : 512
MAC            : 00:16:3E:5E:6C:05
debug          : off
default user   : user
qrexec_installed : False
qrexec timeout : 60
drive         : None
timezone      : localtime
```

## Assigning PCI devices to HVMs

HVMs (including Windows qubes) can be *assigned PCI devices* just like normal app qubes. For example, you can assign a USB controller to a Windows qube, and you should be able to use various devices that require Windows software, such as phones, electronic devices that are configured via FTDI, etc.

One problem at the moment, however, is that after the whole system gets suspended into S3 sleep and subsequently resumed, some attached devices may stop working and should be restarted within the qube. This can be achieved under a Windows HVM by opening the Device Manager, selecting the actual device (such as a USB controller), ‘Disabling’ the device, and then ‘Enabling’ the device again. This is illustrated in the screenshot below:

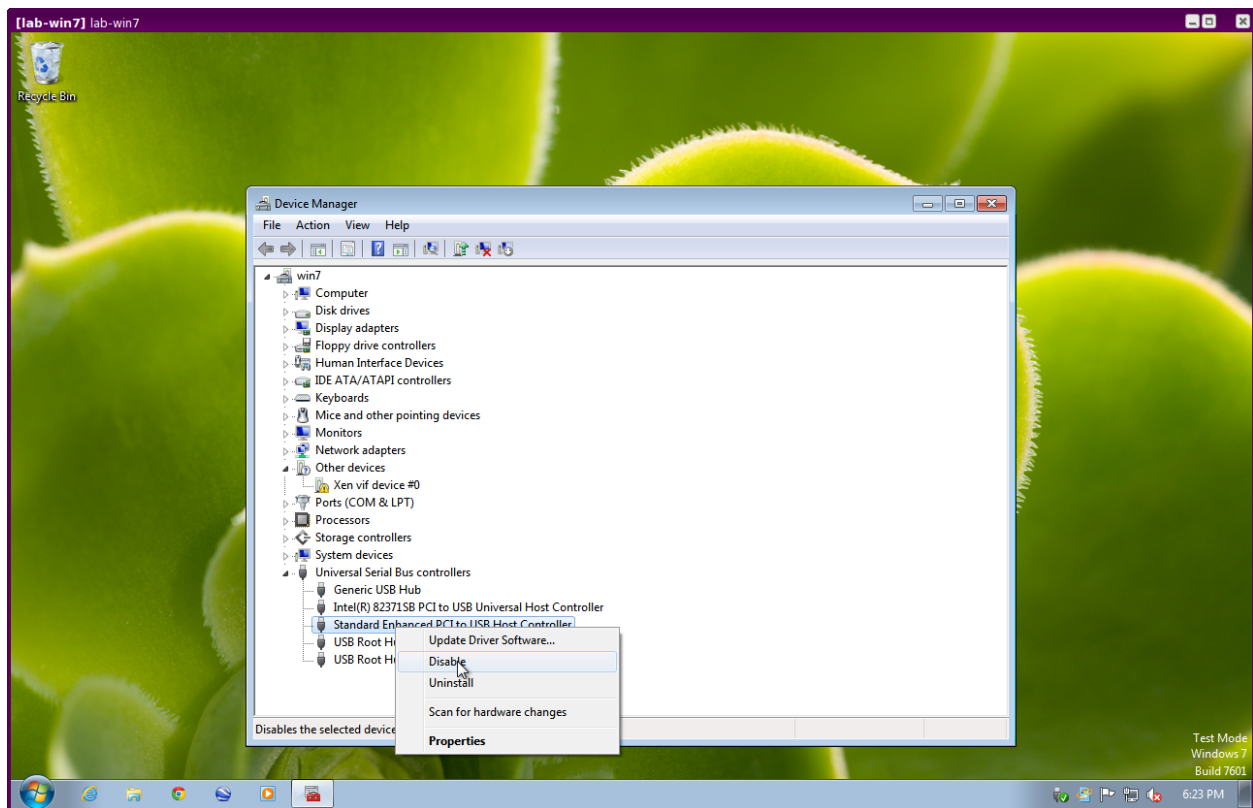


Fig. 11: r2b1-win7-usb-disable.png



## Converting VirtualBox VMs to Qubes HVMs

You can convert any VirtualBox VM to a Qubes HVM using this method.

For example, Microsoft provides [free 90-day evaluation VirtualBox VMs for browser testing](#).

About 60 GB of disk space is required for conversion. Use an external hard drive if needed. The final `root.img` size is 40 GB.

In a Debian app qube, install `qemu-utils` and `unzip`:

```
sudo apt install qemu-utils unzip
```

In a Fedora app qube:

```
sudo dnf install qemu-img
```

Unzip VirtualBox zip file:

```
unzip *.zip
```

Extract OVA tar archive:

```
tar -xvf *.ova
```

Convert vmdk to raw:

```
qemu-img convert -O raw *.vmdk win10.raw
```

Copy the root image file from the originating qube (here called `untrusted`) to a temporary location in `dom0`, typing this in a `dom0` terminal:

```
qvm-run --pass-io untrusted 'cat "/media/user/externalhd/win10.raw"' > /home/user/win10-  
↪root.img
```

From within `dom0`, create a new HVM (here called `win10`) with the root image we just copied to `dom0` (change the amount of RAM in GB as you wish):

```
qvm-create --property=virt_mode=hvm --property=memory=4096 --property=kernel='' --label_  
↪red --standalone --root-move-from /home/user/win10-root.img win10
```

Start `win10`:

```
qvm-start win10
```

## Optional ways to get more information

Filetype of OVA file:

```
file *.ova
```

List files of OVA tar archive:

```
tar -tf *.ova
```

List filetypes supported by `qemu-img`:

```
qemu-img -h | tail -n1
```

## Further reading

Other documents related to HVMs:

- [Windows VMs](#)
- [Linux HVM Tips](#)

## 1.12.58 Config files

### Qubes-specific VM config files

These files are placed in `/rw`, which survives a VM restart. That way, they can be used to customize a single VM instead of all VMs based on the same template. The scripts here all run as root.

- `/rw/config/rc.local` - script runs at VM startup. Good place to change some service settings, replace config files with its copy stored in `/rw/config`, etc. Example usage:

```
# Store bluetooth keys in /rw to keep them across VM restarts
rm -rf /var/lib/bluetooth
ln -s /rw/config/var-lib-bluetooth /var/lib/bluetooth
```

```
# Add entry to /etc/hosts
echo '127.0.0.1 example.com' >> /etc/hosts
```

- `/rw/config/qubes-ip-change-hook` - script runs in NetVM after every external IP change and on “hardware” link status change.
- In ProxyVMs (or app qubes with `qubes-firewall` service enabled), scripts placed in the following directories will be executed in the listed order followed by `qubes-firewall-user-script` at start up. Good place to write custom firewall rules.

```
/etc/qubes/qubes-firewall.d
/rw/config/qubes-firewall.d
/rw/config/qubes-firewall-user-script
```

- `/rw/config/suspend-module-blacklist` - list of modules (one per line) to be unloaded before system goes to sleep. The file is used only in a VM with PCI devices attached. Intended for use with problematic device drivers.
- In NetVMs/ProxyVMs, scripts placed in `/rw/config/network-hooks.d` will be ran when configuring Qubes interfaces. For each script, the `command`, `vif`, `vif_type` and `ip` is passed as arguments (see `/etc/xen/scripts/vif-route-qubes`). For example, consider a PV app qube `work` with IP `10.137.0.100` and `sys-firewall` as NetVM. Assuming it's Xen domain id is arbitrary 12 then, the following script located at `/rw/config/network-hooks.d/hook-100.sh` in `sys-firewall`:

```
#!/bin/bash

command="$1"
vif="$2"
vif_type="$3"
```

(continues on next page)

(continued from previous page)

```

ip="$4"

if [ "$ip" == '10.137.0.100' ]; then
    case "$command" in
        online)
            ip route add 192.168.0.100 via 10.137.0.100
            ;;
        offline)
            ip route del 192.168.0.100
            ;;
    esac
fi

```

will be executed with arguments `online vif12.0 vif 10.137.0.100` when starting work. Please note that in case of an HVM, the script will be called twice - once with `vif_type vif`, then with `vif_type vif_ioemu` (and different interface names). As long as the `ioemu` interface exists, it should be preferred (up to the hook script). When the VM decides to use a PV interface (`vif_type vif`), the `ioemu` one will be unplugged.

Note that scripts need to be executable (`chmod +x`) to be used.

Also, take a look at *bind-dirs* for instructions on how to easily modify arbitrary system files in an app qube and have those changes persist.

## GUI and audio configuration in dom0

The GUI configuration file `/etc/qubes/guid.conf` is one of a few not managed by `qubes-prefs` or the Qubes Manager tool. Sample config (included in default installation):

```

# Sample configuration file for Qubes GUI daemon
# For syntax go https://www.hyperrealm.com/libconfig/libconfig_manual.html

global: {
    # default values
    #allow_fullscreen = false;
    #override_redirect_protection = true;
    #allow_utf8_titles = false;
    #secure_copy_sequence = "Ctrl-Shift-c";
    #secure_paste_sequence = "Ctrl-Shift-v";
    #windows_count_limit = 500;
    #audio_low_latency = true;
    #log_level = 1;
    #trayicon_mode = "border1";
    #startup_timeout = 45;
};

# most of setting can be set per-VM basis

VM: {
    work: {
        allow_utf8_titles = true;
    };
    video-vm: {

```

(continues on next page)

(continued from previous page)

```

    allow_fullscreen = true;
};
};

```

Currently supported settings:

- **allow\_fullscreen** - allow VM to request its windows to go fullscreen (without any colorful frame). **Note:** Regardless of this setting, you can always put a window into fullscreen mode in Xfce4 using the trusted window manager by right-clicking on a window's title bar and selecting "Fullscreen". This functionality should still be considered safe, since a VM window still can't voluntarily enter fullscreen mode. The user must select this option from the trusted window manager in dom0. To exit fullscreen mode from here, press **alt + space** to bring up the title bar menu again, then select "Leave Fullscreen".
- **allow\_utf8\_titles** - allow the use of UTF-8 in window titles; otherwise, non-ASCII characters are replaced by an underscore.
- **secure\_copy\_sequence** and **secure\_paste\_sequence** - key sequences used to trigger secure copy and paste.
- **audio\_low\_latency** - force low-latency audio mode (about 40ms compared to 200-500ms by default). Note that this will cause much higher CPU usage in dom0. It's enabled by default, disabling it may save CPU in dom0.
- **trayicon\_mode** - defines the trayicon coloring mode. Options are - **bg** - color full icon background to the VM color - **border1** - add 1px border at the icon edges - **border2** - add 1px border 1px from the icon edges - **tint** - tint icon to the VM color, can be used with additional modifiers (you can enable multiple of them) - **tint+border1**, **tint+border2** - same as tint, but also add a border - **tint+saturation50** - same as tint, but reduce icon saturation by 50% - **tint+whitehack** - same as tint, but change white pixels (0xffffffff) to almost-white (0xfefefe)
- **log\_level** - defines the log options logs can take. It can have a value of 0 (only errors), 1 (some basic messages), and 2 (debug).
- **startup\_timeout** - The timeout for startup.

### 1.12.59 Secondary storage

Suppose you have a fast but small primary SSD and a large but slow secondary HDD. You want to store a subset of your app qubes on the HDD.

#### Instructions

Qubes 4.0 is more flexible than earlier versions about placing different VMs on different disks. For example, you can keep templates on one disk and app qubes on another, without messy symlinks.

These steps assume you have already created a separate [volume group](#) and [thin pool](#) (not thin volume) for your HDD. See also [this example](#) if you would like to create an encrypted LVM pool (but note you can use a single logical volume if preferred, and to use the **-T** option on **lvcreate** to specify it is thin). You can find the commands for this example applied to Qubes at the bottom of this R4.0 section.

First, collect some information in a dom0 terminal:

```

sudo pvs
sudo lvs

```

Take note of the VG and thin pool names for your HDD, then register it with Qubes:

```
# <pool_name> is a freely chosen pool name
# <vg_name> is LVM volume group name
# <thin_pool_name> is LVM thin pool name
qvm-pool --add <pool_name> lvm_thin -o volume_group=<vg_name>,thin_pool=<thin_pool_name>,
↪revisions_to_keep=2
```

Now, you can create qubes in that pool:

```
qvm-create -P <pool_name> --label red <vmname>
```

It isn't possible to directly migrate an existing qube to the new pool, but you can clone it there, then remove the old one:

```
qvm-clone -P <pool_name> <sourceVMname> <cloneVMname>
qvm-remove <sourceVMname>
```

If that was a template, or other qube referenced elsewhere (NetVM or such), you will need to adjust those references manually after moving. For example:

```
qvm-prefs <appvmname_based_on_old_template> template <new_template_name>
```

In theory, you can still use file-based disk images ("file" pool driver), but it lacks some features such as you won't be able to do backups without shutting down the qube.

## Example HDD setup

Assuming the secondary hard disk is at /dev/sdb (it will be completely erased), you can set it up for encryption by doing in a dom0 terminal (use the same passphrase as the main Qubes disk to avoid a second password prompt at boot):

```
sudo cryptsetup luksFormat --hash=sha512 --key-size=512 --cipher=aes-xts-plain64 --
↪verify-passphrase /dev/sdb
sudo blkid /dev/sdb
```

Note the device's UUID (in this example "b209..."), we will use it as its luks name for auto-mounting at boot, by doing:

```
sudo nano /etc/crypttab
```

And adding this line (change both "b209..." for your device's UUID from blkid) to crypttab:

```
luks-b20975aa-8318-433d-8508-6c23982c6cde UUID=b20975aa-8318-433d-8508-6c23982c6cde none
```

Reboot the computer so the new luks device appears at /dev/mapper/luks-b209... and we can then create its pool, by doing this on a dom0 terminal (substitute the b209... UUIDs with yours):

First create the physical volume

```
sudo pvcreate /dev/mapper/luks-b20975aa-8318-433d-8508-6c23982c6cde
```

Then create the LVM volume group, we will use for example "qubes" as the :

```
sudo vgcreate qubes /dev/mapper/luks-b20975aa-8318-433d-8508-6c23982c6cde
```

And then use "poolhd0" as the (LVM thin pool name):

```
sudo lvcreate -T -n poolhd0 -l +100%FREE qubes
```

Finally we will tell Qubes to add a new pool on the just created thin pool

```
qvm-pool --add poolhd0_qubes lvm_thin -o volume_group=qubes,thin_pool=poolhd0,revisions_
↪to_keep=2
```

By default VMs will be created on the main Qubes disk (i.e. a small SSD), to create them on this secondary HDD do the following on a dom0 terminal:

```
qvm-create -P poolhd0_qubes --label red untrusted-hdd
```

### 1.12.60 RPC policies

*Qubes OS 4.1 introduced a new `qrexec` policy system and policy format. Please see [this article](#) for details.*

This document explains the basics of RPC policies in Qubes. For more information, see [Qrexec: command execution in VMs](#).

Here's an example of an RPC policy file in dom0:

```
[user@dom0 user ~]$ cat /etc/qubes-rpc/policy/qubes.FileCopy
(...)
@tag:work    @default    ask
@tag:work    @tag:work    allow
@tag:work    @anyvm      deny
@anyvm       @tag:work    deny
@anyvm       @anyvm      ask
```

It has three columns (from left to right): source, destination, and permission. Each row is a rule. For example, the second row says that we're **allowed** (third column) to copy a file (since this is the policy file for `qubes.FileCopy`) **from** (first column) any VM tagged with “work” **to** (second column) any VM tagged with “work”. In other words, all the VMs tagged with “work” are allowed to copy files to each other without any prompts. (If the third column were “ask” instead of “allow”, there would be prompts. I.e., we would be **asked** to approve the action, instead of it always being **allowed**.)

Now, the whole policy file is parsed from top to bottom. As soon as a rule is found that matches the action being evaluated, parsing stops. We can see what this means by looking at the third row. It says that we're **denied** from attempting to copy a file **from** any VM tagged with “work” **to** any VM whatsoever. (That's what the `@anyvm` keyword means – literally any VM in the system, except for dom0). But, wait a minute, didn't we just say (in the second row) that all the VMs tagged with work are **allowed** to copy files to each other? That's exactly right. The second and third rows contradict each other, but that's intentional. Since we know that parsing goes from top to bottom (and stops at the first match), we intentionally put the second row above the third row so that it would take precedence. This is how we create a policy that says: “VMs tagged with ‘work’ are allowed to copy files to each other but not to any *other* VMs (i.e., not to VMs that *aren't* tagged with ‘work’).”

When an operation is initiated with a specific target, e.g. `qvm-copy-to-vm other_work_vm some_file` the policy mechanism looks for a row matching `source_work_vm other_work_vm PERMISSION`. In this case, assuming both VMs have the work tag, the second row would match, and the operation would be allowed without any prompts. When an operation is initiated without a specific target, e.g. `qvm-copy some_file`, the policy mechanism looks for a row matching `source_work_vm @default PERMISSION`. In this case, the first row indicates that the user should be prompted for the destination. The list of destination VMs in the prompt is filtered to only include VMs that are valid as per the policy (so in this example, only other work VMs would be listed). If the first row was commented out, the second row would not match (the `@default` placeholder is not included in `@tag:work`) but the third row would match (the `@default` placeholder is included in `@anyvm`). The

`qvm-copy` operation would therefore terminate immediately with the message `Request refused`, without prompting the user with a list of valid destination VMs, and only `qvm-copy-to-vm` operations with valid destinations would be allowed.

The fourth row says that we're **denied** from copying files **from** any VM in the system **to** any VM tagged with "work". Again, since parsing goes from top to bottom, this doesn't mean that no files can ever be copied from *any* VM to a VM tagged with "work". Rather, it means that only VMs that match an earlier rule can do so (in this case, only VMs tagged with "work").

The fifth and final row says that we're **asked** (i.e., prompted) to copy files **from** any VM in the system **to** any VM in the system. (This rule was already in the policy file by default. We added the first four.) Note that it wouldn't make sense to add any rules after this one, since every possible pair of VMs will match the `@anyvm @anyvm` pattern. Therefore, parsing will always stop at this rule, and no rules below it will ever be evaluated.

All together, the four rules we added say that all VMs tagged with "work" are allowed to copy files to each other; however, they're denied from copying files to other VMs (without the "work" tag), and other VMs (without the "work" tag) are denied from copying files to them. The fifth rule means that the user gets prompted for any situation not already covered.

Further details about how this system works can be found in [Qrexec: command execution in VMs](#).

(**Note** : the ```$``` character is deprecated in `qrexec` keywords – please use ```@``` instead (e.g. ```@anyvm```). For more information, see the bulletin [here](#).)

## 1.12.61 USB qubes

A USB qube acts as a secure handler for potentially malicious USB devices, preventing them from coming into contact with `dom0` (which could otherwise be fatal to the security of the whole system). It thereby mitigates some of the *security risks of using USB devices*. Nonetheless, we strongly recommend carefully reading the *security warning on USB input devices* before proceeding.

With a USB qube, every time you connect an untrusted USB device to a USB port managed by that USB controller, you will have to attach it to the qube in which you wish to use it (if different from the USB qube itself).

If you opted to allow the Qubes installer to create a USB qube for you during the installation process, then you should already have a working USB qube, and no further action should be required. However, if you do not have a USB qube, wish to remove the one you have, or have run into some other related problem, this page can help.

## USB keyboards

If you use a USB keyboard, there is a high risk of locking yourself out of your system when experimenting with USB qubes. For example, if a USB qube takes over your sole USB controller (to which your USB keyboard is connected), then your keyboard will no longer be able to control `dom0`. This will prevent you from performing many essential tasks, such as entering your decryption and login passphrases, rendering your system unusable until you reinstall. This section covers various options for addressing this problem.

In general, PS/2 keyboards are preferable to USB keyboards. However, many newer computer models lack PS/2 ports. Moreover, while most laptops use PS/2 connections for the keyboard internally, some use USB. (Check yours by examining the output of the `lsusb` command.) If you have a PS/2 port but still wish to use a USB keyboard, then having a backup PS/2 keyboard handy can be useful in case you accidentally lock yourself out of your system.

## How to create a USB qube for use with a USB keyboard

If you're reading this section, it's likely because the installer did not allow you to create a USB qube automatically because you're using a USB keyboard. This section will explain how to create a USB qube that you can use with your USB keyboard. This section assumes that you have only a single USB controller. If you have more than one USB controller, see *how to enable a USB keyboard on a separate USB controller*.

First, make sure you have the latest `qubes-mgmt-salt-dom0-virtual-machines` package by *updating dom0*. Then, enter the following command in dom0:

```
sudo qubesctl state.sls qvm.usb-keyboard
```

This command will take care of all required configuration, including creating a USB qube if not already present. Note, however, that this setup will expose dom0 to USB devices while you are entering your LUKS passphrase. While only input devices (keyboards, mice, etc.) are initialized at this stage, users are advised to physically disconnect other devices from the system during this vulnerable window in order to minimize the risk.

To undo these changes, see *how to remove a USB qube*.

If you wish to perform only a subset of this configuration (for example, you do not wish to enable the USB keyboard during boot), see the manual instructions below.

## Manual setup for USB keyboards

In order to use a USB keyboard, you must first attach it to a USB qube, then give that qube permission to pass keyboard input to dom0. Edit the `qubes.InputKeyboard` policy file in dom0, which is located here:

```
/etc/qubes-rpc/policy/qubes.InputKeyboard
```

Add a line like this one to the top of the file:

```
sys-usb dom0 allow
```

(Change `sys-usb` to your desired USB qube.)

You can now use your USB keyboard to log in to your dom0 user account (after LUKS decryption).

You can set up your system so that there's a confirmation prompt each time the USB keyboard is connected. However, this will effectively disable your USB keyboard for dom0 user account login and the screen locker, so **don't do this if you want to log into and unlock your device with a USB keyboard!** If you're sure you wish to proceed, change the previous line to:

```
sys-usb dom0 ask,default_target=dom0
```

If you wish to use a USB keyboard to enter your LUKS passphrase, you cannot *hide its USB controller from dom0*. If you've already hidden that USB controller from dom0, you must revert the procedure by removing the `rd.qubes.hide_all_usb` option and employ an alternative strategy for protecting your system by physically disconnecting other devices during startup.

**Qubes 4.1 only:** You should also add the `usbcore.authorized_default=0` option, which prevents the initialization of non-input devices. (Qubes ships with a USBGuard configuration that allows only input devices when `usbcore.authorized_default=0` is set.)



### Qubes 4.1: How to enable a USB keyboard on a separate USB controller

When using a USB keyboard on a system with multiple USB controllers, we recommend that you designate one of them exclusively for the keyboard (and possibly the mouse) and keep other devices connected to the other controller(s). This is often an option on desktop systems, where additional USB controllers can be plugged in as PCIe cards. In this case, the designated controller for input devices should remain in dom0 but be limited to input devices only. To set it up:

1. *Find the controller used for input devices.*
2. Open the file `/etc/default/grub` in dom0.
3. Find the line that begins with `GRUB_CMDLINE_LINUX`.
4. Add `usbcore.authorized_default=0` and `rd.qubes.dom0_usb=<BDF>` to that line, where `<BDF>` is the USB controller identifier.
5. Save and close the file.
6. Run the command `grub2-mkconfig -o /boot/grub2/grub.cfg` (legacy boot) or `grub2-mkconfig -o /boot/efi/EFI/qubes/grub.cfg` (EFI) in dom0.
7. Reboot.
8. Proceed with *creating a USB qube* normally. The selected USB controller will remain in dom0.

These options can be added during installation. (When the installer prompts for a reboot, you can switch to `tty2` and perform the steps from there, after using the `chroot /mnt/sysimage` command.) In that case, the initial setup will create a USB qube automatically, even when a USB keyboard is in use (as long as it is connected to the designated controller).

### USB mice

Handling a USB mouse isn't as critical as handling a keyboard, since you can log in and proceed through confirmation prompts using the keyboard alone.

If you want to attach the USB mouse automatically anyway, you have to edit the `qubes.InputMouse` policy file in dom0, located at:

```
/etc/qubes-rpc/policy/qubes.InputMouse
```

The first line should read similar to:

```
sys-usb dom0 ask,default_target=dom0
```

There will now be a confirmation prompt each time a USB mouse is attached.

If the file is empty or does not exist, something might have gone wrong during setup. Try to rerun `qubesctl state.sls qvm.sys-usb` in dom0.

In case you are absolutely sure you do not want to confirm mouse access from `sys-usb` to `dom0`, you may add the following line to the top of the file:

```
sys-usb dom0 allow
```

(Change `sys-usb` to your desired USB qube.)

## How to create a USB qube

If *automatically creating a USB qube for use with a USB keyboard* does not apply to your situation, then you may be interested in more general methods for creating USB qubes.

You can create a USB qube using the management stack by executing the following command as root in dom0:

```
sudo qubesctl state.sls qvm.sys-usb
```

## Manual creation

You can create a USB qube manually as follows:

1. Read the [PCI devices](#) page to learn how to list and identify your USB controllers. Carefully check whether you have a USB controller that would be appropriate to assign to a USB qube. Note that it should be free of input devices, programmable devices, and any other devices that must be directly available to dom0. If you find a free controller, note its name and proceed to the next step.
2. Create a new qube. Give it an appropriate name and color label (recommended: sys-usb, red).
3. In the qube's settings, go to the "Devices" tab. Find the USB controller that you identified in step 1 in the "Available" list. Move it to the "Selected" list by highlighting it and clicking the single arrow > button. (**Warning:** By assigning a USB controller to a USB qube, it will no longer be available to dom0. This can make your system unusable if, for example, you have only one USB controller, and you are running Qubes off of a USB drive.)
4. Click OK. Restart the qube.
5. Recommended: Check the box on the "Basic" tab that says "Start VM automatically on boot." (This will help to mitigate attacks in which someone forces your system to reboot, then plugs in a malicious USB device.)

If the USB qube will not start, please have a look at [this FAQ entry](#).

## How to hide USB controllers from dom0

USB controllers are automatically hidden from dom0 if you opt to create a USB qube during installation. This also occurs automatically if you choose to *create a USB qube* using the `qubesctl` method. However, if you create a USB qube manually and do not hide USB controllers from dom0, there will be a brief period of time during the boot process when dom0 will be exposed to your USB controllers (and any attached devices). This is a potential security risk, since even brief exposure to a malicious USB device could result in dom0 being compromised. There are two approaches to this problem:

1. Physically disconnect all USB devices whenever you reboot the host.
2. Hide (i.e., blacklist) all USB controllers from dom0.

**Warning:** If you use a USB keyboard, hiding your USB controllers from dom0 could lock you out of your system. See [USB keyboards](#) for more information.

**Warning:** Using a USB AEM device requires dom0 to have access to the USB controller to which your USB AEM device is attached. If dom0 cannot read your USB AEM device, AEM will hang.

The following procedure will hide all USB controllers from dom0.

## GRUB2 (legacy boot or EFI)

1. Open the file `/etc/default/grub` in dom0.
2. Find the line that begins with `GRUB_CMDLINE_LINUX`.
3. Add `rd.qubes.hide_all_usb` to that line.
4. Save and close the file.
5. Run the command `grub2-mkconfig -o /boot/grub2/grub.cfg` (legacy boot) or `grub2-mkconfig -o /boot/efi/EFI/qubes/grub.cfg` (EFI) in dom0.
6. Reboot.

## How to remove a USB qube

**Warning:** This procedure will result in your USB controller(s) being attached directly to dom0.

## GRUB2

1. Shut down the USB qube.
2. In Qubes Manager, right-click on the USB qube and select “Remove VM.”
3. Open the file `/etc/default/grub` in dom0.
4. Find the line(s) that begins with `GRUB_CMDLINE_LINUX`.
5. If `rd.qubes.hide_all_usb` appears anywhere in those lines, remove it.
6. Save and close the file.
7. Run the command `grub2-mkconfig -o /boot/grub2/grub.cfg` in dom0.
8. Reboot.

## Qubes 4.0: EFI

1. Shut down the USB qube.
2. In Qubes Manager, right-click on the USB qube and select “Remove VM.”
3. Open the file `/boot/efi/EFI/qubes/xen.cfg` in dom0.
4. Find the line(s) that begins with `kernel=`.
5. If `rd.qubes.hide_all_usb` appears anywhere in those lines, remove it.
6. Save and close the file.
7. Reboot.

### 1.12.62 Managing qube kernels

By default, VMs kernels are provided by dom0. (See [here](#) for information about upgrading kernels in dom0.) This means that:

1. You can select the kernel version (using GUI VM Settings tool or `qvm-prefs` commandline tool);
2. You can modify kernel options (using `qvm-prefs` commandline tool);
3. You can **not** modify any of the above from inside a VM;
4. Installing additional kernel modules is cumbersome.

*Note* In the examples below, although the specific version numbers might be old, the commands have been verified on R3.2 and R4.0 with debian-9 and fedora-26 templates.

To select which kernel a given VM will use, you can either use Qubes Manager (VM settings, advanced tab), or the `qvm-prefs` tool:

```
[user@dom0 ~]$ qvm-prefs -s my-appvm kernel
Missing kernel version argument!
Possible values:
1) default
2) none (kernels subdir in VM)
3) <kernel version>, one of:
  - 3.18.16-3
  - 3.18.17-4
  - 3.19.fc20
  - 3.18.10-2
[user@dom0 ~]$ qvm-prefs -s my-appvm kernel 3.18.17-4
[user@dom0 ~]$ qvm-prefs -s my-appvm kernel default
```

To check/change the default kernel you can either go to “Global settings” in Qubes Manager, or use the `qubes-prefs` tool:

```
[user@dom0 ~]$ qubes-prefs
clockvm      : sys-net
default-fw-netvm : sys-net
default-kernel : 3.18.17-4
default-netvm  : sys-firewall
default-template : fedora-21
updatevm     : sys-firewall
[user@dom0 ~]$ qubes-prefs -s default-kernel 3.19.fc20
```

To view kernel options, you can use the GUI VM Settings tool; to view and change them, use `qvm-prefs` commandline tool:

```
[user@dom0 ~]$ qvm-prefs -g work kernelopts
nopat
[user@dom0 ~]$ qvm-prefs -s work kernelopts "nopat apparmor=1 security=apparmor"
```

## Installing different kernel using Qubes kernel package

VM kernels are packages by Qubes team in `kernel-qubes-vm` packages. Generally, the system will keep the three newest available versions. You can list them with the `rpm` command:

```
[user@dom0 ~]$ rpm -qa 'kernel-qubes-vm*'
kernel-qubes-vm-3.18.10-2.pvops.qubes.x86_64
kernel-qubes-vm-3.18.16-3.pvops.qubes.x86_64
kernel-qubes-vm-3.18.17-4.pvops.qubes.x86_64
```

If you want a more recent version, you can check the `qubes-dom0-unstable` repository. There is also the `kernel-latest-qubes-vm` package which should provide a more recent (non-LTS) kernel, but has received much less testing. As the names suggest, keep in mind that those packages may be less stable than the default ones.

To check available versions in the `qubes-dom0-unstable` repository:

```
[user@dom0 ~]$ sudo qubes-dom0-update --enablerepo=qubes-dom0-unstable --action=list
↪ kernel-qubes-vm
Using sys-firewall as UpdateVM to download updates for Dom0; this may take some time...
Running command on VM: 'sys-firewall'...
Loaded plugins: langpacks, post-transaction-actions, yum-qubes-hooks
Installed Packages
kernel-qubes-vm.x86_64      1000:3.18.10-2.pvops.qubes      installed
kernel-qubes-vm.x86_64      1000:3.18.16-3.pvops.qubes      installed
kernel-qubes-vm.x86_64      1000:3.18.17-4.pvops.qubes      installed
Available Packages
kernel-qubes-vm.x86_64      1000:4.1.12-6.pvops.qubes       qubes-dom0-unstable
No packages downloaded
Installed Packages
kernel-qubes-vm.x86_64 1000:3.18.10-2.pvops.qubes @anaconda/R3.0
kernel-qubes-vm.x86_64 1000:3.18.16-3.pvops.qubes @/kernel-qubes-vm-3.18.16-3.pvops.
↪ qubes.x86_64
kernel-qubes-vm.x86_64 1000:3.18.17-4.pvops.qubes @qubes-dom0-cached
```

Installing a new version from `qubes-dom0-unstable` repository:

```
[user@dom0 ~]$ sudo qubes-dom0-update --enablerepo=qubes-dom0-unstable kernel-qubes-vm
Using sys-firewall as UpdateVM to download updates for Dom0; this may take some time...
Running command on VM: 'sys-firewall'...
Loaded plugins: langpacks, post-transaction-actions, yum-qubes-hooks
Resolving Dependencies
(...)
```

Package	Arch	Version	Repository	
↪ Size				
Installing:				
kernel-qubes-vm	x86_64	1000:4.1.12-6.pvops.qubes	qubes-dom0-cached	40 ↪
↪ M				
Removing:				
kernel-qubes-vm	x86_64	1000:3.18.10-2.pvops.qubes	@anaconda/R3.0	134 ↪
↪ M				

(continues on next page)

(continued from previous page)

## Transaction Summary

Install 1 Package

Remove 1 Package

Total download size: 40 M

Is this ok [y/d/N]: y

Downloading packages:

Running transaction check

Running transaction test

Transaction test succeeded

Running transaction (shutdown inhibited)

Installing : 1000:kernel-qubes-vm-4.1.12-6.pvops.qubes.x86\_64 1/

↪ 2

mke2fs 1.42.12 (29-Aug-2014)

This kernel version is used by at least one VM, cannot remove

error: %preun(kernel-qubes-vm-1000:3.18.10-2.pvops.qubes.x86\_64) scriptlet failed, exit\_

↪ status 1

Error in PREUN scriptlet in rpm package 1000:kernel-qubes-vm-3.18.10-2.pvops.qubes.x86\_64

Verifying : 1000:kernel-qubes-vm-4.1.12-6.pvops.qubes.x86\_64 1/

↪ 2

Verifying : 1000:kernel-qubes-vm-3.18.10-2.pvops.qubes.x86\_64 2/

↪ 2

Installed:

kernel-qubes-vm.x86\_64 1000:4.1.12-6.pvops.qubes

Failed:

kernel-qubes-vm.x86\_64 1000:3.18.10-2.pvops.qubes

Complete!

[user@dom0 ~]\$

In the above example, it tries to remove the 3.18.10-2.pvops.qubes kernel (to keep only three installed), but since some VM uses it, it fails. Installation of the new package is unaffected by this event.

The newly installed package is set as the default VM kernel.

### Installing different VM kernel based on dom0 kernel

It is possible to package a kernel installed in dom0 as a VM kernel. This makes it possible to use a VM kernel which is not packaged by Qubes team. This includes: \* using a Fedora kernel package \* using a manually compiled kernel

To prepare such a VM kernel, you need to install the qubes-kernel-vm-support package in dom0 and also have matching kernel headers installed (kernel-devel package in the case of a Fedora kernel package). You can install requirements using qubes-dom0-update:

```
[user@dom0 ~]$ sudo qubes-dom0-update qubes-kernel-vm-support kernel-devel
Using sys-firewall as UpdateVM to download updates for Dom0; this may take some time...
Running command on VM: 'sys-firewall'...
Loaded plugins: langpacks, post-transaction-actions, yum-qubes-hooks
Package 1000:kernel-devel-4.1.9-6.pvops.qubes.x86_64 already installed and latest version
```

(continues on next page)

(continued from previous page)

## Resolving Dependencies

(...)

Package	Arch	Version	Repository	Size
Installing:				
qubes-kernel-vm-support	x86_64	3.1.2-1.fc20	qubes-dom0-cached	9.2 k

## Transaction Summary

Install 1 Package

Total download size: 9.2 k

Installed size: 13 k

Is this ok [y/d/N]: y

Downloading packages:

Running transaction check

Running transaction test

Transaction test succeeded

Running transaction (shutdown inhibited)

Installing : qubes-kernel-vm-support-3.1.2-1.fc20.x86\_64 1/1

Creating symlink /var/lib/dkms/u2mfn/3.1.2/source ->  
 /usr/src/u2mfn-3.1.2

DKMS: add completed.

Verifying : qubes-kernel-vm-support-3.1.2-1.fc20.x86\_64 1/1

Installed:

qubes-kernel-vm-support.x86\_64 0:3.1.2-1.fc20

Complete!

Then you can call the `qubes-prepare-vm-kernel` tool to actually package the kernel. The first parameter is kernel version (exactly as seen by the kernel), the second one (optional) is short name. This is visible in Qubes Manager and the `qvm-prefs` tool.

```
[user@dom0 ~]$ sudo qubes-prepare-vm-kernel 4.1.9-6.pvops.qubes.x86_64 4.1.qubes
--> Building files for 4.1.9-6.pvops.qubes.x86_64 in /var/lib/qubes/vm-kernels/4.1.qubes
---> Recompiling kernel module (u2mfn)
---> Generating modules.img
mke2fs 1.42.12 (29-Aug-2014)
---> Generating initramfs
--> Done.
```

## Kernel files structure

Kernel for a VM is stored in `/var/lib/qubes/vm-kernels/KERNEL_VERSION` directory (`KERNEL_VERSION` replaced with actual version). Qubes 4.x supports the following files there:

- `mlinuz` - kernel binary (may not be a Linux kernel)
- `initramfs` - initramfs for the kernel to load
- `modules.img` - ext4 filesystem image containing Linux kernel modules (to be mounted at `/lib/modules`); additionally it should contain a copy of `mlinuz` and `initramfs` in its root directory (for loading by qemu inside stubdomain)
- `default-kernelopts-common.txt` - default kernel options, in addition to those specified with `kernelopts` qube property (can be disabled with `no-default-kernelopts` feature)

All the files besides `mlinuz` are optional in Qubes R4.1 or newer. In Qubes R4.0, `mlinuz` and `initramfs` are both required to be present.

## Using kernel installed in the VM

Both `debian-9` and `fedora-26` templates already have `grub` and related tools preinstalled so if you want to use one of the distribution kernels, all you need to do is clone either template to a new one, then:

```
qvm-prefs <clonetemplatename> virt_mode hvm
qvm-prefs <clonetemplatename> kernel ''
```

If you'd like to use a different kernel than default, continue reading.

## Installing kernel in Fedora VM

Install whatever kernel you want. You need to also ensure you have the `kernel-devel` package for the same kernel version installed.

If you are using a distribution kernel package (`kernel` package), the `initramfs` and kernel modules may be handled automatically. If you are using a manually built kernel, you need to handle this on your own. Take a look at the `dkms` documentation, especially the `dkms autoinstall` command may be useful. If you did not see the `kernel` install rebuild your `initramfs`, or are using a manually built kernel, you will need to rebuild it yourself. Replace the version numbers in the example below with the ones appropriate to the kernel you are installing:

```
sudo dracut -f /boot/initramfs-4.15.14-200.fc26.x86_64.img 4.15.14-200.fc26.x86_64
```

Once the kernel is installed, you need to setup `grub2` by running:

```
sudo grub2-install /dev/xvda
```

Finally, you need to create a GRUB configuration. You may want to adjust some settings in `/etc/default/grub`; for example, lower `GRUB_TIMEOUT` to speed up VM startup. Then, you need to generate the actual configuration. In Fedora it can be done using the `grub2-mkconfig` tool:

```
sudo grub2-mkconfig -o /boot/grub2/grub.cfg
```

You can safely ignore this error message:

```
grub2-probe: error: cannot find a GRUB drive for /dev/mapper/dmroot. Check your device.
↪map
```



Then shutdown the VM.

**Note:** You may also use PV mode instead of HVM but this is not recommended for security purposes. If you require PV mode, install `grub2-xen` in `dom0` and change the template's kernel to `pvgrub2`. Booting to a kernel inside the template is not supported under PVH.

## Installing kernel in Debian VM

### Distribution kernel

Apply the following instruction in a Debian template or in a Debian standalone.

Using a distribution kernel package the `initramfs` and kernel modules should be handled automatically.

Install distribution kernel image, kernel headers and the `grub`.

```
sudo apt install linux-image-amd64 linux-headers-amd64 grub2 qubes-kernel-vm-support
```

If you are doing that on a qube based on “Debian Minimal” template, a `grub` gui will popup during the installation, asking you where you want to install the `grub` loader. You must select `/dev/xvda` (check the box using the space bar, and validate your choice with “Enter”). If this popup does not appear during the installation, you must manually setup `grub2` by running:

```
sudo grub-install /dev/xvda
```

You can safely ignore this error message: `grub2-probe: error: cannot find a GRUB drive for /dev/mapper/dmroot`. Check your `device.map`

You may want to adjust some settings in `/etc/default/grub` (or better `/etc/default/grub.d`). For example, lower `GRUB_TIMEOUT` to speed up VM startup. You need to re-run `sudo update-grub` after making `grub` configuration changes.

Then shutdown the VM.

Go to `dom0` -> Qubes VM Manger -> right click on the VM -> Qube settings -> Advanced

Depends on Virtualization mode setting:

- Virtualization mode PV: Possible, however use of Virtualization mode PV mode is discouraged for security purposes.
  - If you require Virtualization mode PV mode, install `grub2-xen` in `dom0`. This can be done by running command `sudo qubes-dom0-update grub2-xen` in `dom0`.
- Virtualization mode PVH: Possible.
- Virtualization mode HVM: Possible.

The Kernel setting of the Virtualization mode setting:

- If Virtualization is set to PVH -> Kernel -> choose `pvgrub2-pvh` -> OK
- If Virtualization is set to PV -> Kernel -> choose `pvgrub2` -> OK
- If Virtualization is set to HVM -> Kernel -> choose `none` -> OK

Start the VM.

The process of using Qubes VM kernel with distribution kernel is complete.

### Custom kernel

Any kernel can be installed. Just make sure to install kernel headers as well.

If you are building the kernel manually, do this using `dkms` and `initramfs-tools`.

Run DKMS. Replace this with actual kernel version.

```
sudo dkms autoinstall -k <kernel-version>
```

For example.

```
sudo dkms autoinstall -k 4.19.0-6-amd64
```

Update initramfs.

```
sudo update-initramfs -u
```

The output should look like this:

```
$ sudo dkms autoinstall -k 3.16.0-4-amd64

u2mfn:
Running module version sanity check.
- Original module
  - No original module exists within this kernel
- Installation
  - Installing to /lib/modules/3.16.0-4-amd64/updates/dkms/

depmod....

DKMS: install completed.
$ sudo update-initramfs -u
update-initramfs: Generating /boot/initrd.img-3.16.0-4-amd64
```

### Troubleshooting

In case of problems, visit the [VM Troubleshooting guide](#) to learn how to access the VM console, view logs and fix a VM kernel installation.

#### 1.12.63 Salt (management software)

Since the Qubes R3.1 release we have included the Salt (also called SaltStack) management engine in dom0 as default (with some states already configured). Salt allows administrators to easily configure their systems. In this guide we will show how it is set up and how you can modify it for your own purpose.

In the current form the **API is provisional** and subject to change between *minor* releases.

## Understanding Salt

This document is not meant to be comprehensive Salt documentation; however, before writing anything it is required you have at least *some* understanding of basic Salt-related vocabulary. For more exhaustive documentation, visit [official site](#), though we must warn you that it is not easy to read if you just start working with Salt and know nothing.

## The Salt Architecture

Salt is a client-server model, where the server (called *master*) manages its clients (called *minions*). In typical situations, it is intended that the administrator interacts only with the master and keeps the configurations there. In Qubes, we don't have a master. Instead we have one minion which resides in `dom0` and manages domains from there. This setup is also supported by Salt.

Salt is a management engine (similar to Ansible, Puppet, and Chef), that enforces a particular state of a minion system. A *state* is an end effect *declaratively* expressed by the administrator. This is the most important concept in the entire engine. All configurations (i.e., the states) are written in YAML.

A *pillar* is a data back-end declared by the administrator. When states become repetitive, instead of pure YAML they can be written using a template engine (preferably Jinja2); which can use data structures specified in pillars.

A *formula* is a ready to use, packaged solution that combines a state and a pillar (possibly with some file templates and other auxiliary files). There are many formulas made by helpful people all over the Internet.

A *grain* is some data that is also available in templates, but its value is not directly specified by administrator. For example, the distribution (e.g., "Debian" or "Gentoo") is a value of the grain "os". It also contains other information about the kernel, hardware, etc.

A *module* is a Python extension to Salt that is responsible for actually enforcing the state in a particular area. It exposes some *imperative* functions for the administrator. For example, there is a `system` module that has a `system.halt` function that, when issued, will immediately halt a domain. There is another function called `state.highstate` which will synchronize the state of the system with the administrator's configuration/desires.

## Configuration

### States

The smallest unit of configuration is a state. A state is written in YAML and looks like this:

```
stateid:
  cmd.run: #this is the execution module. in this case it will execute a command on the
    ↪ shell
    - name: echo 'hello world' #this is a parameter of the state.
```

The stateid has to be unique throughout all states running for a minion and can be used to order the execution of the references state. `cmd.run` is an execution module. It executes a command on behalf of the administrator. `name: echo 'hello world'` is a parameter for the execution module `cmd.run`. The module used defines which parameters can be passed to it.

There is a list of [officially available states](#). There are many very useful states:

- For [managing files](#): Use this to create files or directories and change them (append lines, replace text, set their content etc.)
- For [installing and uninstalling packages](#).
- For [executing shell commands](#).

With these three states you can define most of the configuration of a VM.

You can also [order the execution](#) of your states:

```
D:
  cmd.run:
    - name: echo 1
    - order: last
C:
  cmd.run:
    - name: echo 1
B:
  cmd.run:
    - name: echo 1
    - require:
      - cmd: A
    - require_in:
      - cmd:C
A:
  cmd.run:
    - name: echo 1
    - order: 1
```

The order of execution will be A, B, C, D. The official documentation has more details on the [require](#) and [order](#) arguments.

## State Files

When configuring a system you will write one or more state files (\*.sls) and put (or symlink) them into the main Salt directory /srv/salt/. Each state file contains multiple states and should describe some unit of configuration (e.g., a state file mail.sls could setup a VM for e-mail).

## Top Files

After you have several state files, you need something to assign them to a VM. This is done by \*.top files ([official documentation](#)). Their structure looks like this:

```
environment:
  target_matching_clause:
    - statefile1
    - folder2.statefile2
```

In most cases, the environment will be called base. The target\_matching\_clause will be used to select your minions (VMs). It can be either the name of a VM or a regular expression. If you are using a regular expressions, you need to give Salt a hint you are doing so:

```
environment:
  ^app-(work|(?!mail).*)$:
    - match: pcre
    - statefile
```

For each target you can write a list of state files. Each line is a path to a state file (without the .sls extension) relative to the main directory. Each / is exchanged with a ., so you can't reference files or directories with a . in their name.

## Enabling Top Files and Applying the States

Now, because we use custom extensions to manage top files (instead of just enabling them all), to enable a particular top file you should issue command:

```
$ qubesctl top.enable my-new-vm
```

To list all enabled top files:

```
$ qubesctl top.enabled
```

And to disable one:

```
$ qubesctl top.disable my-new-vm
```

To apply the states to dom0 and all VMs:

```
$ qubesctl --all state.highstate
```

(More information on the `qubesctl` command further down.)

## Template Files

You will sometimes find yourself writing repetitive states. To solve this, there is the ability to template files or states. This is most commonly done with [Jinja](#). Jinja is similar to Python and in many cases behaves in a similar fashion, but there are sometimes differences when, for example, you set some variable inside a loop: the variable outside will not get changed. Instead, to get this behavior, you would use a `do` statement. So you should take a look at the [Jinja API documentation](#). Documentation about using Jinja to directly call Salt functions and get data about your system can be found in the official [Salt documentation](#).

## Salt Configuration, QubesOS layout

All Salt configuration files are in the `/srv/` directory, as usual. The main directory is `/srv/salt/` where all state files reside. States are contained in `*.sls` files. However, the states that are part of the standard Qubes distribution are mostly templates and the configuration is done in pillars from formulas.

The formulas are in `/srv/formulas`, including stock formulas for domains in `/srv/formulas/dom0/virtual-machines-formula/qvm`, which are used by `firstboot`.

Because we use some code that is not found in older versions of Salt, there is a tool called `qubesctl` that should be run instead of `salt-call --local`. It accepts all the same arguments of the vanilla tool.

## Configuring a VM's System from Dom0

Salt in Qubes can be used to configure VMs from dom0. Simply set the VM name as the target minion name in the top file. You can also use the `qubes` pillar module to select VMs with a particular property (see below). If you do so, then you need to pass additional arguments to the `qubesctl` tool:

```
usage: qubesctl [-h] [--show-output] [--force-color] [--skip-dom0]
               [--targets TARGETS | --templates | --app | --all]
               ...
```

(continues on next page)

(continued from previous page)

positional arguments:

`command` Salt `command` to execute (e.g., `state.highstate`)

optional arguments:

`-h, --help` show this `help` message and `exit`

`--show-output` Show output of management commands

`--force-color` Force color output, allow control characters from VM, UNSAFE

`--skip-dom0` Skip dom0 configuration (VM creation etc)

`--targets TARGETS` Coma separated list of VMs to target

`--templates` Target all templates

`--app` Target all app qubes

`--all` Target all non-disposables (templates and app qubes)

To apply a state to all templates, call `qubesctl --templates state.highstate`.

The actual configuration is applied using `salt-ssh` (running over `qrexec` instead of `ssh`). Which means you don't need to install anything special in a VM you want to manage. Additionally, for each target VM, `salt-ssh` is started from a temporary VM. This way dom0 doesn't directly interact with potentially malicious target VMs; and in the case of a compromised Salt VM, because they are temporary, the compromise cannot spread from one VM to another.

Beginning with Qubes 4.0 and after [QSB #45](#), we implemented two changes:

1. Added the `management_dispvm` VM property, which specifies the disposable Template that should be used for management, such as Salt configuration. App qubes inherit this property from their parent templates. If the value is not set explicitly, the default is taken from the global `management_dispvm` property. The VM-specific property is set with the `qvm-prefs` command, while the global property is set with the `qubes-prefs` command.
2. Created the `default-mgmt-dvm` disposable template, which is hidden from the menu (to avoid accidental use), has networking disabled, and has a black label (the same as templates). This VM is set as the global `management_dispvm`. Keep in mind that this disposable template has full control over the VMs it's used to manage.

## Writing Your Own Configurations

Let's start with a quick example:

```
my new and shiny VM:
qvm.present:
- name: salt-test # can be omitted when same as ID
- template: fedora-21
- label: yellow
- mem: 2000
- vcpus: 4
- flags:
- proxy
```

It uses the Qubes-specific `qvm.present` state, which ensures that the domain is present (if not, it creates it).

- The `name` flag informs Salt that the domain should be named `salt-test` (not `my new and shiny VM`).
- The `template` flag informs Salt which template should be used for the domain.
- The `label` flag informs Salt what color the domain should be.
- The `mem` flag informs Salt how much RAM should be allocated to the domain.

- The `vcpus` flag informs Salt how many Virtual CPUs should be allocated to the domain
- The `proxy` flag informs Salt that the domain should be a ProxyVM.

As you will notice, the options are the same (or very similar) to those used in `qvm-prefs`.

This should be put in `/srv/salt/my-new-vm.sls` or another `.sls` file. A separate `*.top` file should be also written:

```
base:
  dom0:
    - my-new-vm
```

**Note** The third line should contain the name of the previous state file, without the `.sls` extension.

To enable the particular top file you should issue the command:

```
$ qubesctl top.enable my-new-vm
```

To apply the state:

```
$ qubesctl state.highstate
```

## Example of Configuring a VM's System from Dom0

Lets make sure that the `mc` package is installed in all templates. Similar to the previous example, you need to create a state file (`/srv/salt/mc-everywhere.sls`):

```
mc:
  pkg.installed: []
```

Then the appropriate top file (`/srv/salt/mc-everywhere.top`):

```
base:
  qubes:type:template:
    - match: pillar
    - mc-everywhere
```

Now you need to enable the top file:

```
$ qubesctl top.enable mc-everywhere
```

And apply the configuration:

```
$ qubesctl --all state.highstate
```

## All Qubes-specific States

### `qvm.present`

As in the example above, it creates a domain and sets its properties.

### qvm.prefs

You can set properties of an existing domain:

```
my preferences:
  qvm.prefs:
    - name: salt-test2
    - netvm: sys-firewall
```

**Note** The `name:` option will not change the name of a domain, it will only be used to match a domain to apply the configurations to it.

### qvm.service

```
services in my domain:
  qvm.service:
    - name: salt-test3
    - enable:
      - service1
      - service2
    - disable:
      - service3
      - service4
    - default:
      - service5
```

This enables, disables, or sets to default, services as in `qvm-service`.

### qvm.running

Ensures the specified domain is running:

```
domain is running:
  qvm.running:
    - name: salt-test4
```

## Virtual Machine Formulae

You can use these formulae to download, install, and configure VMs in Qubes. These formulae use pillar data to define default VM names and configuration details. The default settings can be overridden in the pillar data located in:

```
/srv/pillar/base/qvm/init.sls
```

In `dom0`, you can apply a single state with `sudo qubesctl state.sls STATE_NAME`. For example, `sudo qubesctl state.sls qvm.personal` will create a personal VM (if it does not already exist) with all its dependencies (`template`, `sys-firewall`, and `sys-net`).



## Available states

### **qvm.sys-net**

System NetVM

### **qvm.sys-usb**

System USB VM

### **qvm.sys-net-as-usbvm**

System USB VM bundled into NetVM. Do not enable together with `qvm.sys-usb`.

### **qvm.usb-keyboard**

Enable USB keyboard together with USB VM, including for early system boot (for LUKS passphrase). This state implicitly creates a USB VM (`qvm.sys-usb` state), if not already done.

### **qvm.sys-firewall**

System firewall ProxyVM

### **qvm.sys-whonix**

Whonix gateway ProxyVM

### **qvm.personal**

Personal app qube

### **qvm.work**

Work app qube

### **qvm.untrusted**

Untrusted app qube

### **qvm.vault**

Vault app qube with no NetVM enabled.

### **qvm.default-dispvm**

Default disposable template - fedora-26-dvm app qube

### **qvm.anon-whonix**

Whonix workstation app qube.

### **qvm.whonix-ws-dvm**

Whonix workstation app qube for Whonix disposables.

### **qvm.updates-via-whonix**

Setup UpdatesProxy to route all templates updates through Tor (sys-whonix here).

### **qvm.template-fedora-21**

Fedora-21 template

### **qvm.template-fedora-21-minimal**

Fedora-21 minimal template

### **qvm.template-debian-7**

Debian 7 (wheezy) template

### **qvm.template-debian-8**

Debian 8 (jessie) template

**qvm.template-whonix-gw**

Whonix Gateway template

**qvm.template-whonix-ws**

Whonix Workstation template

**update.qubes-dom0**

Updates dom0. Example (executed in dom0):

```
$ sudo qubesctl --show-output state.sls update.qubes-dom0
```

**update.qubes-vm**

Updates domUs. Example to update all templates (executed in dom0):

```
$ sudo qubesctl --show-output --skip-dom0 --templates state.sls update.qubes-vm
```

Useful options:

- `--max-concurrency` — Limits how many templates are updated at the same time. Adjust to your available RAM. The default is 4, and the GUI updater sets it to 1.
- `--targets=vm1,vm2,...` — Limit to specific VMs, instead of all of them. (Use instead of `--templates` or `--standalones`.)
- `--show-output` — Show an update summary instead of just OK/FAIL.

For other options, see `qubesctl --help`.

**The qubes Pillar Module**

Additional pillar data is available to ease targeting configurations (for example all templates).

**Note:** This list is subject to change in future releases.

**qubes:type**

VM type. Possible values:

- `admin` - Administration domain (dom0)
- `template` - template
- `standalone` - Standalone VM
- `app` - Template based app qube

### qubes:template

Template name on which a given VM is based (if any).

### qubes:netvm

VM which provides network to the given VM

## Debugging

The output for each VM is logged in `/var/log/qubes/mgmt-VM_NAME.log`.

If the log does not contain useful information: 1. Run `sudo qubesctl --skip-dom0 --target=VM_NAME state.highstate` 2. When your VM is being started (yellow) press Ctrl-z on qubesctl. 3. Open terminal in disp-mgmt-VM\_NAME. 4. Look at `/etc/qubes-rpc/qubes.SaltLinuxVM` - this is what is executed in the management VM. 5. Get the last two lines:

```
...  
$ export PATH="/usr/lib/qubes-vm-connector/ssh-wrapper:$PATH"  
$ salt-ssh "$target_vm" $salt_command  
...
```

Adjust `$target_vm` (VM\_NAME) and `$salt_command` (state.highstate). 6. Execute them, fix problems, repeat.

## Known Pitfalls

### Using fedora-24-minimal

The fedora-24-minimal package is missing the sudo package. You can install it via:

```
$ qvm-run -p -u root fedora-24-minimal-template 'dnf install -y sudo'
```

The `-p` will cause the execution to wait until the package is installed. Having the `-p` flag is important when using a state with `cmd.run`.

## Disk Quota Exceeded (When Installing Templates)

If you install multiple templates you may encounter this error. The solution is to shut down the updateVM between each install:

```
install template and shutdown updateVM:  
cmd.run:  
- name: sudo qubes-dom0-update -y fedora-24; qvm-shutdown {% raw %}{{ salt.cmd.  
↪run(qubes-prefs updateVM) }}{% endraw %}
```

## Further Reading

- [Salt documentation](#)
- [Salt states \(files, commands, packages, ordering\)](#)
- [Top files](#)
- [Jinja templates](#)
- [Qubes specific modules](#)
- [Formulas for default Qubes VMs](#)

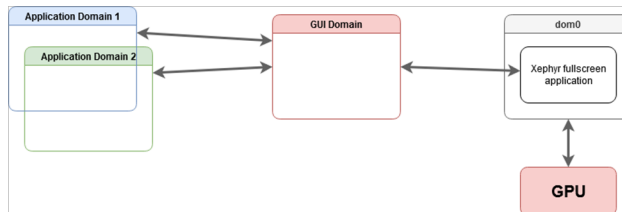
### 1.12.64 GUI domain

On this page, we describe how to set up a [GUI domain](#). In all the cases, the base underlying TemplateVM used is Fedora with XFCE flavor to match current desktop choice in dom0. That can be adapted very easily for other desktops and templates. By default, the configured GUI domain is a management qube with global admin permissions `rwX` but can be adjusted to `ro` (see [Introducing the Qubes Admin API](#)) in pillar data of the corresponding GUI domain to setup. For example, pillar data for `sys-gui` located at `/srv/pillar/base/qvm/sys-gui.sls`. Please note that each GUI domain has no NetVM.

Note: The setup is done using SaltStack formulas with the `qubesctl` tool. When executing it, apply step can take time because it needs to download latest Fedora XFCE TemplateVM and install desktop dependencies.

#### Hybrid GUI domain (sys-gui)

Here, we describe how to setup `sys-gui` that we call *hybrid mode* or referenced as a *compromise solution* in [GUI domain](#).



In `dom0`, enable the formula for `sys-gui` with pillar data:

```
sudo qubesctl top.enable qvm.sys-gui
sudo qubesctl top.enable qvm.sys-gui pillar=True
```

then, execute it:

```
sudo qubesctl --all state.highstate
```

You can now disable the `sys-gui` formula:

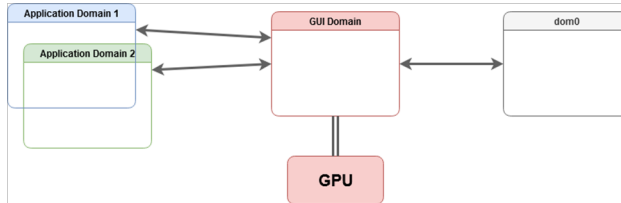
```
sudo qubesctl top.disable qvm.sys-gui
```

At this point, you need to shutdown all your running qubes as the `default_guivm` qubes global property has been set to `sys-gui`. In order to use `sys-gui` as GUI domain, you need to logout and, in the top right corner, select `lightdm` session type to **GUI domain (sys-gui)**. Once logged, you are running `sys-gui` as fullscreen window and you can perform any operation as if you would be in `dom0` desktop.

Note: In order to go back to `dom0` desktop, you need to logout and then, select `lightdm` session to *Session Xfce*.

## GPU GUI domain (sys-gui-gpu)

Here, we describe how to setup `sys-gui-gpu` which is a GUI domain with *GPU passthrough* in `GUI domain`.



In `dom0`, enable the formula for `sys-gui-gpu` with pillar data:

```
sudo qubesctl top.enable qvm.sys-gui-gpu
sudo qubesctl top.enable qvm.sys-gui-gpu pillar=True
```

then, execute it:

```
sudo qubesctl --all state.highstate
```

You can now disable the `sys-gui-gpu` formula:

```
sudo qubesctl top.disable qvm.sys-gui-gpu
```

One more step is needed: attaching the actual GPU to `sys-gui-gpu`. This can be done either manually via `qvm-pci` (remember to enable permissive option), or via:

```
sudo qubesctl state.sls qvm.sys-gui-gpu-attach-gpu
```

The latter option assumes Intel graphics card (it has hardcoded PCI address). If you don't have Intel graphics card, please use the former method with `qvm-pci` (see [How to use PCI devices](#)).

Note: Some platforms can have multiple GPU. For example on laptops, it is usual to have HDMI or DISPLAY port linked to the secondary GPU (generally called *discrete GPU*). In such case, you have to also attach the secondary GPU to `sys-gui-gpu` with permissive option.

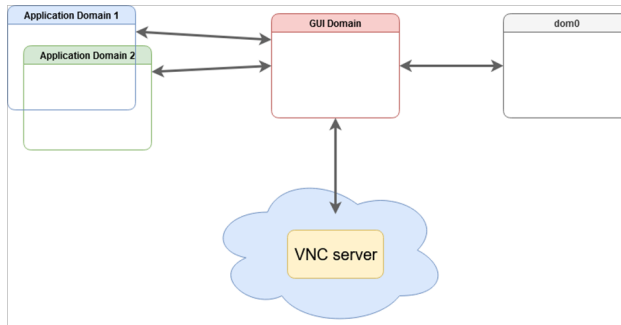
At this point, you need to reboot your Qubes OS machine in order to boot into `sys-gui-gpu`.

Note: For some platforms, it can be sufficient to shutdown all the running qubes and starting `sys-gui-gpu`. Unfortunately, it has been observed that detaching and attaching some GPU cards from `dom0` to `sys-gui-gpu` can freeze computer. We encourage reboot to prevent any data loss.

Once, `lightdm` is started, you can log as user where user refers to the first `dom0` user in qubes group and with corresponding `dom0` password. A better approach for handling password is currently discussed in [QubesOS/qubes-issues#6740](#).

## VNC GUI domain (sys-gui-vnc)

Here, we describe how to setup `sys-gui-vnc` that we call a *remote* GUI domain or referenced as *with a virtual server* in GUI domain.



In `dom0`, enable the formula for `sys-gui-vnc` with pillar data:

```
sudo qubesctl top.enable qvm.sys-gui-vnc
sudo qubesctl top.enable qvm.sys-gui-vnc pillar=True
```

then, execute it:

```
sudo qubesctl --all state.highstate
```

You can now disable the `sys-gui-vnc` formula:

```
sudo qubesctl top.disable qvm.sys-gui-vnc
```

At this point, you need to shutdown all your running qubes as the `default_guivm` qubes global property has been set to `sys-gui-vnc`. Then, you can start `sys-gui-vnc`:

```
qvm-start sys-gui-vnc
```

A VNC server session is running on `localhost:5900` in `sys-gui-vnc`. In order to reach the VNC server, we encourage to not connect `sys-gui-vnc` to a NetVM but rather to use another qube for remote access, say `sys-remote`. First, you need to bind port 5900 of `sys-gui-vnc` into a `sys-remote` local port (you may want to use another port than 5900 to reach `sys-remote` from the outside). For that, use `qubes.ConnectTCP` RPC service (see [Firewall](#)). Then, you can use any VNC client to connect to you `sys-remote` on the chosen local port (5900 if you kept the default one). For the first connection, you will reach `lightdm` for which you can log as `user` where `user` refers to the first `dom0` user in `qubes` group and with corresponding `dom0` password.

Note: `lightdm` session remains logged even if you disconnect your VNC client. Ensure to lock or log out before disconnecting your VNC client session.

**WARNING:** This setup raises multiple security issues: 1) Anyone who can reach the VNC server, can take over the control of the Qubes OS machine, 2) A second client can connect even if a connection is already active and potentially get disconnected, 3) You can get disconnected by some unrelated network issues. Generally, if this VNC server is exposed to open network, it must be protected with some other (cryptographic) layer like VPN. The setup as is, is useful only for purely testing machine.

### Troubleshooting

#### Application menu lacks qubes entries in a fresh GUI domain

See [QubesOS/qubes-issues#5804](#)

#### Delete GUI domain

The following commands have to be run in `dom0`.

Note: For the case of `sys-gui-gpu`, you need to prevent Qubes OS autostart of any qube to reach `dom0`. For that, you need to boot Qubes OS with `qubes.skip_autostart` GRUB parameter.

Set `default_guivm` as `dom0`:

```
qubes-prefs default_guivm dom0
```

and for every selected qubes not using default value for GUI domain property, for example with a qube `personal`:

```
qvm-prefs personal guivm dom0
```

You are now able to delete the GUI domain, for example `sys-gui-gpu`:

```
qvm-remove -f sys-gui-gpu
```

### General issues

For any general GUI domain issues, please take a look at existing issues [QubesOS/qubes-issues](#) under **C: gui-domain** label.

## 1.12.65 Disposable customization

### Introduction

A *disposable* can be based on any *app qube*. You can also choose to use different *disposable templates* for different disposables. To prepare an app qube to be a disposable template, you need to set the `template_for_dispvms` property:

```
[user@dom0 ~]$ qvm-prefs <DISPOSABLE_TEMPLATE> template_for_dispvms True
```

Additionally, if you want to have menu entries for starting applications in disposables based on this app qube (instead of in the app qube itself), you can achieve that with the `appmenus-dispvm` feature:

```
[user@dom0 ~]$ qvm-features <DISPOSABLE_TEMPLATE> appmenus-dispvm 1
```

**Note:** Application shortcuts that existed before setting this feature will not be updated automatically. Please go to the “Applications” tab in the qube’s “Settings” dialog and unselect all existing shortcuts by clicking “<<”, then click “OK” and close the dialog. Give it a few seconds time and then reopen and re-select all the shortcuts you want to see in the menu. See [this page](#) for background information.



## Security

If a disposable template becomes compromised, then any disposable based on that disposable template could be compromised. Therefore, you should not make any risky customizations (e.g., installing untrusted browser plugins) in important disposable templates. In particular, the *default* disposable template is important because it is used by the “Open in disposable” feature. This means that it will have access to everything that you open with this feature. For this reason, it is strongly recommended that you base the default disposable template on a trusted template and refrain from making any risky customizations to it.

## Creating a new disposable template

In Qubes 4.0, you’re no longer restricted to a single disposable template. Instead, you can create as many as you want. Whenever you start a new disposable, you can choose to base it on whichever disposable template you like. To create a new disposable template:

```
[user@dom0 ~]$ qvm-create --template <TEMPLATE> --label red <DISPOSABLE_TEMPLATE>
[user@dom0 ~]$ qvm-prefs <DISPOSABLE_TEMPLATE> template_for_dispvms True
[user@dom0 ~]$ qvm-features <DISPOSABLE_TEMPLATE> appmenus-dispvm 1
```

Optionally, set it as the default disposable template:

```
[user@dom0 ~]$ qubes-prefs default_dispvm <DISPOSABLE_TEMPLATE>
```

The above default is used whenever a qube request starting a new disposable and do not specify which one (for example `qvm-open-in-dvm` tool). This can be also set in qube settings and will affect service calls from that qube. See [qrexec documentation](#) for details.

If you wish to use a [minimal template](#) as a disposable template, please see the [minimal template](#) page.

## Customization of disposable

**Note:** *If you are trying to customize Tor Browser in a Whonix disposable, please consult the [Whonix documentation](#).*

It is possible to change the settings for each new disposable. This can be done by customizing the disposable template on which it is based:

1. Start a terminal in the <DISPOSABLE\_TEMPLATE> qube (or another disposable template) by running the following command in a dom0 terminal. (If you enable `appmenus-dispvm` feature (as explained at the top), applications menu for this VM (<DISPOSABLE\_TEMPLATE>) will be “Disposable: ” (instead of “Domain: ”) and entries there will start new disposable based on that VM (<DISPOSABLE\_TEMPLATE>). Not in that VM (<DISPOSABLE\_TEMPLATE>) itself).

```
[user@dom0 ~]$ qvm-run -a <DISPOSABLE_TEMPLATE> gnome-terminal
```

2. Change the qube’s settings and/or applications, as desired. Some examples of changes you may want to make include:
  - Changing Firefox’s default startup settings and homepage.
  - Changing default editor, image viewer. In Debian-based templates this can be done with the `mimeopen` command.
  - Changing the disposable’s default NetVM. For example, you may wish to set the NetVM to “none.” Then, whenever you start a new disposable, you can choose your desired ProxyVM manually (by changing the newly-started disposables settings). This is useful if you sometimes wish to use a disposable with a

Whonix Gateway, for example. It is also useful if you sometimes wish to open untrusted files in a network-disconnected disposable.

3. Shutdown the qube (either by `poweroff` from qube's terminal, or `qvm-shutdown` from dom0 terminal).

## Using named disposables for service qubes

You can use a *named disposable* for service qubes (such as those with the `sys-*` naming scheme) as long as they are stateless. For example, a `sys-net` using DHCP or `sys-usb` will work. In most cases `sys-firewall` will also work, even if you have configured app qube firewall rules. The only exception is if you require something like VM to VM communication and have manually edited iptables or other items directly inside the firewall app qube.

To create one that has no PCI devices attached, such as for `sys-firewall`:

```
qvm-create -C DispVM -l green <SERVICE_QUBE>
qvm-prefs <SERVICE_QUBE> autostart true
qvm-prefs <SERVICE_QUBE> netvm <NET_QUBE>
qvm-prefs <SERVICE_QUBE> provides_network true
qvm-features <SERVICE_QUBE> appmenus-dispvm ''
```

Next, set the old `sys-firewall` autostart to false, and update any references to the old one to instead point to the new, for example, with `qvm-prefs work netvm sys-firewall2`.

To create one with a PCI device attached such as for `sys-net` or `sys-usb`, use the additional commands as follows.

**Note:** You can use `qvm-pci` to *determine* the <BDF>. Also, you will often need to include the `-o no-strict-reset=True` *option* with USB controllers.

```
qvm-create -C DispVM -l red <SERVICE_QUBE>
qvm-prefs <SERVICE_QUBE> virt_mode hvm
qvm-service <SERVICE_QUBE> meminfo-writer off
qvm-pci attach --persistent <SERVICE_QUBE> dom0:<BDF>
qvm-prefs <SERVICE_QUBE> autostart true
qvm-prefs <SERVICE_QUBE> netvm ''
qvm-features <SERVICE_QUBE> appmenus-dispvm ''
```

Optionally, if this disposable will also provide network access to other qubes:

```
qvm-prefs <SERVICE_QUBE> provides_network true
```

Next, set the old service qube's autostart to false, and update any references to the old one, e.g.:

```
qvm-prefs sys-firewall netvm <SERVICE_QUBE>
```

Also make sure to update any *RPC policies*, if needed.

Here is an example of a complete `sys-net` replacement:

```
qvm-create -C DispVM -l red sys-net2
qvm-prefs sys-net2 virt_mode hvm
qvm-service sys-net2 meminfo-writer off
qvm-pci attach --persistent sys-net2 dom0:00_1a.0
qvm-prefs sys-net2 autostart true
qvm-prefs sys-net2 netvm ''
qvm-features sys-net2 appmenus-dispvm ''
qvm-prefs sys-net2 provides_network true
```

(continues on next page)

(continued from previous page)

```
qvm-prefs sys-net autostart false
qvm-prefs sys-firewall netvm sys-net2
qubes-prefs clockvm sys-net2
```

## Adding programs to the app menu

For added convenience, arbitrary programs can be added to the app menu of the disposable.

In order to do that, select “Qube settings” entry in selected base app qube, go to “Applications” tab and select desired applications as for any other qube.

Note that currently only applications whose main process keeps running until you close the application (i.e. do not start a background process instead) will work. One of known examples of incompatible applications is GNOME Terminal (shown on the list as “Terminal”). Choose different terminal emulator (like XTerm) instead.

## Deleting disposables

While working in a disposable, you may want to open a document in another disposable. For this reason, the property `default_dispvm` may be set to the name of your disposable in a number of qubes:

```
[user@dom0 ~]$ qvm-prefs <QUBE> | grep default_dispvm
default_dispvm      -  <DISPOSABLE_TEMPLATE>
```

This will prevent the deletion of the disposable template. In order to fix this, you need to unset the `default_dispvm` property:

```
[user@dom0 ~]$ qvm-prefs <QUBE> default_dispvm ""
```

You can then delete the disposable template:

```
[user@dom0 ~]$ qvm-remove <DISPOSABLE_TEMPLATE>
This will completely remove the selected VM(s)
<DISPOSABLE_TEMPLATE>
```

If you still encounter a problem, you may have forgotten to clean an entry. Looking at the system logs will help you:

```
[user@dom0 ~]$ journalctl | tail
```

## 1.12.66 Installing contributed packages

*This page is for users who wish to install contributed packages. If you want to contribute a package, please see [package contributions](#).*

Qubes OS contributed packages are available under the [QubesOS-contrib](#) GitHub Project. This is a place where our community can *contribute Qubes OS related packages, additions and various customizations*.

### Installing the repositories

If you want to install one of these packages, first you need to enable the repository in your system (dom0 and/or templates). This can be done by installing the `qubes-repo-contrib` package. This package includes the repository definition and keys necessary to download, verify, and install [QubesOS-contrib](#) packages.

In dom0, use `qubes-dom0-update`:

```
sudo qubes-dom0-update qubes-repo-contrib
```

In a Fedora-based template, use `dnf`:

```
sudo dnf install qubes-repo-contrib
```

In a Debian-based template, use `apt`:

```
sudo apt update && sudo apt install qubes-repo-contrib
```

The new repository definition will be in the usual location for your distro, and it will follow the naming pattern `qubes-contrib-*`, depending on your Qubes release and whether it is in dom0 or a template. For example, in a Fedora template on Qubes 4.0, the new repository definition would be:

```
/etc/yum.repos.d/qubes-contrib-vm-r4.0.repo
```

### Installing packages

After you've installed the repositories, you can install contributed packages.

**Note:** The first time you install a contrib package in dom0, you must use the `--clean` flag.

For example, to install `qvm-screenshot-tool` in dom0:

```
sudo qubes-dom0-update --clean qvm-screenshot-tool
```

Please see the package's README for specific installation and setup instructions.

## 1.12.67 How to make any file persistent (bind-dirs)

### What are bind-dirs?

With `bind-dirs` any arbitrary files or folders can be made persistent in app qubes.

### What is it useful for?

In an app qube all of the file system comes from the template except `/home`, `/usr/local`, and `/rw`. This means that changes in the rest of the filesystem are lost when the app qube is shutdown. `bind-dirs` provides a mechanism whereby files usually taken from the template can be persisted across reboots.

For example, in Whonix, [Tor's data dir `/var/lib/tor` has been made persistent in the TemplateBased ProxyVM `sys-whonix`](#). In this way `sys-whonix` can benefit from the Tor anonymity feature 'persistent Tor entry guards' but does not have to be a standalone.

## How to use bind-dirs.sh?

In this example, we want to make `/var/lib/tor` persistent. Enter all of the following commands in your app qube.

1. Make sure the directory `/rw/config/qubes-bind-dirs.d` exists.

```
sudo mkdir -p /rw/config/qubes-bind-dirs.d
```

2. Create the file `/rw/config/qubes-bind-dirs.d/50_user.conf` with root permissions, if it doesn't already exist.

```
sudo touch /rw/config/qubes-bind-dirs.d/50_user.conf
```

3. Add a line to `/rw/config/qubes-bind-dirs.d/50_user.conf` that appends a folder or file to the `binds` variable.

```
binds+=( '/var/lib/tor' )
```

4. Save.
5. If the directory you wish to make persistent doesn't exist in the template on which the app qube is based, you'll need to create the directory (with its full path) under `/rw/bind-dirs` in the app qube. For example, if `/var/lib/tor` didn't exist in the template, then you would execute the following command in your app qube:

```
sudo mkdir -p /rw/bind-dirs/var/lib/tor
```

6. (optional) If the directory you want to persist across reboots (`/var/lib/tor` in this case) needs special ownership and permissions, make sure the directory you created just under `/rw/bind-dirs/` has the same ones (using the commands `chown` and `chmod`, respectively).
7. Reboot the app qube.
8. Done.

From now on, all files in the `/var/lib/tor` directory will persist across reboots.

You can make as many files or folders persist as you want simply by making multiple entries in the `50_user.conf` file, each on a separate line. For example, if you added the file `/etc/tor/torrc` to the `binds` variable, any modifications to *that* file would also persist across reboots.

```
binds+=( '/var/lib/tor' )
binds+=( '/etc/tor/torrc' )
```

## Other Configuration Folders

- `/usr/lib/qubes-bind-dirs.d` (lowest priority, for packages)
- `/etc/qubes-bind-dirs.d` (intermediate priority, for template wide configuration)
- `/rw/config/qubes-bind-dirs.d` (highest priority, for per VM configuration)

## How does it work?

`bind-dirs.sh` is called at startup of an app qube, and configuration files in the above configuration folders are parsed to build a bash array. Files or folders identified in the array are copied to `/rw/bind-dirs` if they do not already exist there, and are then bind mounted over the original files/folders.

Creation of the files and folders in `/rw/bind-dirs` should be automatic the first time the app qube is restarted after configuration.

If you want to circumvent this process, you can create the relevant file structure under `/rw/bind-dirs` and make any changes at the same time that you perform the configuration, before reboot. Note that you must create the full folder structure under `/rw/bind-dirs` - e.g you would have to create `/rw/bind-dirs/var/lib/tor`

## Limitations

- Files that exist in the template root image cannot be deleted in the app qubes root image using `bind-dirs.sh`.
- Re-running `sudo /usr/lib/qubes/init/bind-dirs.sh` without a previous `sudo /usr/lib/qubes/init/bind-dirs.sh umount` does not work.
- Running `sudo /usr/lib/qubes/init/bind-dirs.sh umount` after boot (before shutdown) is probably not sane and nothing can be done about that.
- Many editors create a temporary file and copy it over the original file. If you have bind mounted an individual file this will break the mount. Any changes you make will not survive a reboot. If you think it likely you will want to edit a file, then either include the parent directory in `bind-dirs` rather than the file, or perform the file operation on the file in `/rw/bind-dirs`.
- Some files are altered when a qube boots - e.g. `/etc/hosts`. If you try to use `bind-dirs` on such files you may break your qube in unpredictable ways. You can add persistent rules to `/etc/hosts` using [\*rw/config/rc.local\*](#)

## How to remove binds from `bind-dirs.sh`?

`binds` is actually just a bash variable (an array) and the `bind-dirs.sh` configuration folders are sourced as bash snippets in lexical order. Therefore if you wanted to remove an existing entry from the `binds` array, you could do that by using a lexically higher configuration file. For example, if you wanted to make `/var/lib/tor` non-persistent in `sys-whonix` without manually editing `/usr/lib/qubes-bind-dirs.d/40_qubes-whonix.conf`, you could use the following in:

`/rw/config/qubes-bind-dirs.d/50_user.conf`

```
binds=( "${binds[@]}/'/var/lib/tor'" )
```

(Editing `/usr/lib/qubes-bind-dirs.d/40_qubes-whonix.conf` directly is strongly discouraged, since such changes get lost when that file is changed in the package on upgrades.)

## Discussion

app qubes: make selected files and folders located in the root image persistent- review bind-dirs.sh

### 1.12.68 GUI configuration

#### Video RAM adjustment for high-resolution displays

When a qube starts, a fixed amount of RAM is allocated to the graphics buffer called video RAM. This buffer needs to be at least as big as the whole desktop, accounting for all displays that are or will be connected to the machine. By default, it is as much as needed for the current display and an additional full HD (FHD) display (1920×1080 8 bit/channel RGBA). This logic fails when the machine has primary display in FHD resolution and, after starting some qubes, a 4K display is connected. If the buffer is too small, and internal desktop resize fails.

To increase the minimum size of the video RAM buffer:

```
qvm-features dom0 gui-videoram-min $((($WIDTH * $HEIGHT * 4 / 1024))
qvm-features dom0 gui-videoram-overhead 0
```

Where `$WIDTH`×` ``$HEIGHT` is the maximum desktop size that you anticipate needing. For example, if you expect to use a 1080p display and a 4k display side-by-side, that is  $(1920 + 3840) \times 2160 \times 4 / 1024 = 48600$ , or slightly more than 48 MiB per qube. After making these adjustments, the qubes need to be restarted.

In the case of multiple display with different orientations or if you plug/unplug displays, the following code will set correct memory size using `xrandr`.

```
qvm-features dom0 gui-videoram-min $(xrandr --verbose | grep "Screen 0" | sed -e 's/.*
↪*current //' -e 's/[,.*// ' | awk '{print $1*$3*4/1024}')
```

The amount of memory allocated per qube is the maximum of: - `gui-videoram-min` - current display + `gui-videoram-overhead`

Default overhead is about 8 MiB, which is enough for a 1080p display (see above). So, the `gui-videoram-overhead` zeroing is not strictly necessary; it only avoids allocating memory that will not be used.

You might face issues when playing video, if the video is choppy instead of smooth display this could be because the X server doesn't work. You can use the Linux terminal (Ctrl-Alt-F2) after starting the virtual machine, login. You can look at the Xorg logs file. As an option you can have the below config as well present in `/etc/X11/xorg.conf.d/90-intel.conf`, depends on HD graphics though -

```
Section "Device"
    Identifier "Intel Graphics"
    Driver "intel"
    Option "TearFree" "true"
EndSection
```

## GUI Troubleshooting

See *GUI Troubleshooting* for issues relating to the Qubes graphical user interface and how to fix them.

### 1.12.69 Resize disk image

#### Resizing Disk Images

By default Qubes uses thin volumes for the disk images. This means that space is not actually allocated for the volume until it is used. So a 2GB private volume with 100M of files will only use 100M. This explains how you can have *many* qubes with large private volumes on quite a small disk. This is called over provisioning. You should keep an eye on the disk-space widget to see how much free space you actually have.

It is easy to increase the size of disk images. There are risks attached to reducing the size of an image, and in general you should not need to do this.

#### Increasing the size of Disk Images

There are several disk images which can be easily extended, but pay attention to the overall consumed space of your sparse/thin disk images. In most cases, the GUI tool Qube Settings (available for every qube from the Start menu, and also in the Qube Manager) will allow you to easily increase maximum disk image size.

In case of standalone qubes and templates, just change the Disk Storage settings above. In case of template-based qubes, the private storage (the /home directory and user files) can be changed in the qube's own settings, but the system root image is *inherited from the template*, and so it must be changed in the template settings. If you are increasing the disk image size for Linux-based qubes installed from Qubes OS repositories in Qubes 4.0 or later, changing the settings above is all you need to do - in other cases, you may need to do more, according to instructions below. See also the OS-specific follow-up instructions below.

#### Increasing the size of Disk Images

Use either GUI tool Qube Settings (`qubes-vm-settings`) or the CLI tool `qvm-volume`. Maximum size which can be assigned through Qube Settings is 1048576 MiB - if you need more, use `qvm-volume`:

```
qvm-volume extend <vm_name>:root <size>
```

OR

```
qvm-volume extend <vm_name>:private <size>
```

Note: Size is the target size (i.e. 4096MB or 16GB, ...), not the size to add to the existing disk.

If you have run out of space for software in your Template, you need to increase *root image* of the Template (not private storage!). **Make sure changes in the Template between reboots don't exceed 10G.** It is recommended that you restart (or start and then shutdown, if it is not running) the template after resizing the root image.

If you are **not** using Linux in the qube, you will also need to:

1. Start the template.
2. Resize the filesystem using OS appropriate tools.
3. Verify available space in the template using `df -h` or OS specific tools.
4. Shutdown the template.



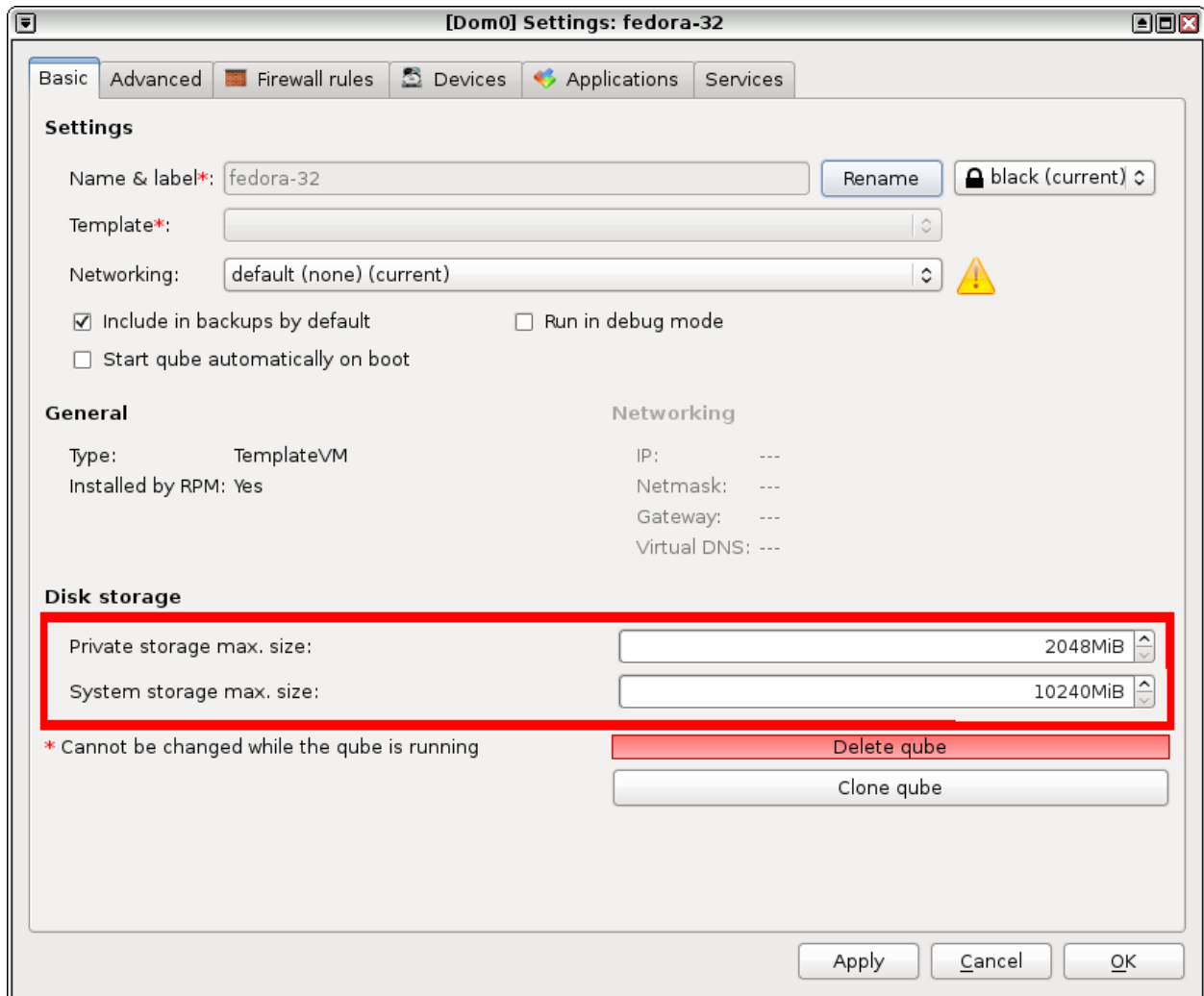


Fig. 12: vm-settings-disk-image.png

### Windows 7

1. Click Start
2. type “diskmgmt.msc” - this takes you to Disk Management
3. Right-click on your existing volume, select “Extend Volume...”
4. Click through the wizard.

No reboot required.

### FreeBSD

```
gpart recover ada0
sysctl kern.geom.debugflags=0x10
gpart resize -i index ada0
zpool online -e poolname ada0
```

### Linux

Qubes will automatically grow the filesystem for you on all app qubes with Qubes packages installed (which are all app qubes installed from templates, cloned from templates etc. - if you have not created an empty HVM and installed a Linux distribution in it, without using Qubes repositories, you are almost certainly safe). Otherwise, you will see that there is unallocated free space at the end of your primary disk. You can use standard linux tools like `fdisk` and `resize2fs` to make this space available.

### Decreasing the size of Disk Images

The number shown for “storage max size” does not mean that the storage is really using that amount. In most cases you need not worry about the size shown. If you have increased the max size, and do not need it, then you *can* reduce the allocated size, but there is a risk of data loss. Remember you really don't need to do this.

You can create a new qube, copy your files in to the new qube, and delete the old qube. (Simple and effective.)

Or you can take the risk of reducing the size of the disk. For example, to reduce the private storage of qube1 to 1GiB: Open a terminal in dom0:

```
qvm-shutdown qube1
sudo lvresize --size 1024M /dev/qubes_dom0/vm-qube1-private
```

If you have a SSD see [here](#) for information on using `fstrim`.

### 1.12.70 Qubes service

Usage documentation is in the `qvm-service` man page. There are also described predefined services.

Under the hood, an enabled service in a VM is signaled by a file in `/var/run/qubes-service`. This can be used to implement an almost enable/disable **per-VM** switch controlled by `dom0`.

Adding support for systemd services is pretty simple. In the VM, create the following file (and directory, if needed): `/etc/systemd/system/<service name>.service.d/30_qubes.conf`. It should contain the following:

```
[Unit]
ConditionPathExists=/var/run/qubes-service/<service name>
```

This will cause the service to be started only when you enable it with `qvm-service` for this VM.

### 1.12.71 How to mount a Qubes partition from another OS

When a Qubes OS install is unbootable or booting it is otherwise undesirable, this process allows for the recovery of files stored within the system.

These functions are manual and do not require any Qubes specific tools. All steps assume the default Qubes install with the following components: - LUKS encrypted disk - LVM based VM storage

Before beginning, if attempting to access one Qubes system from another, it is recommended to pass the entire encrypted Qubes disk to an isolated app qube. This can be done with the command `qvm-block attach <isolated vm> dom0:<disk>` in `dom0`.

#### Decrypting the Disk

1. Find the disk to be accessed:

1. Open a Linux terminal in either `dom0` or the app qube the disk was passed through to and enter `lsblk`, which will result in an output similar to the following. In this example, the currently booted Qubes system is installed on `sda` and the qubes system to be accessed is on `nvme0n1p2`.

```
sda                                                    8:0    0
├─sda1                                                8:1    0
│   └─200M 0 part /boot/efi
├─sda2                                                8:2    0
│   └─1G 0 part /boot
└─sda3                                                8:3    0
    └─110.6G 0 part
        └─luks-fed62fc2-2674-266d-2667-2667259cbdec    253:0    0
            └─110.6G 0 crypt
                └─qubes_dom0-pool00_tmeta            253:1    0
                    └─88M 0 lvm
                        └─qubes_dom0-pool00-tpool      253:3    0
                            └─84.4G 0 lvm
                                └─qubes_dom0-root      253:4    0
                                    └─84.4G 0 lvm /
                                        └─qubes_dom0-pool00 253:6    0
                                            └─84.4G 0 lvm
                                                └─qubes_dom0-vm--fedora--30--dvm--private--1576749131--back 253:7    0
                                                    └─2G 0 lvm
```

(continues on next page)

(continued from previous page)

	qubes_dom0-pool00_tdata	253:2	0	└
↪	84.4G 0 lvm			
	qubes_dom0-pool00-tpool	253:3	0	└
↪	84.4G 0 lvm			
	qubes_dom0-root	253:4	0	└
↪	84.4G 0 lvm /			
	qubes_dom0-pool00	253:6	0	└
↪	84.4G 0 lvm			
	qubes_dom0-vm--fedora--30--dvm--private--1576749131--back	253:7	0	└
↪	2G 0 lvm			
	qubes_dom0-swap	253:5	0	└
↪	4G 0 lvm [SWAP]			
sdb		8:16	0	└
↪	447.1G 0 disk			
	sdb1	8:17	0	└
↪	549M 0 part			
	sdb2	8:18	0	└
↪	446.6G 0 part			
sr0		11:0	1	└
↪	1024M 0 rom			
nvme0n1		259:0	0	└
↪	465.8G 0 disk			
	nvme0n1p1	259:1	0	└
↪	1G 0 part			
	nvme0n1p2	259:2	0	└
↪	464.8G 0 part			

2. Decrypt the disk using the command `cryptsetup luksOpen /dev/<disk>`.

## Accessing LVM Logical Volumes

3. If using an app qube or standard Linux, LVM should automatically discover the Qubes LVM configuration. In this case, continue to step 4.
  1. Qubes uses the default name `qubes_dom0` for its LVM VG. This will conflict with the name of the VG of the currently installed system. To read both, you will have to rename the VG. *Note:* If this is not reversed, the Qubes install being accessed will not be bootable.
  2. Find the UUID of the vg to be accessed using the command `vgdisplay`. This will be the VG named `qubes_dom0` which is not marked active.
  3. The command `vgrename <UUID> other_install` will rename the VG.
4. Run the command `vgscan` to add any new VGs to the device list.

## Mounting the disk

5. Find the disk to be accessed. The `lsblk` command above may be of use. The following rules apply by default:

Disk name	Data type	Explanation
<code>other_install/root</code>	dom0 root	The root partition of dom0.
<code>other_install/-private</code>	VM	The /rw partition of the named VM.
<code>other_install/-root</code>	template root	The root partition of the named template.
<code>other install/pool00_tmeta</code>	LVM Metadata	The metadata LV of this disk.

6. Mount the disk using the command `mount /dev/other_install/<lv name> <mountpoint>`. *Note:* Any compromised data which exists in the volume to be mounted will be accessible here. Do not mount untrusted partitions in dom0.

At this point, all files are available in the chosen mountpoint.

## Reverting Changes

Any changes which were made to the system in the above steps will need to be reverted before the disk will properly boot. However, LVM will not allow an VG to be renamed to a name already in use. These steps must occur either in an app qube or using recovery media.

1. Unmount any disks that were accessed.
2. Rename the VG back to `qubes_dom0` using the command `vgrename other_install qubes_dom0`.

## 1.12.72 KDE (desktop environment)

### Installation

Prior to R3.2, KDE was the default desktop environment in Qubes. Beginning with R3.2, however, *XFCE is the new default desktop environment*. Nonetheless, it is still possible to install KDE by issuing this command in dom0:

```
$ sudo qubes-dom0-update kde-settings-qubes
```

You can also change your default login manager (lightdm) to the new KDE default: sddm

- first you need to edit the `/etc/sddm.conf` to make sure if the custom X parameter is set according to Qubes needs:

```
[XDisplay]
ServerArguments=-nolisten tcp -background none
```

- disable the lightdm service:

```
$ sudo systemctl disable lightdm
```

- enable the sddm service:

```
$ sudo systemctl enable sddm
```

- reboot

If you encounter performance issues with KDE, try switching back to LightDM.

## Window Management

You can set each window's position and size like this:

```
Right click title bar --> More actions --> Special window settings...
```

```
Window matching tab
  Window class (application): Exact Match: <vm_name>
  Window title: Substring Match: <partial or full program name>

Size & Position tab
  [x] Position: Apply Initially: x,y
  [x] Size: Apply Initially: x,y
```

You can also use `kstart` to control virtual desktop placement like this:

```
kstart --desktop 3 --windowclass <vm_name> -q --tray -a <vm_name> '<run_program_command>'
```

(Replace “3” with whichever virtual desktop you want the window to be on.)

This can be useful for creating a simple shell script which will set up your workspace the way you like.

## Removal

If you decide to remove KDE do **not** use `dnf remove @kde-desktop-qubes`. You will almost certainly break your system.

The safest way to remove (most of) KDE is:

```
sudo dnf remove kdelibs plasma-workspace
```

## Mailing List Threads

- [Nalu's KDE customization thread](#)

### 1.12.73 i3 (window manager)

i3 is part of the stable repository (as of Qubes R3.1) and can be installed by using the *dom0 update mechanism*. To install the i3 window manager and the its Qubes specific configuration:

```
$ sudo qubes-dom0-update i3 i3-settings-qubes
```

The Qubes-specific configuration (package `i3-settings-qubes`) can be installed optionally in case you would prefer writing your own configuration (see *customization* section for scripts and configuration).

That's it. After logging out, you can select i3 in the login manager.

## Customization

**Caution:** The following external resources may not have been reviewed by the Qubes team.

- `xdg_autostart_script`
- `i3bar_script`
- `terminal_start_script`
- `i3` config with `dmenu-i3-window-jumper`
- `dmenu` script to open a terminal in a chosen VM

## Compilation and installation from source

Note that the compilation from source is done in a Fedora based domU (could be `dispvmm`). The end result is always an `.rpm` that is copied to `dom0` and then installed through the package manager.

## Getting the code

Clone the `i3-qubes` repository here:

```
$ git clone https://github.com/QubesOS/qubes-desktop-linux-i3
```

In this case, the most interesting file is probably `i3/0001-Show-qubes-domain-in-non-optional-colored-borders.patch`. It's the patch with changes that are necessary to make `i3` work nicely with Qubes OS. The code should not need much explanation, it just gets the `vmname` and `label` from Qubes OS and changes some defaults so the user can't override decisions.

If you want to make any changes to the package, this is the time and place to do it.

## Building

You'll need to install the build dependencies, which are listed in `build-deps.list`. You can verify them and then install them with:

```
$ sudo dnf install -y $(cat build-deps.list)
```

This used to be more complicated, but I finally redid this and use the same buildsystem that's used by Qubes OS for XFCE. It's just a Makefile that helps you get the sources and start off the build:

```
$ make get-sources
$ make verify-sources
$ make rpms
```

### Installing

**Warning:** Manually installing software in dom0 is inherently risky, and the method described here circumvents the usual security mechanisms of qubes-dom0-update.

You should now have your i3 rpm in `./rpm/x86_64/i3-4.8-3.fc20.x86_64.rpm`. Protip: copying this file to `~/i3.rpm` now will save you some typing in the next step.

Now in dom0, copy in the rpm:

```
$ qvm-run --pass-io <src_domain> 'cat </path/to/rpm_in_src_domain>' > i3.rpm
```

Now that the rpm is in dom0 we can proceed with installing it. i3 has some dependencies that we can easily install with:

```
$ sudo qubes-dom0-update perl-AnyEvent-I3 xorg-x11-apps \\  
    rxvt-unicode xcb-util-wm perl-JSON-XS xcb-util-cursor \\  
    dzen2 dmenu xorg-x11-fonts-misc libev
```

After that you can just install the generated rpm like any other local package:

```
$ sudo yum localinstall i3.rpm
```

Log out, select i3, then log in again.

## 1.12.74 AwesomeWM (window manager)

### Qubes-specific features

- support for the Qubes OS window colors
- rudimentary support for the Qubes application menu entries following the freedesktop standard
- support for custom filters and menu entries

### Installation

AwesomeWM can be installed with the standard dom0 installation mechanisms.

```
$ sudo qubes-dom0-update awesome
```

That's it. After logging out, you can select AwesomeWM in the login manager.

### Development

To *contribute code* you may clone the AwesomeWM repository as follows:

```
$ git clone https://github.com/QubesOS/qubes-desktop-linux-awesome
```

For build instructions please check the repository *README*.

The repository attempts to follow the upstream Fedora repository.



## 1.12.75 Command-line tools

### Dom0

- `core-admin`
- `core-admin-client`

### DomU

- `qrexec-client-vm`
- `qvm-copy`
- `qvm-open-in-dvm`
- `qvm-open-in-vm`
- `qvm-run-vm`
- `qvm-convert-pdf`
- `qvm-convert-img`

## 1.12.76 Glossary

### admin qube

A type of *qube* used for administering Qubes OS.

- Currently, the only admin qube is *dom0*.

### app qube

Any *qube* that does not have a root filesystem of its own. Every app qube is based on a *template* from which it borrows the root filesystem.

- Previously known as: `AppVM`, `TemplateBasedVM`.
- Historical note: This term originally meant “a qube intended for running user software applications” (hence the name “app”).

### disposable

A type of temporary *app qube* that self-destructs when its originating window closes. Each disposable is based on a *disposable template*.

See *How to Use Disposables*.

- Previously known as: `DisposableVM`, `DispVM`.

## disposable template

Any *app qube* on which *disposables* are based. A disposable template shares its user directories (and, indirectly, the root filesystem of the regular *template* on which it is based) with all *disposables* based on it.

- Not to be confused with the concept of a regular *template* that is itself disposable, which does not exist in Qubes OS.
- Disposable templates must be app qubes. They cannot be regular *templates*.
- Every *disposable* is based on a disposable template, which is in turn based on a regular *template*.
- Unlike *disposables*, disposable templates have the persistence properties of normal *app qubes*.
- Previously known as: DisposableVM Template, DVM Template, DVM.

## dom0

*Domain* zero. A type of *admin qube*. Also known as the **host** domain, dom0 is the initial qube started by the Xen hypervisor on boot. Dom0 runs the Xen management toolstack and has special privileges relative to other domains, such as direct access to most hardware.

- The term “dom0” is a common noun and should follow the capitalization rules of common nouns.

## domain

In Xen, a synonym for *VM*.

See “[domain](#)” on the Xen Wiki.

- This term has no official meaning in Qubes OS.

## domU

Unprivileged *domain*. Also known as **guest** domains, domUs are the counterparts to dom0. In Xen, all VMs except dom0 are domUs. By default, most domUs lack direct hardware access.

- The term “domU” is a common noun and should follow the capitalization rules of common nouns.
- Sometimes the term *VM* is used as a synonym for domU. This is technically inaccurate, as *dom0* is also a VM in Xen.

## HVM

Hardware-assisted Virtual Machine. Any fully virtualized, or hardware-assisted, *VM* utilizing the virtualization extensions of the host CPU. Although HVMs are typically slower than paravirtualized qubes due to the required emulation, HVMs allow the user to create domains based on any operating system.

See *Standalones and HVM*.

## management qube

A *qube* used for automated management of a Qubes OS installation via *Salt*.

## named disposable

A type of *disposable* given a permanent name that continues to exist even after it is shut down and can be restarted again. Like a regular *disposable*, a named disposable has no persistent state: Any changes made are lost when it is shut down.

- Only one instance of a named disposable can run at a time.
- Like a regular *disposable*, a named disposable always has the same state when it starts, namely that of the *disposable template* on which it is based.
- Technical note: Named disposables are useful for certain *service qubes*, where the combination of persistent device assignment and ephemeral qube state is desirable.

## net qube

Internally known as *netvm*. The property of a *qube* that specifies from which qube, if any, it receives network access. Despite the name, “net qube” (or *netvm*) is a *property* of a qube, not a *type* of qube. For example, it is common for the net qube of an *app qube* to be the *service qube* *sys-firewall*, which in turn uses *sys-net* as its net qube.

- The name *netvm* derives from “Networking Virtual Machine.” Before Qubes 4.0, there was a type of *service qube* called a “NetVM.” The name of the *netvm* property is a holdover from that era.

## qube

A secure compartment in Qubes OS. Currently, qubes are implemented as Xen *VMs*, but Qubes OS is independent of its underlying compartmentalization technology. VMs could be replaced with a different technology, and qubes would still be called “qubes.”

- **Important:** The term “qube” is a common noun and should follow the capitalization rules of common nouns. For example, “I have three qubes” is correct, while “I have three Qubes” is incorrect.
- Note that starting a sentence with the plural of “qube” (i.e., “Qubes...”) can be ambiguous, since it may not be clear whether the referent is a plurality of qubes or *Qubes OS*. You may wish to rephrase sentences in order to avoid this ambiguity.
- Example usage: “In Qubes OS, you do your banking in your ‘banking’ qube and your web surfing in your ‘untrusted’ qube. That way, if your ‘untrusted’ qube is compromised, your banking activities will remain secure.”
- Historical note: The term “qube” was originally invented as an alternative to “VM” intended to make it easier for less technical users to understand Qubes OS and learn how to use it.

### Qubes OS

A security-oriented operating system (OS). The main principle of Qubes OS is security by compartmentalization (or isolation), in which activities are compartmentalized (or isolated) in separate *qubes*.

- **Important:** The official name is “Qubes OS” (note the capitalization and the space between “Qubes” and “OS”). In casual conversation, this is often shortened to “Qubes.” Only in technical contexts where spaces are not permitted (e.g., in usernames) may the space be omitted, as in @QubesOS.

### Qubes Windows Tools (QWT)

A set of programs and drivers that provide integration of Windows qubes with the rest of the Qubes OS system.

See Qubes Windows Tools and Windows.

### service qube

Any *app qube* the primary purpose of which is to provide services to other qubes. `sys-net` and `sys-firewall` are examples of service qubes.

### standalone

Any *qube* that has its own root filesystem and does not share it with another qube. Distinct from both *templates* and *app qubes*.

See *Standalones and HVMs*.

- Previously known as: StandaloneVM.

### template

Any *qube* that shares its root filesystem with another qube. A qube that is borrowing a template’s root filesystem is known as an *app qube* and is said to be “based on” the template. Templates are intended for installing and updating software applications, but not for running them.

See *Templates*.

- No template is an *app qube*.
- A template cannot be based on another template.
- Regular templates cannot function as *disposable templates*. (Disposable templates must be app qubes.)
- Previously known as: TemplateVM.

### VM

An abbreviation for “virtual machine.” A software implementation of a machine (for example, a computer) that executes programs like a physical machine.

### 1.12.77 Qubes OS project security center

This page provides a central hub for topics pertaining to the security of the Qubes OS Project. For topics pertaining to software security *within* Qubes OS, see [security in Qubes](#). The following is a list of important project security pages:

- [Qubes security pack \(qubes-secpack\)](#)
- [Qubes security bulletins \(QSBs\)](#)
- [Qubes canaries](#)
- [Xen security advisory \(XSA\) tracker](#)
- [Verifying signatures](#)
- [PGP keys](#)
- [Security FAQ](#)

### Reporting security issues in Qubes OS

**Warning:** Please note: The Qubes security team email address is intended for responsible disclosure by security researchers and others who discover legitimate security vulnerabilities. It is not intended for everyone who suspects they've been hacked. Please do not attempt to contact the Qubes security team unless you can demonstrate an actual security vulnerability or unless the team will be able to take reasonable steps to verify your claims.

If you've discovered a security issue affecting Qubes OS, either directly or indirectly (e.g., the issue affects Xen in a configuration that is used in Qubes OS), then we would be more than happy to hear from you! We promise to take all reported issues seriously. If our investigation confirms that an issue affects Qubes, we will patch it within a reasonable time and release a public [Qubes security bulletin \(QSB\)](#) that describes the issue, discusses the potential impact of the vulnerability, references applicable patches or workarounds, and credits the discoverer. Please use the [Qubes security team PGP key](#) to encrypt your email to this address:

security at qubes-os dot org

This key is signed by the [Qubes Master Signing Key](#). Please see [verifying signatures](#) for information about how to authenticate these keys.

### Security updates

Qubes security updates are obtained by [updating Qubes OS](#).

### Qubes security team

The **Qubes security team (QST)** is the subset of the [core team](#) that is responsible for ensuring the security of Qubes OS and the Qubes OS Project. In particular, the QST is responsible for:

- Responding to [reported security issues](#)
- Evaluating whether [XSAs](#) affect the security of Qubes OS
- Writing, applying, and/or distributing security patches to fix vulnerabilities in Qubes OS
- Writing, signing, and publishing [Qubes security bulletins \(QSBs\)](#)
- Writing, signing, and publishing [Qubes canaries](#)

- Generating, safeguarding, and using the project's [PGP keys](#)

As a security-oriented operating system, the QST is fundamentally important to Qubes, and every Qubes user implicitly trusts the members of the QST by virtue of the actions listed above.

### Members of the security team

- [Marek Marczykowski-Górecki](#)
- [Simon Gaiser \(aka HW42\)](#)
- [Joanna Rutkowska \(emeritus, canaries only\)](#)

### 1.12.78 Qubes security pack (qubes-secpack)

The **Qubes security pack (qubes-secpack)** is a Git repository that contains:

- Qubes security bulletins (QSBs)
- Qubes canaries
- *Qubes ISO cryptographic hash values*
- Qubes fund information
- Qubes PGP keys
- Security-related information and announcements (e.g., key revocations)

While qubes-secpack itself is independent of any particular host, its current official location is:

<https://github.com/QubesOS/qubes-secpack>

### How to obtain and authenticate

The following example demonstrates one method of obtaining the qubes-secpack and verifying its authenticity. This requires Git and *OpenPGP software*.

1. Use Git to clone the qubes-secpack repo.

```
$ git clone https://github.com/QubesOS/qubes-secpack.git
Cloning into 'qubes-secpack'...
remote: Counting objects: 195, done.
remote: Total 195 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (195/195), 130.94 KiB | 207.00 KiB/s, done.
Resolving deltas: 100% (47/47), done.
Checking connectivity... done.
```

2. Import the included PGP keys. See our *PGP key policies* for important information about these keys.

```
$ gpg --import qubes-secpack/keys/*/*
gpg: directory `/home/user/.gnupg' created
gpg: new configuration file `/home/user/.gnupg/gpg.conf' created
gpg: WARNING: options in `/home/user/.gnupg/gpg.conf' are not yet active during
↳ this run
gpg: keyring `/home/user/.gnupg/secring.gpg' created
gpg: keyring `/home/user/.gnupg/pubring.gpg' created
```

(continues on next page)

(continued from previous page)

```

gpg: /home/user/.gnupg/trustdb.gpg: trustdb created
gpg: key C37BB66B: public key "Joanna Rutkowska (Qubes OS signing key)
↳<joanna@invisiblethingslab.com>" imported
gpg: key 1E30A75D: public key "Joanna Rutkowska (Qubes OS signing key)
↳<joanna@invisiblethingslab.com>" imported
gpg: key 74EADABC: public key "Joanna Rutkowska (Qubes OS signing key)
↳<joanna@invisiblethingslab.com>" imported
gpg: key 65EF29CA: public key "Joanna Rutkowska (Qubes OS Signing Key)
↳<joanna@invisiblethingslab.com>" imported
gpg: key 34898310: public key "Joanna Rutkowska (Qubes OS Signing Key)
↳<joanna@invisiblethingslab.com>" imported
gpg: key B298547C: public key "Marek Marczykowski (Qubes OS signing key)
↳<marmarek@mimuw.edu.pl>" imported
gpg: key AB5EEF90: public key "Marek Marczykowski (Qubes OS signing key)
↳<marmarek@invisiblethingslab.com>" imported
gpg: key A603BCB6: public key "Marek Marczykowski (Qubes OS signing key)
↳<marmarek@invisiblethingslab.com>" imported
gpg: key 42CFA724: public key "Marek Marczykowski-Górecki (Qubes OS signing key)
↳<marmarek@invisiblethingslab.com>" imported
gpg: key 15CE40BF: public key "Wojciech Zygmunt Porczyk (Qubes OS signing key)
↳<woju@invisiblethingslab.com>" imported
gpg: key 36879494: public key "Qubes Master Signing Key" imported
gpg: key 211093A7: public key "Qubes OS Release 1 Signing Key" imported
gpg: key 0A40E458: public key "Qubes OS Release 2 Signing Key" imported
gpg: key 03FA5082: public key "Qubes OS Release 3 Signing Key" imported
gpg: key 92C7B3DC: public key "Joanna Rutkowska (Qubes Security Pack Signing Key)
↳<joanna@invisiblethingslab.com>" imported
gpg: key 1830E06A: public key "Marek Marczykowski-Górecki (Qubes security pack)
↳<marmarek@invisiblethingslab.com>" imported
gpg: key 3F48CB21: public key "Qubes OS Security Team <security@qubes-os.org>"
↳imported
gpg: Total number processed: 17
gpg:             imported: 17 (RSA: 17)
gpg: no ultimately trusted keys found

```

3. *Authenticate and set the trust level of the Qubes Master Signing Key (QMSK).*

4. Verify signed Git tags.

```

$ cd qubes-secpack/
$ git tag -v `git describe`
object 2bb7f0b966593d8ed74e140a04d60c68b96b164e
type commit
tag joanna_sec_2bb7f0b9
tagger Joanna Rutkowska <joanna@invisiblethingslab.com> 1468335706 +0000

Tag for commit 2bb7f0b966593d8ed74e140a04d60c68b96b164e
gpg: Signature made 2016-07-12T08:01:46 PDT
gpg:             using RSA key 0x4E6829BC92C7B3DC
gpg: Good signature from "Joanna Rutkowska (Qubes Security Pack Signing Key)
↳<joanna@invisiblethingslab.com>" [full]

```

The final line of output confirms that the signature is good.

5. Verify detached PGP signatures.

```
$ cd canaries/
$ gpg --verify canary-001-2015.txt.sig.joanna canary-001-2015.txt
gpg: Signature made Mon Jan  5 20:21:40 2015 UTC using RSA key ID 92C7B3DC
gpg: Good signature from "Joanna Rutkowska (Qubes Security Pack Signing Key)
-><joanna@invisiblethingslab.com>"
$ gpg --verify canary-001-2015.txt.sig.marmarek canary-001-2015.txt
gpg: Signature made Mon Jan  5 20:13:37 2015 UTC using RSA key ID 1830E06A
gpg: Good signature from "Marek Marczykowski-Górecki (Qubes security pack)
-><marmarek@invisiblethingslab.com>"
```

The fourth and final lines of output confirm that the two signatures are good.

The same procedures can be applied to any directory or file in the qubes-secpack. Two methods of verification (signed Git tags and detached PGP signatures) are provided to ensure that the system is robust (e.g., against a potential failure in Git tag-based verification) and to give users more options to verify the files.

## PGP key policies

- **Inclusion criteria.** The qubes-secpack generally includes only those PGP keys used to sign some kind of official project asset, such as Qubes release ISOs (release signing keys), Git tags and commits (code signing, doc signing, and security team keys), and the qubes-secpack's own files and Git tags (security team keys again). This means that email keys are generally not included, even for official project email addresses. There is one exception to this rule: the official *Qubes security team* email address, which is used to report security vulnerabilities in Qubes OS to our security team.
- **Key signing (certification).** Only some keys in the qubes-secpack are signed by the QMSK. Keys that are not signed directly by the QMSK are still signed indirectly by virtue of being included in the qubes-secpack, which is itself signed (via Git tags and/or commits) by keys that are in turn signed by the QMSK.

## History and rationale

On 2013-01-05, Joanna Rutkowska announced the qubes-secpack and explained its rationale in an [email](#) to the Qubes mailing lists:

```
Hello,

A new Qubes Security Bulletin has been just released and is available here:

https://github.com/QubesOS/qubes-secpack/blob/master/QSBs/qsb-013-2015.txt

As per the previous discussions about recent problems with verifying
digital signatures on messages sent to Google Groups (thanks to
automatic footer addition by Google), we have decided to change the way
we publish Qubes Security Bulletins, as well as other security-related
info pertinent to the Qubes Project.

Starting today, we will be maintain a Git repository -- "Qubes Security
Pack" -- which will contain all the QSBs released so far, all the keys,
warrant canaries [1], and potentially some additional info or
announcements (e.g. key revocations). The whole repo can be found here:
```

(continues on next page)



(continued from previous page)

<https://github.com/QubesOS/qubes-secpack>

Note that all the keys distributed there should be signed by Qubes Master Key. The Master Key is also attached **in** the repo, but should really be obtained/verified using a different channel.

Additionally, most of the files are signed by core Qubes developers (currently by Marek and myself) via detached signatures as well as git tag signatures.

There are several advantages of using Git to distribute all this information:

- 1) Git repo is a collection of files, some of which can be detached GPG signatures **for** other files and we can ensure all these files are distributed together.
- 2) Git makes it easy **for** people to clone and redistribute these collection of files, as well as to easily host them and view on the Web.
- 3) Git provides **for** signed tags mechanisms which is another mean we utilize to ensure integrity of the distributed files.

A few words about the Warrant Canary which we've just introduced today, and which can be seen here:

<https://github.com/QubesOS/qubes-secpack/blob/master/canaries/canary-001-2015.txt>

Even though we're not providing any kind of services (such as e.g. email hosting), that could be searched or tapped by authorities, there are other possibilities that worry us [2], **in** the light of various recent law "developments", such as those that might be coercing people to hand over their private keys to authorities.

Until we fully decentralize the root of trust **for** Qubes, something that requires the move to deterministic builds [3], and so won't happen very soon, the possibility of having to disclose any of the Qubes signing keys to anybody might have pretty serious consequences for those who decided to entrust Qubes with anything serious. And we would like to somehow minimize these consequences with this canary thing.

Additionally the canary is a nice way of ensuring "freshness" of our messaging to the community.

Of course the canary doesn't solve all the problems. E.g. **if** my signing keys were somehow stolen without our knowledge, it wouldn't help. Neither it could help in case me being or becoming a miscreant. And probably it doesn't address many other potential problems, which could only be solved one day with a multi-signature scheme. But anyway, **until** that time, this is the best we can **do**, I think.

And congrats to Jann **for** the very interesting clipboard attack (even though mostly theoretical, still very cool)!

(continues on next page)

(continued from previous page)

Thanks,  
joanna.

--

The Qubes Security Team  
<https://www.qubes-os.org/doc/SecurityPage>

[1] [http://en.wikipedia.org/wiki/Warrant\\_canary](http://en.wikipedia.org/wiki/Warrant_canary)

[2] Especially myself, because I'm currently the Root Of Trust for all Qubes binaries :/

[3] Deterministic builds are required because it's the only way we can implement multiple signature scheme for distributed binaries.

## 1.12.79 Verifying signatures

The Qubes OS Project uses [digital signatures](#) to guarantee the authenticity and integrity of certain important assets. This page explains how to verify those signatures. It is extremely important for your security to understand and apply these practices.

### What digital signatures can and cannot prove

Most people — even programmers — are confused about the basic concepts underlying digital signatures. Therefore, most people should read this section, even if it looks trivial at first sight.

Digital signatures can prove both **authenticity** and **integrity** to a reasonable degree of certainty. **Authenticity** ensures that a given file was indeed created by the person who signed it (i.e., that a third party did not forge it). **Integrity** ensures that the contents of the file have not been tampered with (i.e., that a third party has not undetectably altered its contents *en route*).

Digital signatures **cannot** prove, e.g., that the signed file is not malicious. In fact, there is nothing that could stop someone from signing a malicious program (and it happens from time to time in reality).

The point is that we must decide who we will trust (e.g., Linus Torvalds, Microsoft, or the Qubes Project) and assume that if a trusted party signed a given file, then it should not be malicious or negligently buggy. The decision of whether to trust any given party is beyond the scope of digital signatures. It's more of a social and political decision.

Once we decide to trust certain parties, digital signatures are useful, because they make it possible for us to limit our trust only to those few parties we choose and not to worry about all the bad things that can happen between them and us, e.g., server compromises (qubes-os.org will surely be compromised one day, so *don't blindly trust the live version of this site*), dishonest IT staff at the hosting company, dishonest staff at the ISPs, Wi-Fi attacks, etc. We call this philosophy *distrusting the infrastructure*.

By verifying all the files we download that purport to be authored by a party we've chosen to trust, we eliminate concerns about the bad things discussed above, since we can easily detect whether any files have been tampered with (and subsequently choose to refrain from executing, installing, or opening them).

However, for digital signatures to make sense, we must ensure that the public keys we use for signature verification are the original ones. Anybody can generate a cryptographic key that purports to belong to "The Qubes OS Project," but of course only the keys that we (the real Qubes developers) generate are the genuine ones. The rest of this page

explains how to verify the authenticity of the various keys used in the project and how to use those keys to verify certain important assets.

## OpenPGP software

We use **PGP** (specifically, the **OpenPGP** standard). Before we begin, you'll need software that can manage PGP keys and verify PGP signatures. Any program that complies with the OpenPGP standard will do, but here are some examples for popular operating systems:

**Linux:** **GnuPG** ([documentation](#)). Open a terminal and use the `gpg2` command. If you don't already have GnuPG installed, install it via your distro's package manager or from the GnuPG website.

**Mac:** **GPG Suite** ([documentation](#)). Open a terminal to enter commands.

**Windows:** **Gpg4win** ([documentation](#)). Use the Windows command line (`cmd.exe`) to enter commands.

Throughout this page, we'll use GnuPG via the `gpg2` command. If that doesn't work for you, try `gpg` instead. If that still doesn't work, please consult the documentation for your specific program (see links above) and the [troubleshooting FAQ](#) below.

## How to import and authenticate the Qubes Master Signing Key

Many important Qubes OS Project assets (e.g., ISOs, RPMs, TGZs, and Git objects) are digitally signed by an official team member's key or by a release signing key (RSK). Each such key is, in turn, signed by the **Qubes Master Signing Key (QMSK)** (`0x427F11FD0FAA4B080123F01CDDFA1A3E36879494`). In this way, the QMSK is the ultimate root of trust for the Qubes OS Project.

The developer signing keys are set to expire after one year, while the QMSK and RSKs have no expiration date. The QMSK was generated on and is kept only on a dedicated, air-gapped "vault" machine, and the private portion will (hopefully) never leave this isolated machine.

Before we proceed, you must first complete the prerequisite step of [installing OpenPGP software](#).

Once you have appropriate OpenPGP software installed, there are several ways to get the QMSK.

- If you're on Qubes OS, it's available in every qube (except `dom0`):

```
$ gpg2 --import /usr/share/qubes/qubes-master-key.asc
```

- If you're on Fedora, you can get it in the `distribution-gpg-keys` package:

```
$ dnf install distribution-gpg-keys
$ gpg2 --import /usr/share/distribution-gpg-keys/qubes/*
```

- If you're on Debian, it may already be included in your keyring.
- Fetch it with GPG:

```
$ gpg2 --fetch-keys https://keys.qubes-os.org/keys/qubes-master-signing-key.asc
```

- Get it from a public [keyserver](#) (specified on first use with `--keyserver <URI>` along with keyserver options to include key signatures), e.g.:

```
$ gpg2 --keyserver-options no-self-sigs-only,no-import-clean --keyserver hkp://
↪keyserver.ubuntu.com --recv-keys 0x427F11FD0FAA4B080123F01CDDFA1A3E36879494
```

- Download it as a file, then import the file. Here are some example download locations:

- [Qubes security pack](#)
- [Qubes keyserver](#)
- [Email to qubes-devel](#)
- [Email to qubes-users](#)

Once you have the key as a file, import it:

```
$ gpg2 --import /<PATH_TO_FILE>/qubes-master-signing-key.asc
```

Once you’ve obtained the QMSK, you must verify that it’s authentic rather than a forgery. Anyone can create a PGP key with the name “Qubes Master Signing Key” and the short key ID `0x36879494`, so you cannot rely on these alone. You also should not rely on any single website, not even over HTTPS.

So, what *should* you do? One option is to use the PGP [Web of Trust](#). In addition, some operating systems include the means to acquire the QMSK securely. For example, on Fedora, `dnf install distribution-gpg-keys` will get you the QMSK along with several other Qubes keys. On Debian, your keyring may already contain the necessary keys.

Perhaps the most common route is to rely on the key’s fingerprint, which is a string of 40 alphanumeric characters, like this:

```
427F 11FD 0FAA 4B08 0123 F01C DDFA 1A3E 3687 9494
```

Every PGP key has one of these fingerprints, which uniquely identifies it among all PGP keys. (On the command line, you can view a key’s fingerprint with the `gpg2 --fingerprint <KEY_ID>` command.) Therefore, if you know the genuine QMSK fingerprint, then you always have an easy way to confirm whether any purported copy of it is authentic, simply by comparing the fingerprints.

But how do you know which fingerprint is the real one? After all, *this website could be compromised*, so the fingerprint you see here may not be genuine. That’s why we strongly suggest obtaining the fingerprint from *multiple independent sources in several different ways*, then comparing the strings of letters and numbers to make sure they match.

For the purpose of convincing yourself that you know the authentic QMSK fingerprint, spaces and capitalization don’t matter. In other words, all of these fingerprints are considered the same:

```
427F 11FD 0FAA 4B08 0123 F01C DDFA 1A3E 3687 9494
427f 11fd 0faa 4b08 0123 f01c ddfa 1a3e 3687 9494
427F11FD0FAA4B080123F01CDDFA1A3E36879494
427f11fd0faa4b080123f01cddfa1a3e36879494
```

Instead, what matters is that *all* the characters are present in *exactly* the same order. If even one character is different, the fingerprints should not be considered the same. Even if two fingerprints have all the same characters, if any of those characters are in a different order, sequence, or position, then the fingerprints should not be considered the same.

However, for the purpose of *searching for*, *looking up*, or *entering* keys, spaces and capitalization can matter, depending on the software or tool you’re using. You may need to try different variations (e.g., with and without spaces). You may also sometimes see (or need to enter) the entire fingerprint prefixed with `0x`, as in:

```
0x427F11FD0FAA4B080123F01CDDFA1A3E36879494
0x427f11fd0faa4b080123f01cddfa1a3e36879494
```

The `0x` prefix is sometimes used to indicate that the string following it is a hexadecimal value, and some PGP-related tools may require this prefix. Again, for the purpose of convincing yourself that you know the authentic QMSK fingerprint, you may safely ignore the `0x` prefix, as it is not part of the fingerprint. As long as the 40-character string after the `0x` matches exactly, the fingerprint is considered the same. The `0x` prefix only matters if the software or tool you’re using cares about it.

The general idea of “comparing fingerprints” is to go out into the world (whether digitally, physically, or both) and find other 40-character strings purporting to be the QMSK fingerprint, then compare them to your own purported QMSK fingerprint to ensure that the sequence of alphanumeric characters is exactly the same (again, regardless of spaces or capitalization). If any of the characters do not match or are not in the same order, then at least one of the fingerprints is a forgery. Here are some ideas to get you started:

- Check the fingerprint on various websites (e.g., [mailing lists](#), [discussion forums](#), [social media](#), [personal websites](#)).
- Check against PDFs, photographs, and videos in which the fingerprint appears (e.g., [slides from a talk](#), on a [T-shirt](#), or in the [recording of a presentation](#)).
- Ask people to post the fingerprint on various mailing lists, forums, and chat rooms.
- Download old Qubes ISOs from different sources and check the included Qubes Master Signing Key.
- Repeat the above over Tor.
- Repeat the above over various VPNs and proxy servers.
- Repeat the above on different networks (work, school, internet cafe, etc.).
- Text, email, call, video chat, snail mail, or meet up with people you know to confirm the fingerprint.
- Repeat the above from different computers and devices.

Once you’ve observed enough matching fingerprints from enough independent sources in enough different ways that you feel confident that you have the genuine fingerprint, keep it in a safe place. Every time you need to check whether a key claiming to be the QMSK is authentic, compare that key’s fingerprint to your trusted copy and confirm they match.

Now that you’ve imported the authentic QMSK, set its trust level to “ultimate” so that it can be used to automatically verify all the keys signed by the QMSK (in particular, RSKs).

```
$ gpg2 --edit-key 0x427F11FD0FAA4B080123F01CDDFA1A3E36879494
gpg (GnuPG) 1.4.18; Copyright (C) 2014 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

pub 4096R/36879494  created: 2010-04-01  expires: never      usage: SC
                        trust: unknown    validity: unknown
[ unknown] (1). Qubes Master Signing Key

gpg> fpr
pub 4096R/36879494 2010-04-01 Qubes Master Signing Key
Primary key fingerprint: 427F 11FD 0FAA 4B08 0123  F01C DDFA 1A3E 3687 9494

gpg> trust
pub 4096R/36879494  created: 2010-04-01  expires: never      usage: SC
                        trust: unknown    validity: unknown
[ unknown] (1). Qubes Master Signing Key

Please decide how far you trust this user to correctly verify other users' keys
(by looking at passports, checking fingerprints from different sources, etc.)

 1 = I don't know or won't say
 2 = I do NOT trust
 3 = I trust marginally
 4 = I trust fully
 5 = I trust ultimately
m = back to the main menu
```

(continues on next page)

(continued from previous page)

```

Your decision? 5
Do you really want to set this key to ultimate trust? (y/N) y

pub 4096R/36879494  created: 2010-04-01  expires: never      usage: SC
                        trust: ultimate    validity: unknown
[ unknown] (1). Qubes Master Signing Key
Please note that the shown key validity is not necessarily correct
unless you restart the program.

gpg> q

```

Now, when you import any of the release signing keys and many Qubes team member keys, they will already be trusted in virtue of being signed by the QMSK.

As a final sanity check, make sure the QMSK is in your keyring with the correct trust level.

```

$ gpg2 -k "Qubes Master Signing Key"
pub  rsa4096 2010-04-01 [SC]
    427F11FD0FAA4B080123F01CDDFA1A3E36879494
uid          [ultimate] Qubes Master Signing Key

```

If you don't see the QMSK here with a trust level of "ultimate," go back and follow the instructions in this section carefully and consult the [troubleshooting FAQ](#) below.

## How to import and authenticate release signing keys

Every Qubes OS release is signed by a **release signing key (RSK)**, which is, in turn, signed by the Qubes Master Signing Key (QMSK).

Before we proceed, you must first complete the following prerequisite steps:

1. *Install OpenPGP software.*
2. *Import and authenticate the QMSK.*

After you have completed these two prerequisite steps, the next step is to obtain the correct RSK. The filename pattern for RSKs is `qubes-release-X-signing-key.asc`, where `X` is either a major or minor Qubes release number, such as 4 or 4.2. There are several ways to get the RSK for your Qubes release.

- If you have access to an existing Qubes installation, the release keys are available in dom0 in `/etc/pki/rpm-gpg/RPM-GPG-KEY-qubes-*`. These can be *copied* into other qubes for further use. In addition, every other qube contains the release key corresponding to that installation's release in `/etc/pki/rpm-gpg/RPM-GPG-KEY-qubes-*`. If you wish to use one of these keys, make sure to import it into your keyring, e.g.:

```
$ gpg2 --import /etc/pki/rpm-gpg/RPM-GPG-KEY-qubes-*
```

- Fetch it with GPG:

```
$ gpg2 --keyserver-options no-self-sigs-only,no-import-clean --fetch-keys https://
↪keys.qubes-os.org/keys/qubes-release-X-signing-key.asc
```

- Download it as a file. You can find the RSK for your Qubes release on the [downloads page](#). You can also download all the currently used developers' signing keys, RSKs, and the Qubes Master Signing Key from the [Qubes security pack](#) and the [Qubes keyserver](#). Once you've downloaded your RSK, import it with GPG:

```
$ gpg2 --keyserver-options no-self-sigs-only,no-import-clean --import ./qubes-
↪release-X-signing-key.asc
```

Now that you have the correct RSK, you simply need to verify that it is signed by the QMSK:

```
$ gpg2 --check-signatures "Qubes OS Release X Signing Key"
pub  rsa4096 2017-03-06 [SC]
    5817A43B283DE5A9181A522E1848792F9E2795E9
uid      [ full ] Qubes OS Release X Signing Key
sig!3    1848792F9E2795E9 2017-03-06 Qubes OS Release X Signing Key
sig!     DDFA1A3E36879494 2017-03-08 Qubes Master Signing Key

gpg: 2 good signatures
```

This is just an example, so the output you receive may not look exactly the same. What matters is the line with a `sig!` prefix showing that the QMSK has signed this key. This verifies the authenticity of the RSK. Note that the `!` flag after the `sig` tag is important because it means that the key signature is valid. A `sig-` prefix would indicate a bad signature, and `sig%` would mean that `gpg` encountered an error while verifying the signature. It is not necessary to independently verify the authenticity of the RSK, since you already verified the authenticity of the QMSK.

As a final sanity check, make sure the RSK is in your keyring with the correct trust level:

```
$ gpg2 -k "Qubes OS Release"
pub  rsa4096 2017-03-06 [SC]
    5817A43B283DE5A9181A522E1848792F9E2795E9
uid      [ full ] Qubes OS Release X Signing Key
```

If you don't see the correct RSK here with a trust level of “full” or higher, go back and follow the instructions in this section carefully, and consult the [troubleshooting FAQ](#) below.

## How to obtain and authenticate other signing keys

Please see the [Qubes security pack](#) documentation.

## How to verify the cryptographic hash values of Qubes ISOs

There are two ways to verify Qubes ISOs: cryptographic hash values and detached PGP signatures. Both methods are equally secure. Using just one method is sufficient to verify your Qubes ISO. Using both methods is not necessary, but you can do so if you like. One method might be more convenient than another in certain circumstances, so we provide both. This section covers cryptographic hash values. For the other method, see [how to verify detached PGP signatures on Qubes ISOs](#).

Before we proceed, you must first complete the following prerequisite steps:

1. *Install OpenPGP software.*
2. *Import and authenticate the Qubes Master Signing Key.*
3. *Import and authenticate your release signing key.*

Each Qubes ISO is accompanied by a set of **cryptographic hash values** contained in a plain text file ending in `.DIGESTS`, which can find on the downloads page alongside the ISO. This file contains the output of running several different cryptographic hash functions on the ISO (a process known as “hashing”) to obtain alphanumeric outputs known as “hash values” or “digests.”



One convenient property of hash values is that they can be generated on any computer. This means, for example, that you can download a Qubes ISO on one computer, hash it, then visually compare that hash value to the one you generated or have saved on a different computer.

In addition to the .DIGESTS files on the downloads page alongside each ISO, and you can always find all the digest files for every Qubes ISO in the [Qubes security pack](#).

If the filename of your ISO is Qubes-RX-x86\_64.iso, then the name of the digest file for that ISO is Qubes-RX-x86\_64.iso.DIGESTS, where X is a specific release of Qubes. The digest filename is always the same as the ISO filename followed by .DIGESTS. Since the digest file is a plain text file, you can open it with any text editor. Inside, you should find text that looks similar to this:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA256

3c951138b8b9867d8657f173c1b58b82 *Qubes-RX-x86_64.iso
1fc9508160d7c4cba6cacc3025165b0f996c843f *Qubes-RX-x86_64.iso
6b998045a513dcdd45c1c6e1ace4f1b4e7eff799f381dccb9eb0170c80f678a *Qubes-RX-x86_64.iso
de1eb2e76bdb48559906f6fe344027ece20658d4a7f04ba00d4e40c63723171c62bdcc869375e7a4a4499d7bfff484d7a621c3ac
↪ *Qubes-RX-x86_64.iso
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v2

iQIcBAEBCAAGBQJX4X0/AAoJEMsRyh0D+lCCL9sP/jlZ26zhv1DEX/eaA/ANa/6b
Dpsh/sqZEpz1S0wUxdm0gS+anc8nSDoCQSMBxnafuBbmwTChdHI/P7NvNirCULma
9nw+EYCsCiNZ9+WCeroR8XDFSiDjvfkve0R8nwfma1XDqu1bN2ed4n/zNoGgQ8w0
t5LEVDKCVJ+65pI7RzOSMbWaw+uWfGehbgumD7a6rfe0qOTONoZ0jJJTnM0+NFJF
Qz5yBg+0FYc7FmfX+tY801AwSyevj3LKgqZN1GVcU9hhoHH7f2BcbdNk9I5WHHq
doKmnZtcdyadQgWMNB68Wu9+0CWsXvk6E00QfW69M4d6w0gbyoJyUL1uzxgixb50
godxrqeitXQSZZvU4kom5zlSjqZs4dGK+Ueplpkr8voT8TSWer0Nbh/VMfrNS1z
0/j+e/KMjor7XxehR+XhNwa2YLjA5l5H9rP+Ct/LAfVFP4uhsAnYf0rUskhCStxf
Zmtqz4F0w/iSz00s+IVcnRcyTYWh3e9XaW56b9J/ou0wlwmJ7oJuEikOHBDjrUph
2a8AM+QzNmnc0tDBWTtT2frXcotqL+Evp/kQr5G5pJM/mTR5EQm7+LKS17yCPoCj
g8JqGYYptgkxjQdX3YAy9VDS CJ/6EkFc2lkQHbgZxjXqyrEMbgeSxtMltZ7cCqw1
3N/6YZw1gSuvBlTquP27
=e9oD
-----END PGP SIGNATURE-----
```

Four digests have been computed for this ISO. The hash functions used, in order from top to bottom, are MD5, SHA-1, SHA-256, and SHA-512. One way to verify that the ISO you downloaded matches any of these hash values is by using the respective `*sum` command:

```
$ md5sum -c Qubes-RX-x86_64.iso.DIGESTS
Qubes-RX-x86_64.iso: OK
md5sum: WARNING: 23 lines are improperly formatted
$ sha1sum -c Qubes-RX-x86_64.iso.DIGESTS
Qubes-RX-x86_64.iso: OK
sha1sum: WARNING: 23 lines are improperly formatted
$ sha256sum -c Qubes-RX-x86_64.iso.DIGESTS
Qubes-RX-x86_64.iso: OK
sha256sum: WARNING: 23 lines are improperly formatted
$ sha512sum -c Qubes-RX-x86_64.iso.DIGESTS
Qubes-RX-x86_64.iso: OK
sha512sum: WARNING: 23 lines are improperly formatted
```



The OK response tells us that the hash value for that particular hash function matches. The program also warns us that there are 23 improperly formatted lines, but this is expected. This is because each file contains lines for several different hash values (as mentioned above), but each `*sum` program verifies only the line for its own hash function. In addition, there are lines for the PGP signature that the `*sum` programs do not know how to read. Therefore, it is safe to ignore these warning lines.

Another way is to use `openssl` to compute each hash value, then compare them to the contents of the digest file:

```
$ openssl dgst -md5 Qubes-RX-x86_64.iso
MD5(Qubes-RX-x86_64.iso)= 3c951138b8b9867d8657f173c1b58b82
$ openssl dgst -sha1 Qubes-RX-x86_64.iso
SHA1(Qubes-RX-x86_64.iso)= 1fc9508160d7c4cba6cacc3025165b0f996c843f
$ openssl dgst -sha256 Qubes-RX-x86_64.iso
SHA256(Qubes-RX-x86_64.iso)=
↳ 6b998045a513dcdd45c1c6e61ace4f1b4e7eff799f381dcc9eb0170c80f678a
$ openssl dgst -sha512 Qubes-RX-x86_64.iso
SHA512(Qubes-RX-x86_64.iso)=
↳ de1eb2e76bdb48559906f6fe344027ece20658d4a7f04ba00d4e40c63723171c62bdcc869375e7a4a4499d7bfff484d7a621c3
```

(Notice that the outputs match the values from the digest file.)

However, it is possible that an attacker replaced `Qubes-RX-x86_64.iso` with a malicious ISO, computed the hash values for that malicious ISO, and replaced the values in `Qubes-RX-x86_64.iso.DIGESTS` with his own set of values. Therefore, we should also verify the authenticity of the listed hash values. Since `Qubes-RX-x86_64.iso.DIGESTS` is a clearsigned PGP file, we can use GPG to verify the signature in the digest file:

```
$ gpg2 -v --verify Qubes-RX-x86_64.iso.DIGESTS
gpg: armor header: Hash: SHA256
gpg: armor header: Version: GnuPG v2
gpg: original file name=''
gpg: Signature made Tue 20 Sep 2016 10:37:03 AM PDT using RSA key ID 03FA5082
gpg: using PGP trust model
gpg: Good signature from "Qubes OS Release X Signing Key"
gpg: textmode signature, digest algorithm SHA256
```

This is just an example, so the output you receive will not look exactly the same. What matters is the line that says Good signature from "Qubes OS Release X Signing Key". This confirms that the signature on the digest file is good.

If you don't see a good signature here, go back and follow the instructions in this section carefully, and consult the [troubleshooting FAQ](#) below.

## How to verify detached PGP signatures on Qubes ISOs

There are two ways to verify Qubes ISOs: cryptographic hash values and detached PGP signatures. Both methods are equally secure. Using just one method is sufficient to verify your Qubes ISO. Using both methods is not necessary, but you can do so if you like. One method might be more convenient than another in certain circumstances, so we provide both. This section covers detached PGP signatures. For the other method, see [how to verify the cryptographic hash values of Qubes ISOs](#).

Before we proceed, you must first complete the following prerequisite steps:

1. *Install OpenPGP software.*
2. *Import and authenticate the Qubes Master Signing Key.*
3. *Import and authenticate your release signing key.*

Every Qubes ISO is released with a **detached PGP signature** file, which you can find on the downloads page alongside the ISO. If the filename of your ISO is `Qubes-RX-x86_64.iso`, then the name of the signature file for that ISO is `Qubes-RX-x86_64.iso.asc`, where X is a specific release of Qubes. The signature filename is always the same as the ISO filename followed by `.asc`.

Download both the ISO and its signature file. Put both of them in the same directory, then navigate to that directory. Now, you can verify the ISO by executing this GPG command in the directory that contains both files:

```
$ gpg2 -v --verify Qubes-RX-x86_64.iso.asc Qubes-RX-x86_64.iso
gpg: armor header: Version: GnuPG v1
gpg: Signature made Tue 08 Mar 2016 07:40:56 PM PST using RSA key ID 03FA5082
gpg: using PGP trust model
gpg: Good signature from "Qubes OS Release X Signing Key"
gpg: binary signature, digest algorithm SHA256
```

This is just an example, so the output you receive will not look exactly the same. What matters is the line that says `Good signature from "Qubes OS Release X Signing Key"`. This confirms that the signature on the ISO is good.

If you don't see a good signature here, go back and follow the instructions in this section carefully, and consult the [troubleshooting FAQ](#) below.

## How to re-verify installation media after writing

*This is an optional section intended for advanced users.*

After you have authenticated your Qubes ISO and written it onto your desired medium (such as a USB drive or optical disc), you can re-verify the data that has been written to your medium. Why would you want to do this when you've already verified the original ISO? Well, it's conceivable that a sufficiently sophisticated adversary might allow your initial ISO verification to succeed (so as not to alert you that your machine has been compromised, for example), then surreptitiously modify the data as it is being written onto your installation medium, resulting in a compromised Qubes installer. This might increase the odds that the attack goes undetected. One way to mitigate this risk is to re-verify the installer after writing it onto an installation medium that cannot be altered, such as a USB drive with a properly-implemented physical write-protect switch and firmware that is either unflashable or cryptographically-signed (or both), as discussed in our [installation security considerations](#).

This section will walk through an example of re-verifying the installer on such a device. We begin by assuming that you have just [written your desired Qubes ISO onto the USB drive](#). First, unplug your USB drive and flip the write protect switch so that the data on the drive can no longer be altered. If you have a different computer from the one you used to create the installation medium, consider using that computer. If not, try to at least use a fresh VM (e.g., if it's a Qubes system). The idea is that the original machine may have been compromised, and using a different one for re-verification forces your hypothetical adversary to compromise an additional machine in order to succeed.

Now, our goal is to perform the same verification steps as we did with the original ISO, except, this time, we'll be reading the installer data directly from the write-protected USB drive instead of from the original ISO file. First, let's compute the SHA-256 hash value of the data on the drive. (This assumes you're already familiar with [how to verify the cryptographic hash values of Qubes ISOs](#).) In order to do this, we have to know the exact size, in bytes, of the original ISO. There are two ways to get this information: from the ISO itself and from the Qubes website. Here's an example of the first way:

```
$ dd if=/dev/sdX bs=1M count=$(stat -c %s /path/to/iso) iflag=count_bytes | sha256sum
```

(Where `/dev/sdX` is your USB drive and `/path/to/iso` is the path to your Qubes ISO.)

This command reads exactly the number of bytes of your Qubes ISO (obtained with `stat -c %s /path/to/iso`) from the USB drive and pipes them into `sha256sum`. The output should look something like this:

```
0e68dd3347b68618d9e5f3ddb580bf7ecdd2166747630859b3582803f1ca8801 -
5523+0 records in
5523+0 records out
5791285248 bytes (5.8 GB, 5.4 GiB) copied, 76.3369 s, 75.9 MB/s
```

Note that your actual SHA-256 hash value and byte number will depend on which Qubes ISO you're using. This is just an example. Your SHA-256 hash value should match the hash value of your genuine original Qubes ISO.

Now, reading the number of bytes directly from the ISO is fine, but you may be concerned that a sufficiently sophisticated adversary may have compromised the machine on which you're performing this re-verification and may therefore be capable of feeding you a false success result. After all, if your adversary knows the answer you're looking for — namely, a match to the genuine ISO — and has access to that very ISO in the same re-verification environment, then there is little to prevent him from simply hashing the original ISO and feeding you that result (perhaps while also reading from the USB drive and piping it into `/dev/null` so that you see the light on the USB drive blinking to support the illusion that the data is being read from the USB drive).

Therefore, in order to make things a bit more difficult for your hypothetical adversary, you may instead wish to perform the re-verification in an environment that has never seen the original ISO, e.g., a separate offline computer or a fresh VM the storage space of which is too small to hold the ISO. (Note: If you're doing this in Qubes, you can attach the block device from `sys-usb` to a separate new qube. You don't have to perform the re-verification directly in `sys-usb`.) In that case, you'll have to obtain the size of the ISO in bytes and enter it into the above command manually. You can, of course, obtain the size by simply using the `stat -c %s /path/to/iso` command from above on the machine that has the ISO. You can also obtain it from the Qubes website by hovering over any ISO download button on the downloads page. (You can also view these values directly in the downloads page's [source data](#).) Once you have the exact size of the ISO in bytes, simply insert it into the same command, for example:

```
$ dd if=/dev/sdX bs=1M count=5791285248 iflag=count_bytes | sha256sum
```

If you wish to compute the values of other hash functions, you can replace `sha256sum`, e.g., with `md5sum`, `sha1sum`, or `sha512sum`.

In addition to checking hash values, you can also use GnuPG to verify the detached PGP signature directly against the data on the USB drive. (This assumes you're already familiar with [how to verify detached PGP signatures on Qubes ISOs](#).)

```
$ dd if=/dev/sdX bs=1M count=<ISO_SIZE> iflag=count_bytes | gpg -v --verify Qubes-RX-x86_
↳ 64.iso.asc -
gpg: Signature made Thu 14 Jul 2022 08:49:38 PM PDT
gpg:          using RSA key 5817A43B283DE5A9181A522E1848792F9E2795E9
gpg: using pgp trust model
gpg: Good signature from "Qubes OS Release X Signing Key" [full]
gpg: binary signature, digest algorithm SHA256, key algorithm rsa4096
5523+0 records in
5523+0 records out
5791285248 bytes (5.8 GB, 5.4 GiB) copied, 76.6013 s, 75.6 MB/s
```

(Where `/dev/sdX` is your USB drive, `<ISO_SIZE>` is the size of the original ISO in bytes, and `Qubes-RX-x86_64.iso.asc` is the detached signature file of the original ISO.)

This command reads the exact number of bytes from your USB drive as the size of the original ISO and pipes them into `gpg`. The usual form of a `gpg` verification command is `gpg --verify <SIGNATURE> <SIGNED_DATA>`. Our command is using shell redirection in order to use data from your USB drive as the `<SIGNED_DATA>`, which is why the `-` at the end of the command is required. Remember that you still must have properly imported and trusted the [QMSK](#) and appropriate [RSK](#) in order for this to work. You should receive a Good signature message for the appropriate RSK, which should be signed by a copy of the QMSK that you previously confirmed to be genuine.

## How to verify signatures on Git repository tags and commits

Before we proceed, you must first complete the following prerequisite steps:

1. *Install OpenPGP software.*
2. *Import and authenticate the Qubes Master Signing Key.*
3. *Import and authenticate keys from the Qubes security pack (qubes-secpack).* Please see our *PGP key policies* for important information about these keys.

Whenever you use one of the [Qubes repositories](#), you should use Git to verify the PGP signature in a tag on the latest commit or on the latest commit itself. (One or both may be present, but only one is required.) If there is no trusted signed tag or commit on top, any commits after the latest trusted signed tag or commit should **not** be trusted. If you come across a repo with any unsigned commits, you should not add any of your own signed tags or commits on top of them unless you personally vouch for the trustworthiness of the unsigned commits. Instead, ask the person who pushed the unsigned commits to sign them.

You should always perform this verification on a trusted local machine with properly authenticated keys rather than relying on a third party, such as GitHub. While the GitHub interface may claim that a commit has a verified signature from a member of the Qubes team, this is only trustworthy if GitHub has performed the signature check correctly, the account identity is authentic, an admin has not replaced the user's key, GitHub's servers have not been compromised, and so on. Since there's no way for you to be certain that all such conditions hold, you're much better off verifying signatures yourself. (Also see: *distrusting the infrastructure*.)

### How to verify a signature on a Git tag

```
$ git tag -v <tag name>
```

or

```
$ git verify-tag <tag name>
```

### How to verify a signature on a Git commit

```
$ git log --show-signature <commit ID>
```

or

```
$ git verify-commit <commit ID>
```

## Troubleshooting FAQ

### Why am I getting “Can’t check signature: public key not found”?

You don't have the correct *release signing key*.

### Why am I getting “BAD signature from ‘Qubes OS Release X Signing Key’”?

The problem could be one or more of the following:

- You’re trying to verify the wrong file(s). Reread this page carefully.
- You’re using the wrong GPG command. Follow the provided examples carefully, or try using `gpg` instead of `gpg2` (or vice versa).
- The ISO or *detached PGP signature file* is bad (e.g., incomplete or corrupt download). Try downloading the signature file again from a different source, then try verifying again. If you still get the same result, try downloading the ISO again from a different source, then try verifying again.

### Why am I getting “bash: gpg2: command not found”?

You don’t have `gpg2` installed. Please install it using the method appropriate for your environment (e.g., via your package manager), or try using `gpg` instead.

### Why am I getting “No such file or directory”?

Your working directory does not contain the required files. Go back and follow the instructions more carefully, making sure that you put all required files in the same directory *and* navigate to that directory.

### Why am I getting “can’t open signed data ‘Qubes-RX-x86\_64.iso’ / can’t hash datafile: file open error”?

The correct ISO is not in your working directory.

### Why am I getting “can’t open ‘Qubes-RX-x86\_64.iso.asc’ / verify signatures failed: file open error”?

The correct *detached PGP signature file* is not in your working directory.

### Why am I getting “no valid OpenPGP data found”?

Either you don’t have the correct *detached PGP signature file*, or you inverted the arguments to `gpg2`. (The signature file goes first.)

### Why am I getting “WARNING: This key is not certified with a trusted signature! There is no indication that the signature belongs to the owner.”?

There are several possibilities: - You don’t have the *Qubes Master Signing Key*. - You have not *set the Qubes Master Signing Key’s trust level correctly*. - In the case of a key that is not directly signed by the Qubes Master Signing Key, you have not *set that key’s trust level correctly*.

### Why am I getting “X signature not checked due to a missing key”?

You don’t have the keys that created those signatures in your keyring. For the purpose of verifying a Qubes ISO, you don’t need them as long as you have the *Qubes Master Signing Key* and the *release signing key* for your Qubes release.

### Why am I seeing additional signatures on a key with “[User ID not found]” or from a revoked key?

This is just a fundamental part of how OpenPGP works. Anyone can sign anyone else’s public key and upload the signed public key to keyservers. Everyone is also free to revoke their own keys at any time (assuming they possess or can create a revocation certificate). This has no impact on verifying Qubes ISOs, code, or keys.

### Why am I getting “verify signatures failed: unexpected data”?

You’re not verifying against the correct *detached PGP signature file*.

### Why am I getting “not a detached signature”?

You’re not verifying against the correct *detached PGP signature file*.

### Why am I getting “CRC error; [...] no signature found [...]”?

You’re not verifying against the correct *detached PGP signature file*, or the signature file has been modified. Try downloading it again or from a different source.

### Do I have to verify both the detached PGP signature file and the cryptographic hash values?

No, either method is sufficient by itself, but you can do both if you like.

### Why am I getting “no properly formatted X checksum lines found”?

You’re not checking the correct *cryptographic hash values*.

### Why am I getting “WARNING: X lines are improperly formatted”?

Read *how to verify the cryptographic hash values of Qubes ISOs* again.

### Why am I getting “WARNING: 1 listed file could not be read”?

The correct ISO is not in your working directory.

## I have another problem that isn't mentioned here.

Carefully reread this page to be certain that you didn't skip any steps. In particular, make sure you have the [Qubes Master Signing Key](#), the [release signing key](#) for your Qubes release, and the [cryptographic hash values](#) and/or [detached PGP signature file](#), all for the *correct* Qubes OS release. If your question is about GPG, please see the [GnuPG documentation](#). Still have question? Please see [help](#), [support](#), [mailing lists](#), and [forum](#) for places where you can ask!

## 1.13 Developer Documentation

Core documentation for Qubes developers and advanced users.

### 1.13.1 Package contributions

*This page is for developers who wish to contribute packages. If you want to install contributed packages, please see [installing contributed packages](#).*

We're very grateful to the talented and hard-working community members who contribute software packages to Qubes OS. This page explains the inclusion criteria and procedures for such packages, as well as the roles and responsibilities of those involved.

#### Inclusion Criteria

In order to be accepted, packages must:

- In no way weaken the security of Qubes OS.
- Be published under an open-source license (read about the [Qubes OS License](#)).
- Follow our [coding guidelines](#).
- Be thoroughly tested.
- Have a clearly-defined use case for Qubes users.
- Not be unduly burdensome to review.

(Please note that we always reserve the right to add criteria to this list.)

#### Contribution Procedure

Before you start putting serious work into a package, we recommend that you discuss your idea with the Qubes developers and the broader community on the [qubes-devel mailing list](#). Once you have a package that's ready to become part of Qubes OS, please follow this procedure:

1. Ensure that your package satisfies the [Inclusion Criteria](#).
2. If your code isn't already on GitHub, create a GitHub repo that contains your code. You can have a look to an example package called [qubes-skeleton](#).
3. If you haven't already, [sign your code](#).
4. Create an issue in [qubes-issues](#) with the title [Contribution] your-package-name. Include a link to your repo, a brief description of your package, and a brief explanation of why you think it should be included in Qubes. Please note that the Qubes core developers are very busy. If they are under heavy load when you submit your contribution, it may be a very long time before they have time to review your package. If this happens, please do not be discouraged. If you think they may have forgotten about your pending contribution, you may "bump"



your request by commenting on your issue, but please do this *very* sparingly (i.e., no more than once a month). We appreciate your understanding!

5. You may be asked followup questions. If we decide to accept your contribution, you will be invited to join the [QubesOS-contrib](#) organization on GitHub as public recognition of your contribution (but without push access; see [Review Procedure](#)), and [QubesOS-contrib](#) will fork your repo. If we decide not to accept your contribution, we will state the reason and close the issue.

### Update Procedure

*Anyone* can provide an update (patch) to a contributed package, not just the person who contributed that package! The update procedure is the same for everyone, including the original package contributor.

If you would like to update an already-contributed package (specifically, a fork owned by [QubesOS-contrib](#)), please submit a *signed*, fast-forwardable pull request to that repo with your changes. Please note that your pull request **must** be both *signed* and fast-forwardable, or else it will be closed without further review. One or more reviewers may post comments on your pull request. Please be prepared to read and respond to these comments.

### Review Procedure

This review procedure covers both original package contributions (see [Contribution Procedure](#)) and all subsequent updates to those packages, including updates from the original package contributor (see [Update Procedure](#)). All changes will be reviewed by a Qubes Core Reviewer (QCR) and the [Package Maintainer](#) (PM). In all cases, the QCR will be a core Qubes developer. In some cases, the QCR and the PM will be the same person. For example, if someone contributes a package, then disappears, and no suitable replacement has been found, then it is likely that a core Qubes developer will play both the QCR and PM roles for that package, at least until another suitable candidate volunteers to become the PM for that package.

The review procedure is as follows:

1. Someone, S, wishes to make a change to a package, P.
2. S submits a fast-forwardable pull request against the fork of P's repo owned by [QubesOS-contrib](#).
3. The PM reviews the pull request. If the pull request passes the PM's review, the PM adds a *signed comment* on the pull request stating that it has passed review. (In cases in which S = PM, the PM can simply add a *signed tag* to the HEAD commit prior to submitting the pull request.) If the pull request does not pass the PM's review, the PM leaves a comment on the pull request explaining why not.
4. The QCR reviews the pull request. If the pull request passes the QCR's review, the QCR pushes a *signed tag* to the HEAD commit stating that it has passed review and fast-forward merges the pull request. If the pull request does not pass the QCR's review, the QCR leaves a comment on the pull request explaining why not, and the QCR may decide to close the pull request.

In all the cases, the first condition to be validated by the QCR's review is to ensure that the contribution **will not** hijack any core packages of [QubesOS](#) and of course, none of the [QubesOS-contrib](#) packages too. More precisely, particular attention to the whole build pipeline will be made with a specific review of:

- Package dependencies,
- Build scripts (including downloaded ones),
- All downloaded components should be verified against static hash,
- RPM/DEB installation scripts (e.g. looking at constraints who would hijack other packages),
- Makefiles,
- Package build [reproducible](#)



and any steps which would result in partial/total compromise of legitimate components. For this part, you can have a look to an example package called [qubes-skeleton](#).

## Package Maintainers

If you contribute a package, we assume that you will be the maintainer of that package, unless you tell us otherwise. As the maintainer of the package, it is your privilege and responsibility to:

- [Review](#) each pull request made against the package.
- Decide when the package has reached a new version, and notify the Qubes core developers when this occurs.

If you do not wish to be the maintainer of your package, please let us know. If you do not act on your maintainer duties for a given package for an extended period of time and after at least one reminder, we will assume that you no longer wish to be the maintainer for that package.

## 1.13.2 Google Summer of Code (GSoC)

### Information for Students

Thank you for your interest in participating in the [Google Summer of Code](#) program with the Qubes OS team. You can read more about the Google Summer of Code program at the [official website](#) and the [official FAQ](#).

Being accepted as a Google Summer of Code contributor is quite competitive. If you are interested in participating in the Summer of Code please be aware that you must be able to produce code for Qubes OS for 3-5 months. Your mentors, Qubes developers, will dedicate a portion of their time towards mentoring you. Therefore, we seek candidates who are committed to helping Qubes long-term and are willing to do quality work and be proactive in communicating with your mentor.

You don't have to be a proven developer – in fact, this whole program is meant to facilitate joining Qubes and other free and open source communities. The Qubes community maintains information about [contributing to Qubes development](#) and [how to send patches](#). In order to contribute code to the Qubes project, you must be able to [sign your code](#).

You should start learning the components that you plan on working on before the start date. Qubes developers are available on the [mailing lists](#) for help. The GSoC timeline reserves a lot of time for bonding with the project – use that time wisely. Good communication is key, you should plan to communicate with your team daily and formally report progress and plans weekly. Students who neglect active communication will be failed.

### Overview of Steps

- Join the [qubes-devel list](#) and introduce yourself, and meet your fellow developers
- Read [Google's instructions for participating](#) and the [GSoC Student Manual](#)
- Take a look at the list of ideas below
- Come up with a project that you are interested in (and feel free to propose your own! Don't feel limited by the list below.)
- Read the Contributor Proposal guidelines below
- Write a first draft proposal and send it to the qubes-devel mailing list for review
- Submit proposal using [Google's web interface](#) ahead of the deadline (this requires a Google Account!)
- Submit proof of enrollment well ahead of the deadline

Coming up with an interesting idea that you can realistically achieve in the time available to you (one summer) is probably the most difficult part. We strongly recommend getting involved in advance of the beginning of GSoC, and we will look favorably on applications from prospective contributors who have already started to act like free and open source developers.

Before the summer starts, there are some preparatory tasks which are highly encouraged. First, if you aren't already, definitely start using Qubes as your primary OS as soon as possible! Also, it is encouraged that you become familiar and comfortable with the Qubes development workflow sooner than later. A good way to do this (and also a great way to stand out as an awesome applicant and make us want to accept you!) might be to pick up some issues from [qubes-issues](#) (our issue-tracking repo) and submit some patches addressing them. Some suitable issues might be those with tags “help wanted” and “P: minor” (although more significant things are also welcome, of course). Doing this will get you some practice with [qubes-builder](#), our code-signing policies, and some familiarity with our code base in general so you are ready to hit the ground running come summer.

### Contributor proposal guidelines

A project proposal is what you will be judged upon. Write a clear proposal on what you plan to do, the scope of your project, and why we should choose you to do it. Proposals are the basis of the GSoC projects and therefore one of the most important things to do well.

Below is the application template:

#### # Introduction

Every software project should solve a problem. Before offering the solution (your Google Summer of Code project), you should first define the problem. What's the current state of things? What's the issue you wish to solve and why? Then you should conclude with a sentence or two about your solution. Include links to discussions, features, or bugs that describe the problem further if necessary.

#### # Project goals

Be short and to the point, and perhaps format it as a list. Propose a clear list of deliverables, explaining exactly what you promise to do and what you do not plan to do. “Future developments” can be mentioned, but your promise for the Google Summer of Code term is what counts.

#### # Implementation

Be detailed. Describe what you plan to do as a solution for the problem you defined above. Include technical details, showing that you understand the technology. Illustrate key technical elements of your proposed solution in reasonable detail.

#### # Timeline

Show that you understand the problem, have a solution, have also broken it down into manageable parts, and that you have a realistic plan on how to accomplish your goal. Here you set expectations, so don't make promises you can't keep. A modest, realistic and detailed timeline is better than promising the impossible.

If you have other commitments during GSoC, such as a job, vacation, exams, internship, seminars, or papers to write, disclose them here. GSoC should be treated like a full-time job, and we will expect approximately 40 hours of work per week. If you have conflicts, explain how you will work around them. If you are found to have conflicts which you did not disclose, you may be failed.

(continues on next page)

(continued from previous page)

Open and clear communication is of utmost importance. Include your plans for

- communication in your proposal; daily if possible. You will need to initiate weekly
- formal communications such as a detailed email to the qubes-devel mailing list. Lack
- of communication will result in you being failed.

### # About me

Provide your contact information and write a few sentences about you and why you think

- you are the best for this job. Prior contributions to Qubes are helpful; list your
- commits. Name people (other developers, students, professors) who can act as a
- reference for you. Mention your field of study if necessary. Now is the time to join
- the relevant mailing lists. We want you to be a part of our community, not just
- contribute your code.

Tell us if you are submitting proposals to other organizations, and whether or not you

- would choose Qubes if given the choice.

Other things to think about:

- \* Are you comfortable working independently under a supervisor or mentor who is several
- thousand miles away, and perhaps 12 time zones away? How will you work with your
- mentor to track your work? Have you worked in this style before?
- \* If your native language is not English, are you comfortable working closely with a
- supervisor whose native language is English? What is your native language, as that may
- help us find a mentor who has the same native language?
- \* After you have written your proposal, you should get it reviewed. Do not rely on the
- Qubes mentors to do it for you via the web interface, although we will try to comment
- on every proposal. It is wise to ask a colleague or a developer to critique your
- proposal. Clarity and completeness are important.

## Project Ideas

These project ideas were contributed by our developers and may be incomplete. If you are interested in submitting a proposal based on these ideas, you should contact the [qubes-devel mailing list](#) and associated GitHub issue to learn more about the idea.

### ### Adding a Proposal

**\*\*Project\*\*:** Something that you're totally excited about

**\*\*Brief explanation\*\*:** What is the project, where does the code live?

**\*\*Expected results\*\*:** What is the expected result in the timeframe given

**\*\*Difficulty\*\*:** easy / medium / hard

**\*\*Knowledge prerequisite\*\*:** Pre-requisites for working on the project. What coding

- language and knowledge is needed?

If applicable, links to more information or discussions

**\*\*Size of the project\*\*:** either 175 hours (medium) or 350 hours (large)

(continues on next page)

(continued from previous page)

**\*\*Mentor\*\***: Name and email address.

### Qubes as a Vagrant provider

**Project**: Qubes as a Vagrant provider

**Brief explanation**: Currently using Vagrant on Qubes requires finding an image that uses Docker as isolation provider and running Docker in a qube, or downloading the Vagrantfile and manually setting up a qube according to the Vagrantfile. This project aims at simplifying this workflow. Since introduction of Admin API, it's possible for a qube to provision another qube - which is exactly what is needed for Vagrant. [Related discussion](#)

**Expected results**:

- Design how Vagrant Qubes provider should look like, including:
  - [box format](#)
  - method for running commands inside (ssh vs qvm-run)
- Write a Vagrant provider able to create/start/stop/etc a VM
- Document how to configure and use the provider, including required qrexec policy changes and possibly firewall rules
- Write integration tests

**Difficulty**: medium

**Knowledge prerequisite**:

- Ruby
- Vagrant concepts

**Size of the project**: 350 hours

**Mentor**: [Wojtek Porczyk](#), [Marek Marczykowski-Górecki](#)

### System health monitor

**Project**: System health monitor

**Brief explanation**: A tool that informs the user about common system and configuration issues. Some of this is already available, but scattered across different places. See related issues: [6663](#), [2134](#)

**Expected results**:

- a tool / service that checks for common issues and things needing user attention, for example:
  - some updates to be applied (separate widget already exists)
  - running out of disk space (separate widget already exists)
  - insecure USB configuration (USB in dom0)
  - some system VM crashed
  - ...
- a GUI that provides terse overview of the system state, and notifies the user if something bad happens

**Difficulty:** medium

**Knowledge prerequisite:**

- Python
- basic knowledge about systemd services
- PyGTK (optional)

**Size of the project:** 350 hours

**Mentor:** [Marta Marczykowska-Górecka](#)

## Mechanism for maintaining in-VM configuration

**Project:** Mechanism for maintaining in-VM configuration

**Brief explanation:** Large number of VMs is hard to maintain. Templates helps with keeping them updated, but many applications have configuration in user home directory, which is not synchronized.

**Expected results:**

- Design a mechanism how to *safely* synchronize application configuration living in user home directory (~/.config, some other “dotfiles”). Mechanism should be resistant against malicious VM forcing its configuration on other VMs. Some approach could be a strict control which VM can send what changes (whitelist approach, not blacklist).
- Implementation of the above mechanism.
- Documentation how to configure it securely.

**Difficulty:** medium

**Knowledge prerequisite:**

- shell and/or python scripting
- Qubes OS grexec services

**Size of the project:** 175 hours

**Mentor:** [Frédéric Pierret](#)

## Qubes Live USB

**Project:** Revive Qubes Live USB, integrate it with installer

**Brief explanation:** Qubes Live USB is based on Fedora tools to build live distributions. But for Qubes we need some adjustments: starting Xen instead of Linux kernel, smarter copy-on-write handling (we run there multiple VMs, so a lot more data to save) and few more. Additionally in Qubes 3.2 we have so many default VMs that default installation does not fit in 16GB image (default value) - some subset of those VMs should be chosen. Ideally we’d like to have just one image being both live system and installation image. More details: [#1552](#), [#1965](#).

**Expected results:**

- Adjust set of VMs and templates included in live edition.
- Update and fix build scripts for recent Qubes OS version.
- Update startup script to mount appropriate directories as either copy-on-write (device-mapper snapshot), or tmpfs.

- Optimize memory usage: should be possible to run sys-net, sys-firewall, and at least two more VMs on 4GB machine. This include minimizing writes to copy-on-write layer and tmpfs (disable logging etc).
- Research option to install the system from live image. If feasible add this option.

**Difficulty:** hard

**Knowledge prerequisite:**

- System startup sequence: bootloaders (isolinux, syslinux, grub, UEFI), initramfs, systemd.
- Python and Bash scripting
- Filesystems and block devices: loop devices, device-mapper, tmpfs, overlayfs, sparse files.

**Size of the project:** 350 hours

**Mentor:** [Frédéric Pierret](#)

### LogVM(s)

**Project:** LogVM(s)

**Brief explanation:** Qubes AppVMs do not have persistent /var (on purpose). It would be useful to send logs generated by various VMs to a dedicated log-collecting VM. This way logs will not only survive VM shutdown, but also be immune to altering past entries. See [#830](#) for details.

**Expected results:**

- Design a *simple* protocol for transferring logs. The less metadata (parsed in log-collecting VM) the better.
- Implement log collecting service. Besides logs itself, should save information about logs origin (VM name) and timestamp. The service should *not* trust sending VM in any of those.
- Implement log forwarder compatible with systemd-journald and rsyslog. A mechanism (service/plugin) fetching logs in real time from those and sending to log-collecting VM over qrexec service.
- Document the protocol.
- Write unit tests and integration tests.

**Difficulty:** easy

**Knowledge prerequisite:**

- syslog
- systemd
- Python/Bash scripting

**Size of the project:** 175 hours

**Mentor:** [Frédéric Pierret](#)

## Whonix IPv6 and nftables support

**Project:** Whonix IPv6 and nftables support

**Brief explanation:** [T509](#)

**Expected results:**

- Work at upstream Tor: An older version of [TransparentProxy](#) page was the origin of Whonix. Update that page for nftables / IPv6 support without mentioning Whonix. Then discuss that on the tor-talk mailing list for wider input. [here](#)
- implement corridor feature request add IPv6 support / port to nftables - [issue](#)
- port [whonix-firewall](#) to nftables
- make connections to IPv6 Tor relays work
- make connections to IPv6 destinations work

**Difficulty:** medium

**Knowledge prerequisite:**

- nftables
- iptables
- IPv6

**Size of the project:** 175 hours

**Mentor:** [Patrick Schleizer](#)

## GUI agent for Windows 8/10

**Project:** GUI agent for Windows 8/10

**Brief explanation:** Add support for Windows 8+ to the Qubes GUI agent and video driver. Starting from Windows 8, Microsoft requires all video drivers to conform to the WDDM display driver model which is incompatible with the current Qubes video driver. Unfortunately the WDDM model is much more complex than the old XPDM one and officially *requires* a physical GPU device (which may be emulated). Some progress has been made to create a full WDDM driver that *doesn't* require a GPU device, but the driver isn't working correctly yet. Alternatively, WDDM model supports display-only drivers which are much simpler but don't have access to system video memory and rendering surfaces (a key feature that would simplify seamless GUI mode). [#1861](#)

**Expected results:** Working display-only WDDM video driver or significant progress towards making the full WDDM driver work correctly.

**Difficulty:** hard

**Knowledge prerequisite:** C/C++ languages, familiarity with Windows API, familiarity with the core Windows WDM driver model. Ideally familiarity with the WDDM display driver model.

**Size of the project:** 175 hours

**Mentor:** [Rafał Wojdyła](#)

## **GNOME support in dom0 / GUI VM**

**Project:** GNOME support in dom0

**Brief explanation:** Integrating GNOME into Qubes dom0. This include:

- patching window manager to add colorful borders
- removing stuff not needed in dom0 (file manager(s), indexing services etc)
- adjusting menu for easy navigation (same applications in different VMs and such problems, dom0-related entries in one place)
- More info: [#1806](#)

**Expected results:**

- Review existing support for other desktop environments (KDE, Xfce4, i3, awesome).
- Patch window manager to draw colorful borders (we use only server-side decorations), there is already very similar patch in [Cappsule project](#).
- Configure GNOME to not make use of dom0 user home in visible way (no search in files there, no file manager, etc).
- Configure GNOME to not look into external devices plugged in (no auto mounting, device notifications etc).
- Package above modifications as RPMs, preferably as extra configuration files and/or plugins than overwriting existing files. Exceptions to this rule may apply if no other option.
- Adjust comps.xml (in installer-qubes-os repo) to define package group with all required packages.
- Document installation procedure.

**Difficulty:** hard

**Knowledge prerequisite:**

- GNOME architecture
- C language (patching metacity)
- Probably also javascript - for modifying GNOME shell extensions

**Size of the project:** 175 hours

**Mentor:** [Frédéric Pierret](#), [Marek Marczykowski-Górecki](#)

## **Generalize the Qubes PDF Converter to other types of files**

**Project:** Qubes Converters

**Brief explanation:** One of the pioneering ideas of Qubes is to use disposable virtual machines to convert untrustworthy files (such as documents given to journalists by unknown and potentially malicious whistleblowers) into trustworthy files. See [Joanna's blog on the Qubes PDF Convert](#) for details of the idea. Joanna has implemented a prototype for PDF documents. The goal of this project would be to generalize beyond the simple prototype to accommodate a wide variety of file formats, including Word documents, audio files, video files, spreadsheets, and so on. The converters should prioritise safety over faithful conversion. For example the Qubes PDF converter typically leads to lower quality PDFs (e.g. cut and paste is no longer possible), because this makes the conversion process safer.

**Expected results:** We expect that in the timeframe, it will be possible to implement many converters for many file formats. However, if any unexpected difficulties arise, we would prioritise a small number of safe and high quality converters over a large number of unsafe or unuseful converters.



**Difficulty:** easy

**Knowledge prerequisite:** Most of the coding will probably be implemented as shell scripts to interface with pre-existing converters (such as ImageMagick in the Qubes PDF converter). However, shell scripts are not safe for processing untrusted data, so any extra processing will need to be implemented in another language – probably Python.

**Size of the project:** 175 hours

**Mentors:** Andrew Clausen and Jean-Philippe Ouellet

## Progress towards reproducible builds

**Project:** Progress towards reproducible builds

**Brief explanation:** A long-term goal is to be able to build the entire OS and installation media in a completely bit-wise deterministic manner, but there are many baby steps to be taken along that path. See:

- [“Security challenges for the Qubes build process”](#)
- [This mailing list post](#)
- and [reproducible-builds.org](https://reproducible-builds.org)

for more information and qubes-specific background.

**Expected results:** Significant progress towards making the Qubes build process deterministic. This would likely involve cooperation with and hacking on several upstream build tools to eliminate sources of variability.

**Difficulty:** medium

**Knowledge prerequisite:** qubes-builder [\[1\]](#) [\[2\]](#) [\[3\]](#), and efficient at introspecting complex systems: comfortable with tracing and debugging tools, ability to quickly identify and locate issues within a large codebase (upstream build tools), etc.

**Size of the project:** 350 hours

**Mentor:** [Marek Marczykowski-Górecki](#)

## Porting Qubes to ARM/aarch64

**Project:** Porting Qubes to ARM/aarch64

**Brief explanation:**

Qubes currently only supports the x86\_64 CPU architecture. Xen currently has additional support for ARM32/ARM64 processors, however work needs to be done to integrate this into the Qubes build process, as well as work in integrating this with the Qubes toolstack and security model. This may also be beneficial in simplifying the process of porting to other architectures.

Some related discussion:

- [#4318](#) on porting to ppc64.
- [#3894](#) on porting to L4 microkernel.

**Expected results:**

- Add cross-compilation support to qubes-builder and related components.
- Make aarch64 specific adjustments to Qubes toolstacks/manager (including passthrough of devices from device tree to guest domains).
- Aarch64 specific integration and unit tests.

- Production of generic u-boot or uefi capable image/iso for target hardware.

**Difficulty:** hard

**Knowledge prerequisite:**

- Libvirt and Qubes toolstacks (C and python languages).
- Xen debugging.
- General ARM architecture knowledge.

**Size of the project:** 350 hours

**Mentor:** [Marek Marczykowski-Górecki](#)

### Android development in Qubes

**Project:** Research running Android in Qubes VM (probably HVM) and connecting it to Android Studio

**Brief explanation:** The goal is to enable Android development (and testing!) on Qubes OS. Currently it's only possible using qemu-emulated Android for ARM. Since it's software emulation it's rather slow. Details, reference: [#2233](#)

**Expected results:**

- a simple way of setting up Android qubes with hardware emulation (distributed as a template or as a salt, handling various modern Android versions)
- figuring out and implementing an easy and secure way to connect an Android qube to a development qube with Android studio
- documentation and tests

**Difficulty:** hard

**Knowledge prerequisite:**

**Size of the project:** 350 hours

**Mentor:** Inquire on [qubes-devel](#).

### Admin API Fuzzer

**Project:** Develop a [Fuzzer](#) for the *Qubes OS Admin API*.

**Brief explanation:** The *Qubes OS Admin API* enables VMs to execute privileged actions on other VMs or dom0 - if allowed by the Qubes OS RPC policy. Programming errors in the Admin API however may cause these access rights to be more permissive than anticipated by the programmer.

Since the Admin API is continuously growing and changing, continuous security assessments are required. A [Fuzzer](#) would help to automate part of these assessments.

**Expected results:** - fully automated & extensible Fuzzer for parts of the Admin API - user & developer documentation

**Difficulty:** medium

**Prerequisites:** - basic Python understanding - some knowledge about fuzzing & existing fuzzing frameworks (e.g. [oss-fuzz](#)) - a hacker's curiosity

**Size of the project:** 175 hours

**Mentor:** Inquire on [qubes-devel](#).

## Secure Boot support

**Project:** Add support for protecting boot binaries with Secure Boot technology, using user-generated keys.

**Brief explanation:** Since recently, Xen supports “unified EFI boot” which allows to sign not only Xen binary itself, but also dom0 kernel and their parameters. While the base technology is there, enabling it is a painful and complex process. The goal of this project is to integrate configuration of this feature into Qubes, automating as much as possible. See discussion in [issue #4371](#)

**Expected results:** - a tool to prepare relevant boot files for unified Xen EFI boot - this includes collecting Xen, dom0 kernel, initramfs, config file, and possibly few more (ucode update?); the tool should then sign the file with user provided key (preferably propose to generate it too) - integrate it with updates mechanism, so new Xen or dom0 kernel will be picked up automatically - include a fallback configuration that can be used for troubleshooting (main unified Xen EFI intentionally does not allow to manipulate parameters at boot time)

**Difficulty:** hard

**Knowledge prerequisite:** - basic understanding of Secure Boot - Bash and Python scripting

**Size of the project:** 175 hours

**Mentor:** [Marek Marczykowski-Górecki](#)

## Reduce logging of Disposable VMs

**Project:** Reduce logging of Disposable VMs

**Brief explanation:** Partial metadata of a DisposableVM is stored in the dom0 filesystem. This applies to various logs, GUI status files etc. There should be an option to hide as much of that as possible - including bypassing some logging, and removing various state files, or at the very least obfuscating any hints what is running inside DisposableVM. More details at [issue #4972](#)

**Expected results:** A DisposableVM should not leave logs hinting what was running inside.

**Difficulty:** medium

**Knowledge prerequisite:** - Python scripting - Basic knowledge of Linux system services management (systemd, syslog etc)

**Size of the project:** 350 hours

**Mentor:** [Marek Marczykowski-Górecki](#)

## Past Projects

You can view the projects we had in 2017 in the [GSoC 2017 archive](#). We also participated in GSoC 2020 and GSoC 2021, and you can see the project in the [GSoC 2020 archive](#) and [GSoC 2021 archive](#).

Here are some successful projects which have been implemented in the past by Google Summer of Code participants.

## Template manager, new template distribution mechanism

**Project:** Template manager, new template distribution mechanism

**Brief explanation:** Template VMs currently are distributed using RPM packages. There are multiple problems with that, mostly related to static nature of RPM package (what files belong to the package). This means such Template VM cannot be renamed, migrated to another storage (like LVM), etc. Also we don't want RPM to automatically update template package itself (which would override all the user changes there). More details: [#2064](#), [#2534](#), [#3573](#).

**Expected results:**

- Design new mechanism for distributing templates (possibly including some package format - either reuse something already existing, or design new one). The mechanism needs to handle:
  - integrity protection (digital signatures), not parsing any data in dom0 prior to signature verification
  - efficient handling of large sparse files
  - ability to deploy the template into various storage mechanisms (sparse files, LVM thin volumes etc).
  - template metadata, templates repository - enable the user to browse available templates (probably should be done in dedicated VM, or DisposableVM)
  - manual template removal by users (without it, see problems such as [#5509](#))
- Implement the above mechanism:
  - tool to download named template - should perform download operation in some VM (as dom0 have no network access), then transfer the data to dom0, verify its integrity and then create Template VM and feed it's root filesystem image with downloaded data.
  - tool to browse templates repository - both CLI and GUI (preferably integrated with existing Template Manager tool)
  - integrate both tools - user should be able to choose some template to be installed from repository browsing tool - see [#1705](#) for some idea (this one lacks integrity verification, but a similar service could be developed with that added)
- If new “package” format is developed, add support for it into [linux-template-builder](#).
- Document the mechanism.
- Write unit tests and integration tests.

**Knowledge prerequisite:**

- Large files (disk images) handling (sparse files, archive formats)
- Bash and Python scripting
- Data integrity handling - digital signatures (gpg2, gpgv2)
- PyGTK
- RPM package format, (yum) repository basics

**Mentor:** [Marek Marczykowski-Górecki](#)

---

We adapted some of the language here about GSoC from the [KDE GSoC page](#).

### 1.13.3 Google Season of Docs (GSoD)

Thank you for your interest in participating in the [2023 Google Season of Docs](#) program with the [Qubes OS team](#). This page details our 2023 project idea as well as completed past projects. You can read more about the Google Season of Docs in the official [guides](#) and [FAQ](#).

## Instructional video series – Qubes OS

### About the Qubes OS Project

Qubes OS is a security-focused operating system that allows you to organize your digital life into compartments called “qubes.” If one qube is compromised, the others remain safe, so a single cyberattack can no longer take down your entire digital life in one fell swoop. You can think of using Qubes OS as having many different computers on your desk for different activities but with the convenience of a single physical machine, a single unified desktop environment, and a set of tools for using qubes together securely as parts of a unified system.

Qubes OS was launched in 2011 and has [received praise from security experts and organizations](#) like Edward Snowden, the Freedom of the Press Foundation, Micah Lee, and Let’s Encrypt. Qubes has [over 40,000 active users](#). From network-level to software-level protections, as well as protections against firmware and hardware attacks, Qubes OS is trying to protect the user from the most significant attacks they encounter so that they can get their work done safely.

### The project’s problem

There is user demand for high-quality, up-to-date video guides that take users from zero Linux knowledge to using Qubes as a daily driver and performing specific tasks inside of Qubes, but almost no such videos exist. Although most of the required knowledge is documented, many users report that they would prefer to watch videos rather than read text or that they would find videos easier to understand and follow along with.

### The project’s scope

This project consists of creating a series of instructional videos that satisfy the following criteria:

- Prospective users who are not yet familiar with Linux or Qubes OS can easily understand and follow the videos.
- The videos make a good effort to catch and keep the attention of their target audience.
- Users can follow the videos step-by-step to install Qubes OS and accomplish various tasks.
- The videos show the actual software being used (i.e., Qubes OS and any relevant software running inside of it).
- The videos are technically accurate, include security warnings where appropriate, and use terminology in a way that is consistent with the rest of the documentation (also see the [glossary](#)).
- The video series is comprehensive enough that users do not need to consult the documentation or ask questions (e.g., on the forum) in order to accomplish the most popular tasks and activities.
- The videos include voice narration. (Showing the speaker is optional.)
- The quality of the videos is consistent with current standards regarding things like editing, transitions, animations, lighting, and audio quality.
- The videos are in high definition (minimum 1080p, preferably 4k).
- The videos are separated into a series, where each video is an appropriate length and is appropriately connected to the other videos in the series.

- The videos are suitable for upload and sharing on popular video-sharing and social-media platforms, such as YouTube and Twitter. (The account or channel under which the videos are uploaded is open to discussion on platforms where the Qubes OS Project does not already have a significant established presence, such as YouTube.)
- The videos are suitable for embedding in appropriate places in the Qubes documentation. (E.g., a video on how to update Qubes OS should be appropriate for appearing on the [how to update](#) page.)
- Where possible, the videos should strive to be version-independent. (For example, a video explaining the template system should still be relevant many releases from now if the template system has not changed.)

Below is an example of the content (which is already *documented*) that the video series is likely to cover. The precise scope of content is to be determined in consultation with the video creator.

- Introduction to Qubes
- Selecting appropriate hardware
- How to install Qubes OS
- First steps after installing
- How to organize your qubes
- How to update
- How to back up, restore, and migrate
- How to copy and paste text (including dom0)
- How to copy and move files (including dom0)
- How to install software
- How to use and customize disposables
- How to enter fullscreen mode
- How to use devices (including block storage, USB, PCI, and optical)
- Templates: understanding, installing, uninstalling, reinstalling, etc.
- Common troubleshooting (preferably included in previous videos at appropriate points)
- The Qubes firewall
- Passwordless root
- Anti Evil Maid
- Split GPG
- U2F proxy
- YubiKey
- Whonix
- How to install and use a VPN in Qubes
- How to install and use Windows in Qubes
- Other popular topics, as time permits

The project is estimated to need around six months to complete (see the timeline below). Qubes team members, including Michael Carbone, Andrew Wong, and Marek Marczykowski-Górecki, will supervise and support the creator.

## Measuring the project's success

We will consider the project successful if, after publication of the video series:

- Actual prospective users with no prior familiarity with Linux or Qubes OS are able to successfully install and use Qubes OS as intended by following along with the videos.
- The reception to the videos is generally positive and complaints about quality and accuracy are minimal.
- Appropriate analytics (e.g., YouTube metrics) are average or better for videos of this type (to be determined in consultation with the creator).

## Timeline

Dates	Action items
March	Orientation
April–October	Create Qubes OS video series
November	Final project evaluation and case study

## Project budget

Expense	Amount
Video creator (20 hours/week, 6 months)	\$12,000
TOTAL	\$12,000

## Additional information

Qubes OS regularly participates in Google Summer of Code and Google Season of Docs. This is our third time participating in Google Season of Docs. Our mentorships for GSoD 2019 and 2020 were successes, and both projects were completed within the times allotted. The past Google Season of Docs projects have given us experience in working with technical writers and have helped us to understand the benefits that technical writers can bring to our project. While our experience in working with video creators is more limited, we are keenly aware of the benefits of high-quality video content, as well as the significant time, resources, and talent required to create it.

## Past Projects

You can view the project we had in 2019 in the [2019 GSoD archive](#) and the [2019 writer's report](#).

You can also view the project we had in 2020 in the [2020 GSoD archive](#) and the [2020 writer's report](#).

Here are some successful projects which have been implemented in the past by Google Season of Docs participants.

### Consolidate troubleshooting guides

**Project:** Consolidate troubleshooting guides

**Brief explanation:** Troubleshooting guides are scattered across many pages and sometimes incomplete, leading to repeatedly posting the same instruction over and over when helping users to diagnose problems. This could be helped by writing a consolidated guide with a clear list of symptom-action layout.

**Expected results:**

- Review existing *troubleshooting guides*
- Review *issues* containing common troubleshooting steps (checking specific logs etc)
- Propose updated, consolidated troubleshooting documentation, including its layout

**Knowledge prerequisite:**

- *Markdown*

**Mentor:** Marek Marczykowski-Górecki

### Improve Getting Started page

**Project:** Improve Getting Started page

**Brief explanation:** The *Getting Started page* is the place a new user would go to understand better how to use Qubes. It is currently has old screenshots not using the default desktop environment and could have much better flow. In addition, this improved page content may end up being served more directly to the user via the *offline documentation* or the firstboot guide.

**Expected results:**

- Review the existing page and website, similar pages for other OSes
- Provide visual mock-ups and proposed text

**Knowledge prerequisite:**

- basic Qubes OS knowledge
- *Markdown*

**Mentor:** Michael Carbone

### Rewrite qrexec documentation

**Project:** Rewrite qrexec documentation

**Brief explanation:** Current qrexec (qubes remote exec) documentation is hard to follow, important information is hidden within a wall of text. Some parts are split into multiple sections, for example version specific to avoid duplication, but it doesn't help reading it. Additionally, protocol documentation describes only few specific use cases, instead of being clear and precise protocol specification. Fixing this last point may require very close cooperation with developers, as the current documentation doesn't multiple corner cases (that's one of the issue with its current shape).

**Expected results:**

- Review existing *qrexec documentation* and an *issue about it*
- Propose updated, consolidated admin documentation (policy writing, adding services)



- Propose consolidated protocol specification, based on the current documentation, and cooperation with developers

**Knowledge prerequisite:**

- [Markdown](#)

**Mentor:** [Marek Marczykowski-Górecki](#)

### 1.13.4 Continuous integration (CI)

This page explains the [continuous integration \(CI\)](#) infrastructure that the Qubes OS Project uses.

#### Website and documentation

The following commands may be useful as a way to interact with our CI infrastructure on website ([qubesos.github.io](#)) and documentation ([qubes-doc](#)) pull requests (PRs). Note that special permissions may be required to use some of these commands. These commands are generally issued by adding a comment to a PR containing only the command.

- **PipelineRetry:** Attempts to run the entire build pipeline over again. This can be useful if CI incorrectly uses a stale branch instead of testing the PR as if it were merged into `master`.
- **PipelineRetryFailed:** Retry just failed CI jobs. Do not update the branch.
- **PipelineRefresh:** Like **PipelineRetry**, except it only fetches the job status from GitLab. It doesn't schedule a new build.
- **TestDeploy:** Deploys a test website, which is a live version of the Qubes website as if this PR had been merged. This can be useful for previewing a PR on a live public website. **Note:** You must wait for the site to finish building before issuing this command, or else it will deploy an empty website. To find the URL of the test website, look for text similar to "This branch was successfully deployed" and a button named something like "View deployment." Note that there are two different testing sites: `wwwttest` is manually updated, whereas `wwwpreview` is managed by the **TestDeploy** command.

### 1.13.5 Usability & UX

Software that is too complicated to use, is often unused. Because we want as many people as possible to benefit from its unique security properties, the usability and user experience of Qubes OS is an utmost priority!

We ask anyone developing for Qubes OS to please read through this guide to better understand the user experience we strive to achieve. We also ask them to review our visual style guide for other design related information.

---

#### Easy To Use

An ideal user experience is friendly, and it beckons a new user to explore the interface. In this process, they can naturally discover how to use the software. Below are some guidelines that will help you design a user interface that accomplishes this goal.

---

**Important:** Interfaces Should Not

---

- Require extensive configuration before a user can *begin* doing things
- Make it possible to break provided features or actions in unrecoverable ways

- Perform actions which compromise security and data
- Overwhelm the user with too much information and cognitive load

Perhaps the most common cause of mistakes is complexity. If there is a configuration setting that will significantly affect the user's experience, choose a safe and smart default then tuck this setting in an `Advanced Settings` panel.

---

### **Important:** Interfaces Should

---

- Make it easy to discover features and available actions
- Provide some understanding of what discovered features do
- Offer the ability to easily undo mistakes
- Choose intelligent defaults for settings

In making software easy to use, it is crucial to be mindful of `cognitive load` which dictates that *“humans are generally able to hold only seven +/- two units of information in short-term memory.”* Making sure your interfaces don't pass this short-term memory limit is perhaps the most important factor in helping a user feel comfortable instead of overwhelmed.

---

## Easy to Understand

There will always be the need to communicate things to users. In these cases, an interface should aim to make this information easy to understand. The following are simple guides to help achieve this - none of these are absolute maxims!

---

### **Important:** Avoid Acronyms

---

Acronyms are compact and make good names for command line tools. They do not make graphical user interfaces more intuitive for non-technical users. Until one learns an acronym's meaning, it is gibberish. Avoid acronyms in your interfaces whenever possible!

- DVM - Disposable Virtual Machine
- GUID - Global Unique Identifier
- PID - Process Identification Number
- NetVM - Networking Virtual Machine

Despite this rule, some acronyms like USB are widely used and understood due to being in common use for over a decade. It is good to use these acronyms when the full words like `Universal Serial Bus` are more likely to confuse users.

---

### **Important:** Use Simple Words

---

Use the minimum amount of words needed to be descriptive, but also informative. Go with common words that are as widely understood. Sometimes, inventing a word such as `Qube` to describe a `virtual machine` makes the life of the user much easier.

- Use `Disposable Qube` instead of `DVM` or `Disposable Virtual Machine`
- Use `interface` instead of `GUI` or `Graphical User Interface`

- Use `application number` instead of PID or Process Identification Number
  - Use `Networking` or `Networking Qube` instead of `NetVM` given context
- 

---

**Important:** Avoid Technical Words

---

Technical words are usually more accurate, but they often *only* make sense to technical users and are confusing and unhelpful to non-technical users. Examples of technical words that might show up in Qubes OS are:

- `root.img`
- `savefile`
- `qrexec-daemon`

These are all terms that have at some point showed up in users' notification messages. Each term is very specific, but requires the user to understand virtualization to interpret.

---

**Important:** Use Common Concepts

---

Large amounts of the global population have been using computers for one or two decades and have formed some mental models of how things work. Leveraging these mental models are a huge gain.

- Use `disk space` instead of `root.img`, since while not quite accurate, it makes contextual sense
- Use `saving` instead of `savefile` as the former is the action trying to be completed
- Use `Qubes` instead of `qrexec-daemon` as it gives better context on what is happening

These words are more abstract and user relevant- they help a user understand what is happening based on already known concepts (disk space) or start to form a mental model of something new (Qubes).

---

**Important:** Avoid Inconsistencies

---

It is easy to start abbreviating (or making acronyms) of long terms like `Disposable Virtual Machine` depending on where the term shows up in an interface.

- `DVM`
- `DispVM`
- `DisposableVM`

This variation in terms can cause new users to question or second guess what the three different variations mean, which can lead to inaction or mistakes.

---

**Important:** Make Things Consistent

---

Always strive to keep things consistent in the interfaces as well as documentation and other materials.

- Use `Disposable Qube` at all times as it meets other criteria as well.

By using the same term throughout an interface, a user can create a mental model and relationship with that term allowing them to feel empowered.

---

---

**Important:** Avoid Duplicate Words

---

It is easy to add words like `Domain` before items in a list or menu in an attempt to be descriptive, such as:

Menu

- Domain: work
- Domain: banking
- Domain: personal

The repeated use of the word `Domain` requires a user to read it for each item in the list, which makes extra work for the eye in parsing out the relevant word like `work`, `banking`, or `personal`. This also affects horizontal space on fixed width lines.

---

---

**Important:** Create Groups & Categories

---

It is more efficient to group things under headings instead as this allows the eye to easily scan the uniqueness of the items. (As per our previous example:)

Domains

- Work
- Banking
- Personal

---

## Easy To Complete

Lastly, expected (and unexpected) situations often require user actions or input. Make resolving these occurrences as easy as possible to complete the action.

---

---

**Important:** Don't Leave Users Stranded

---

Consider the following notifications:

- The disk space of your Qube "Work" is full
- There was an error saving Qube "Personal"

Instead of displaying solvable errors like these and neglecting to provide a fix:

---

---

**Important:** Offer Actionable Solutions

---

Error messages and limits such as those in the previous example can be greatly improved by adding buttons or links to helpful information.

- Add a button to Increase Disk Space
  - Add a link to a documentation page called Troubleshoot saving data
-

In adhering to these principles, you'll make undesirable situations more manageable for users instead of feeling stranded.

---

---

**Important:** Minimize Repetitive Steps

---

There are many cases where a user wants to perform an action on more than one file or folder. However in order to do the action, the user must repeat certain steps such as:

1. Click on **Open File** from a menu or button
2. Navigate through file system
  - Click Folder One
  - Click Folder Two
  - Click Folder Three
  - Click Folder Four
3. Select proper file
4. Complete task on file

That subtle act of clicking through a file system can prove to be significant if a user needs to open more than a couple files in the same directory. We can alleviate some of the work by changing the process:

1. Click on **Open File** from a menu or button
2. Remember last open folder/file system
3. Select proper file
4. Complete task

Clearly, cutting out something as simple as navigating through the file system can save a user quite a bit of time. Alternatively, adding a button or menu item like **Open Multiple Files** might be even better, because remembering and using relevant hotkeys is often something only power users know how to do!

---

## GNOME, KDE, and Xfce

The desktop GUIs that QubesOS versions 1 - 4.1 offer are [KDE](#) and [Xfce](#). We are currently migrating towards using [GNOME](#). We know some people prefer KDE, but we believe Gnome is easier to use for average non-technical users. Xfce will always be supported, and technical users will always have the choice to use KDE or other desktop environments.

This change means you should use [GTK](#) rather than Qt for new GUIs.

All three of these mentioned desktop environments have their own [human interface guidelines](#), and we suggest you familiarize yourself with the platform you developing for.

- [GNOME Human Interface Guidelines](#)
  - [KDE HIG](#)
  - [Xfce UI Guidelines](#)
-

## **Further Learning & Inspiration**

Learning to make well designing intuitive interfaces and software is specialized skillset that can take years to cultivate, but if you are interested in furthering your understanding, we suggest the following resources:

- [Learn Design Principles](#) by Melissa Mandelbaum
- [Usability in Free Software](#) by Jan C. Borchardt
- [Superheroes & Villains in Design](#) by Aral Balkan
- [First Rule of Usability? Don't Listen to Users](#) by Jakob Nielsen
- [10 Usability Heuristics for User Interface Design](#) by Jakob Nielsen
- [Hack Design](#) - online learning program

### **1.13.6 Research**

Here are links to various research papers, projects, videos, and blog posts related to Qubes OS.

#### **Secure Software Development**

- [Security challenges for the Qubes build process](#) by Joanna Rutkowska, May 2016

#### **Towards Trusted Hardware**

- [Thoughts on the “physically secure” ORWL computer](#) by Joanna Rutkowska, September 2016
- [State considered harmful](#) by Joanna Rutkowska, December 2015
- [Intel x86 considered harmful](#) by Joanna Rutkowska, October 2015

#### **Intel's Safeguard Extensions**

- [Thoughts on Intel's upcoming SoftwareGuard Extensions \(Part 2\)](#) by Joanna Rutkowska, September 2013
- [Thoughts on Intel's upcoming SoftwareGuard Extensions \(Part 1\)](#) by Joanna Rutkowska, August 2013

#### **Attacks on Intel TXT**

- [Attacking Intel TXT® via SINIT code execution hijacking](#) by Rafal Wojtczuk and Joanna Rutkowska, November 2011
- [Another Way to Circumvent Intel® Trusted Execution Technology](#) by Rafal Wojtczuk, Joanna Rutkowska, Alex Tereshkin, December 2009
- [ACPI: Design Principles and Concerns](#) by Loic Duflot, Olivier Levillain, and Benjamin Morin, 2009
- [Attacking Intel® Trusted Execution Technology](#) by Rafal Wojtczuk and Joanna Rutkowska

## Software Attacks Coming Through Devices

- [Following the White Rabbit: Software Attacks against Intel® VT-d](#) by Rafal Wojtczuk and Joanna Rutkowska, April 2011
- [On Formally Verified Microkernels \(and on attacking them\)](#) by Joanna Rutkowska, May 2010
- [Remotely Attacking Network Cards \(or why we do need VT-d and TXT\)](#) by Joanna Rutkowska, April 2010
- [Can you still trust your network card?](#) by Loïc Duflot, Yves-Alexis Perez and others

## Application-level Security

- [Virtics: A System for Privilege Separation of Legacy Desktop Applications](#) by Matt Piotrowski, May 2010

## Compartmentalization, Isolation, and Separation

- [Software compartmentalization vs. physical separation](#) by Joanna Rutkowska, August 2014
- [Breaking Up is Hard to Do: Security and Functionality in a Commodity Hypervisor](#) by Patrick Colp et al., October 2011

## The Qubes Architecture

- [Qubes virtual mini-summit 2021](#) by 3mdeb and the Qubes team, August 2021
- [Qubes Air: Generalizing the Qubes Architecture](#) by Joanna Rutkowska, January 2018
- [Introducing the Next Generation Qubes Core Stack](#) by Joanna Rutkowska, October 2017
- [Introducing the Qubes Admin API](#) by Joanna Rutkowska, June 2017
- [Qubes OS Architecture Spec v0.3](#) by Joanna Rutkowska and Rafal Wojtczuk, January 2010

### 1.13.7 Developer books

Below is a list of various books that might be useful in learning some basics needed for Qubes development.

- A must-read about Xen internals:
  - [The Definitive Guide to the Xen Hypervisor](#), by David Chisnall
- Some good books about the Linux kernel:
  - [Linux Kernel Development](#), by Robert Love
  - [Linux Device Drivers](#), by Jonathan Corbet
- Solid intro into Trusted Computing:
  - [Dynamics of a Trusted Platform](#), by David Grawrock (original Intel architect for TXT)
- Good book about GIT:
  - [Pro Git](#), by Scott Chacon and Ben Straub (complete book available free online)
- Useful books about Python:
  - [Programming in Python 3](#), by Mark Summerfield

- [Rapid GUI Programming with Python and Qt](#), by Mark Summerfield (Although note that *Qt is being replaced by GTK* in Qubes code.)

### 1.13.8 Source code

All the Qubes code is kept in Git repositories. We have divided the project into several components, each of which has its own separate repository, for example:

- `core-admin.git` – The core Qubes infrastructure, responsible for VM management, VM templates, fs sharing, etc.
- `gui-daemon.git` – GUI virtualization, Dom0 side.
- `gui-agent-linux.git` – GUI virtualization, Linux VM side.
- `linux-template-builder.git` – Scripts and other files used to create Qubes template images.

All of our repositories are available under the [QubesOS GitHub account](#).

To clone a repository:

```
git clone https://github.com/QubesOS/qubes-<repo_name>.git <repo_name>
```

e.g.:

```
git clone https://github.com/QubesOS/qubes-core-admin.git core-admin
```

To build Qubes you do not need to download all these repositories. If you use *qubes builder* you can specify *what* you want to build, and download only the repositories needed to build that target.

If you really do want to clone **all** of the repositories, you can use these commands:

```
curl "https://api.github.com/orgs/QubesOS/repos?page=1&per_page=100" | grep -e 'clone_
↳url*' | cut -d \" -f 4 | xargs -L1 git clone
curl "https://api.github.com/orgs/QubesOS/repos?page=2&per_page=100" | grep -e 'clone_
↳url*' | cut -d \" -f 4 | xargs -L1 git clone
```

To update (git fetch) **all** of these repositories :

```
find . -mindepth 1 -maxdepth 1 -type d -exec git -C {} fetch --tags --recurse-
↳submodules=on-demand --all \;
```

(Alternatively, you can pull instead of just fetching.)

### How to Send Patches

If you want to *contribute code* to the project, there are two ways. Whichever method you choose, you must *sign your code* before it can be accepted.

- **Preferred:** Use GitHub's [fork & pull requests](#). Opening a pull request on GitHub greatly eases the code review and tracking process. In addition, especially for bigger changes, it's a good idea to send a message to the [qubes-devel mailing list](#) in order to notify people who do not receive GitHub notifications.
- Send a patch to the [qubes-devel mailing list](#) (`git format-patch`).
  1. Make all the changes in your working directory, i.e. edit files, move them around (you can use 'git mv' for this), etc.



2. Add the changes and commit them (`git add`, `git commit`). Never mix different changes into one commit! Write a good description of the commit. The first line should contain a short summary, and then, if you feel like more explanations are needed, enter an empty new line, and then start the long, detailed description (optional).
3. Test your changes NOW: check if RPMs build fine, etc.
4. Create the patch using `git format-patch`. This has an advantage over `git diff`, because the former will also include your commit message, your name and email, so that *your* name will be used as a commit's author.
5. Send your patch to `qubes-devel`. Start the message subject with `[PATCH]`.

### 1.13.9 Software license

Qubes OS is a compilation of software packages, each under its own license. The compilation is made available under the GNU General Public License version 2 (GPLv2). However, the license for this compilation does not supersede the license of any package included in the compilation.

The source code of Qubes OS is contained in repositories under the [@QubesOS](#) account on GitHub. This source code is made available under GPLv2, unless there is a `LICENSE` file in the root of the containing repository that specifies a different license.

The full text of the GPLv2 license can be found [here](#).

### 1.13.10 Coding style

#### Rationale

Maintaining proper coding style is very important for any large software project, such as Qubes. Here's why:

- It eases maintenance tasks, such as adding new functionality or generalizing code later,
- It allows others (as well as the future you!) to easily understand fragments of code and what they were supposed to do, and thus makes it easier to later extend them with newer functionality or bug fixes,
- It allows others to easily review the code and catch various bugs,
- It provides for an aesthetically pleasing experience when one reads the code...

Often, developers, usually smart ones, undersell the value of proper coding style, thinking that it's much more important how their code works. These developers expect that if their code solves some problem using a nice and neat trick, then that's all that is really required. Such thinking shows, however, immaturity and is a signal that the developer, no matter how bright and smart, might not be a good fit for larger projects. Writing a clever exploit for a Black Hat show is one thing - writing useful software supposed to be used and maintained for years is quite a different story. If you want to show off what a smart programmer you are, then you should become a researcher and write exploits. If, on the other hand, you want to be part of a team that makes real, useful software, you should ensure your coding style is impeccable. At Qubes project, we often took shortcuts and wrote nasty code, and this has always back fired at us, sometime months, sometime years later, the net result being we had to spend time fixing code, rather than implementing new functionality.

And here's a [link to the real case](#) (one Qubes Security Bulletin) demonstrating how the lackadaisical coding style lead to a real security bug. Never assume you're smart enough to disregard clean and rigorous coding!

## General typographic conventions

- **Use space-expanded tabs that equal 4 spaces.** Yes, we know, there are many arguments for using “real” tabs instead of space-expanded tabs, but we need to pick one convention to make the project consistent. One argument for using space-expanded tabs is that this way the programmer is in control of how the code will look like, despite how other users have configured their editors to visualize the tabs (of course, we assume any sane person uses a fixed-width font for viewing the source code). If it makes you feel any better, assume this is just an arbitrary choice made to enforce a unified style.
- **Maintain max. line length of 80 characters.** Even though today’s monitors often are very wide and it’s often not a problem to have 120 characters displayed in an editor, maintaining shorter line lengths improves readability. It also allows others to have two parallel windows open, side by side, each with different parts of the source code.
- **Naming conventions for any OS other than Windows:**
  - `ClassName`
  - `some_variable`, `some_function`, `some_argument`
- **Naming convention for Windows OS** – exceptionally to preserve Windows conventions please use the following:
  - `ClassName`, `FunctionName`
  - `pszArgumentOne`, `hPipe` – use Hungarian notation for argument and variables
- **Maintain a decent amount of horizontal spacing.** e.g. add a space after `if` or before `{` in C, and similar in other languages. Whether and where to also use spaces within expressions, such as  $(x^2+5)$  vs.  $(x * 2 + 5)$  is left to the developer’s judgment. Do not put spaces immediately after or before the brackets in expressions, so avoid constructs like this: `if ( condition )` and use ones like this: `if (condition)` instead.
- **Use single new lines** (‘n’ aka LF) in any non-Windows source code. On Windows, exceptionally, use the CRLF line endings (–). This will allow the source code to be easily viewable in various Windows-based programs.
- **Use descriptive names for variables and functions!** Really, at a time when most editors have auto-completion features, there is no excuse for using short variable names.
- **Comments should be indented together with the code**, e.g. like this:

```
for (...) {  
    // The following code finds PGP private key matching the given public key in_  
    ↪ 0(1)  
    while (key_found) {  
        (...)  
    }  
}
```

## File naming conventions

- All file names written with small letters, use dash to separate words, rather than underscores, e.g. `qubes-dom0-update`. Never use spaces!

### File naming in Linux/Unix-like systems:

- User commands that operate on particular VMs (also those accessible in VMs): `/usr/bin/qvm-*`
- User commands that apply to the whole system (Dom0 only): `/usr/bin/qubes-*`
- Auxiliary executables and scripts in `/usr/libexec/qubes/` (Note: previously we used `/usr/lib/qubes` but decided to change that)
- Helper, non-executable files in `/usr/share/qubes/`

- Various config files in `/etc/qubes`
- Qubes RPC services in `/etc/qubes-rpc`. Qubes RPC Policy definitions (only in Dom0) in `/etc/qubes-rpc/policy/`
- All VM-related configs, images, and other files in `/var/lib/qubes/`
- System-wide temporary files which reflect the current state of system in `/var/run/qubes`
- Logs: either log to the system-wide messages, or to `/var/log/qubes/`

#### File naming in Windows systems:

- All base qubes-related files in `C:\Program Files\Invisible Things Lab\Qubes\` (Exceptionally spaces are allowed here to adhere to Windows naming conventions)
- Other, third-party files, not Qubes-specific, such as e.g. Xen PV drivers might be in different vendor subdirs, e.g. `C:\Program Files\Xen PV Drivers`

### General programming style guidelines

- Do not try to impress with your coding kung-fu, do not use tricks to save 2 lines of code, always prefer readability over trickiness!
- Make sure your code compiles and builds without warnings.
- Always think first about interfaces (e.g. function arguments, or class methods) and data structures before you start writing the actual code.
- Use comments to explain non-trivial code fragments, or expected behavior of more complex functions, if it is not clear from their name.
- Do **not** use comments for code fragments where it is immediately clear what the code does. E.g. avoid constructs like this:

```
// Return window id
int get_window_id (...) {
    (...)
    return id;
}
```

- Do **not** use comments to disable code fragments. In production code there should really be no commented or disabled code fragments. If you really, really have a good reason to retain some fragment of unused code, use `#if` or `#ifdef` to disable it, e.g.:

```
#if 0
    (...)    // Some unused code here
#endif
```

... and preferably use some descriptive macro instead of just `0`, e.g.:

```
#if USE_OLD_WINDOW_TRAVERSING
    (...)    // Some unused code here
#endif
```

Not sure how to do similar thing in Python... Anyone?

But generally, there is little excuse to keep old, unused code fragments in the code. One should really use the functionality provided by the source code management system, such as git, instead. E.g. create

a special branch for storing the old, unused code – this way you will always be able to merge this code into upstream in the future.

- Do not hardcode values in the code! The only three numbers that are an exception here and for which it is acceptable to hardcode them are: 0, 1 and -1, and frankly the last two are still controversial...

### Source Code management (Git) guidelines

- Use git to maintain all code for Qubes project.
- Before you start using git, make sure you understand that git is a decentralized Source Code Management system, and that it doesn't behave like traditional, centralized source code management systems, such as SVN. Here's a good [introductory book on git](#). Read it.
- Qubes code is divided into many git repositories. There are several reasons for that:
  - This creates natural boundaries between different code blocks, enforcing proper interfaces, and easing independent development to be conducted on various code parts at the same time, without the fear of running into conflicts.
  - By maintaining relatively small git repositories, it is easy for new developers to understand the code and contribute new patches, without the need to understand all the other code.
  - Code repositories represent also licensing boundaries. So, e.g. because `core-agent-linux` and `core-agent-windows` are maintained in two different repositories, it is possible to have the latter under a proprietary, non-GPL license, while keeping the former fully open source.
  - We have drastically changed the layout and naming of the code repositories shortly after Qubes OS R2 Beta 2 release. For details on the current code layout, please read [this article](#).

### Commit message guidelines

Please attempt to follow these conventions when writing your Git commit messages:

- Separate the subject line from the body with a blank line.
- Limit the subject line to approximately 50 characters.
- Capitalize the subject line.
- Do not end the subject line with a period.
- Use the imperative mood in the subject line.
- Wrap the body at 72 characters.
- Use the body to explain *what* and *why* rather than *how*.

For details, examples, and the rationale behind each of these conventions, please see [this blog post](#), which is the source of this list.

## Security coding guidelines

- As a general rule: **untrusted input** is our #1 enemy!
- Any input that comes from untrusted, or less trusted, or just differently-trusted, entity should always be considered as malicious and should always be sanitized and verified. So, if your software runs in Dom0 and processes some input from any of the VMs, this input should be considered to be malicious. Even if your software runs in a VM, and processes input from some other VM, you should also assume that the input is malicious and verify it.
- Use `untrusted_` prefix for all variables that hold values read from untrusted party and which have not yet been verified to be decent, e.g.:

```
read_struct(untrusted_conf);
/* sanitize start */
if (untrusted_conf.width > MAX_WINDOW_WIDTH)
    untrusted_conf.width = MAX_WINDOW_WIDTH;
if (untrusted_conf.height > MAX_WINDOW_HEIGHT)
    untrusted_conf.height = MAX_WINDOW_HEIGHT;
width = untrusted_conf.width;
height = untrusted_conf.height;
```

- Use others variables, without the `untrusted_` prefix to hold the sanitized values, as shown above.

## Python-specific guidelines

- Please follow the guidelines [here](#), unless they were in conflict with what is written on this page.

## C and C++ specific guidelines

- Do not place code in `*.h` files.
- Use `const` whenever possible, e.g. in function arguments passed via pointers.
- Do not mix procedural and objective code together – if you write in C++, use classes and objects.
- Think about classes hierarchy, before starting to implement specific methods.

## Bash-specific guidelines

- Avoid writing scripts in bash whenever possible. Use python instead. Bash-scripts are Unix-specific and will not work under Windows VMs, or in Windows admin domain, or Windows gui domain.

### 1.13.11 Code signing

All contributions to the Qubes OS *source code* must be cryptographically signed by the author's PGP key.

## Generating a Key

(Note: If you already have a PGP key, you may skip this step.)

Alex Cabal has written an excellent [guide](#) on creating a PGP keypair. Below, we reproduce just the minimum steps in generating a keypair using GnuPG. Please read Cabal's full guide for further important details.

```
$ gpg --gen-key
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: directory '/home/user/.gnupg' created
gpg: keybox '/home/user/.gnupg/pubring.kbx' created
Note: Use "gpg --full-generate-key" for a full featured key generation dialog.

GnuPG needs to construct a user ID to identify your key.

Real name: Bilbo Baggins
Email address: bilbo@shire.org
You selected this USER-ID:
    "Bilbo Baggins <bilbo@shire.org>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit? 0
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

<type your passphrase>

We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: /home/user/.gnupg/trustdb.gpg: trustdb created
gpg: key 6E2F4E7AF50A5827 marked as ultimately trusted
gpg: directory '/home/user/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/user/.gnupg/openpgp-revocs.d/
↳87975838063F97A968D503266E2F4E7AF50A5827.rev'
public and secret key created and signed.

pub   rsa3072 2021-12-30 [SC] [expires: 2023-12-30]
       87975838063F97A968D503266E2F4E7AF50A5827
uid           Bilbo Baggins <bilbo@shire.org>
sub   rsa3072 2021-12-30 [E] [expires: 2023-12-30]
```

## Upload the Key

For others to find the public key, please upload it to a server.

Currently, [these](#) are the recognized servers.

In the example below, we will use `keyserver.ubuntu.com`.

Replace `6E2F4E7AF50A5827` with your key ID, which is the last 8 hex digits of the long number in the second line of the output above:

```
pub  rsa3072 2021-12-30 [SC] [expires: 2023-12-30]
      87975838063F97A968D503266E2F4E7AF50A5827
```

```
$ gpg --send-keys --keyserver hkps://keyserver.ubuntu.com 6E2F4E7AF50A5827
gpg: sending key 6E2F4E7AF50A5827 to hkps://keyserver.ubuntu.com
```

```
$ gpg --send-keys --keyserver hkps://keyserver.ubuntu.com 6E2F4E7AF50A5827
gpg: sending key 6E2F4E7AF50A5827 to hkps://keyserver.ubuntu.com
```

## Using PGP with Git

If you're submitting a patch via GitHub (or a similar Git server), please sign your Git commits.

1. Set up Git to use your key:

```
git config --global user.signingkey <KEYID>
```

2. Set up Git to sign your commits with your key:

```
git config --global commit.gpgsign true
```

Alternatively, manually specify when a commit is to be signed:

```
git commit -S
```

3. (Optional) Create signed tags. Signed commits are totally sufficient to contribute to Qubes OS. However, if you have commits which are not signed and you do not want to change them, you can create a signed tag for the commit and push it before the check. This is useful for example, if you have a commit back in the git history which you like to sign now without rewriting the history.

```
git tag -s <tag_name> -m "<tag_message>"
```

You can also create an alias to make this easier. Edit your `~/.gitconfig` file. In the `[alias]` section, add `stag` to create signed tags and `spush` to create signed tags and push them.

```
[alias]
stag = "!bash -c 'id=\"`git rev-parse --verify HEAD`\"; tag_name=\"signed_tag_for_${id:0:8}\"; git tag -s \"$tag_name\" -m \"Tag for commit $id\"; echo \"$tag_name\"'"
spush = "!bash -c 'git push origin `git stag`'"
```

You may also find it convenient to have an alias for verifying the tag on the latest commit:

```
vtag = !git tag -v `git describe`
```

## GitHub Signature Verification (optional)

GitHub shows a green **Verified** label indicating that the GPG signature could be verified using any of the contributor's GPG keys uploaded to GitHub. You can upload your public key on GitHub by adding your public GPG key on the [New GPG key](#) under the [SSH GPG keys](#) page.

## Code Signature Checks

The [signature-checker](#) checks if code contributions are signed. Although GitHub adds a little green **Verified** button next to the commit, the [signature-checker](#) uses this algorithm to check if a commit is correctly signed:

1. Is the commit signed? If the commit is not signed, you can see the message > policy/qubesos/code-signing — No signature found
2. If the commit is signed, the key is downloaded from a GPG key server. If you can see the following error message, please check if you have uploaded the key to a key server. > policy/qubesos/code-signing — Unable to verify (no valid key found)

## No Signature Found

policy/qubesos/code-signing — No signature found

In this case, you have several options to sign the commit:

1. Amend the commit and replace it with a signed commit. You can use this command to create a new signed commit:

```
git commit --amend -S
```

This also rewrites the commit so you need to push it forcefully:

```
git push -f
```

2. Create a signed tag for the unsigned commit. If the commit is back in history and you do not want to change it, you can create a signed tag for this commit and push the signature. You can use the alias from above:

```
git checkout <commit>
git spush
```

Now, the signature checker needs to re-check the signature. Please comment on the pull request that you would like to have the signatures checked again.

## Unable To Verify

policy/qubesos/code-signing — Unable to verify (no valid key found)

This means that the [signature-checker](#) has found a signature for the commit but is not able to verify it using the any key available. This might be that you forgot to upload the key to a key server. Please upload it.

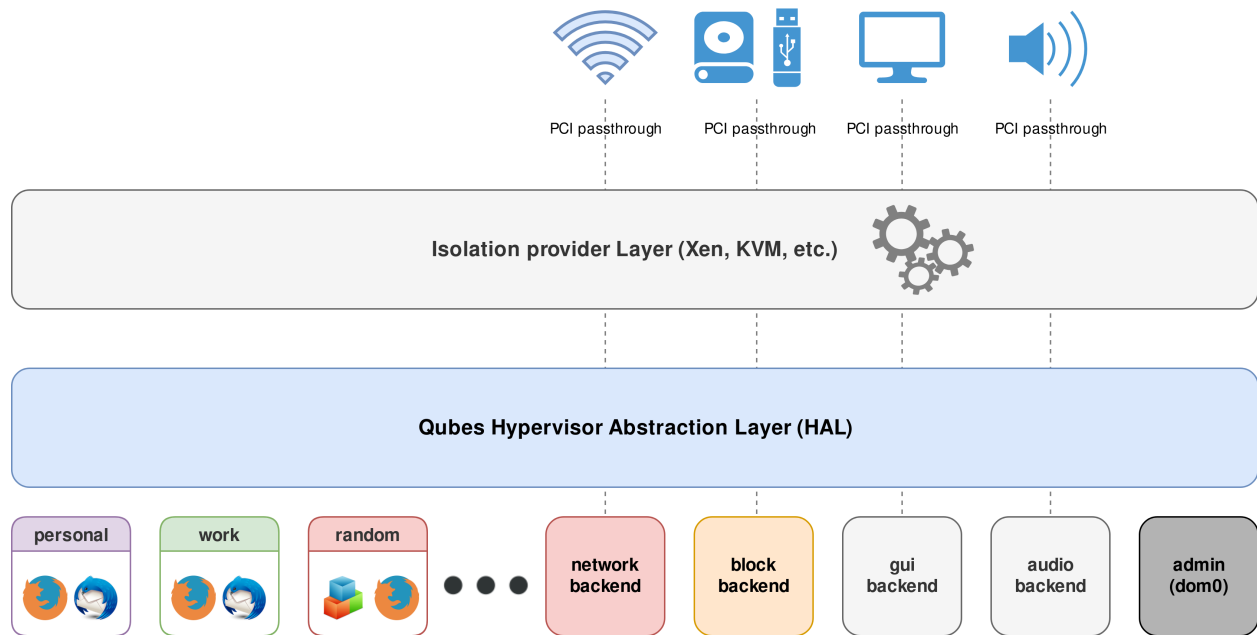


## Using PGP with Email

If you're submitting a patch by emailing the [developer mailing list](#), simply sign your email with your PGP key. One good way to do this is with a program like [Enigmail](#). Enigmail is a security addon for the Mozilla Thunderbird email client that allows you to easily digitally encrypt and sign your emails.

### 1.13.12 Architecture

Qubes implements a security-by-compartmentalization approach. To do this, Qubes utilizes virtualization technology in order to isolate various programs from each other and even to sandbox many system-level components, such as networking and storage subsystems, so that the compromise of any of these programs or components does not affect the integrity of the rest of the system.



Qubes lets the user define many secure compartments known as *qubes*, which are implemented as lightweight *virtual machines (VMs)*. For example, the user can have “personal,” “work,” “shopping,” “bank,” and “random” app qubes and can use the applications within those qubes just as if they were executing on the local machine. At the same time, however, these applications are well isolated from each other. Qubes also supports secure copy-and-paste and file sharing between qubes, of course.

### Key architecture features

- Based on a secure bare-metal hypervisor (Xen)
- Networking code sand-boxed in an unprivileged VM (using IOMMU/VT-d)
- USB stacks and drivers sand-boxed in an unprivileged VM (currently experimental feature)
- No networking code in the privileged domain (dom0)
- All user applications run in “app qubes,” lightweight VMs based on Linux
- Centralized updates of all app qubes based on the same template
- Qubes GUI virtualization presents applications as if they were running locally
- Qubes GUI provides isolation between apps sharing the same desktop

- Secure system boot based (optional)

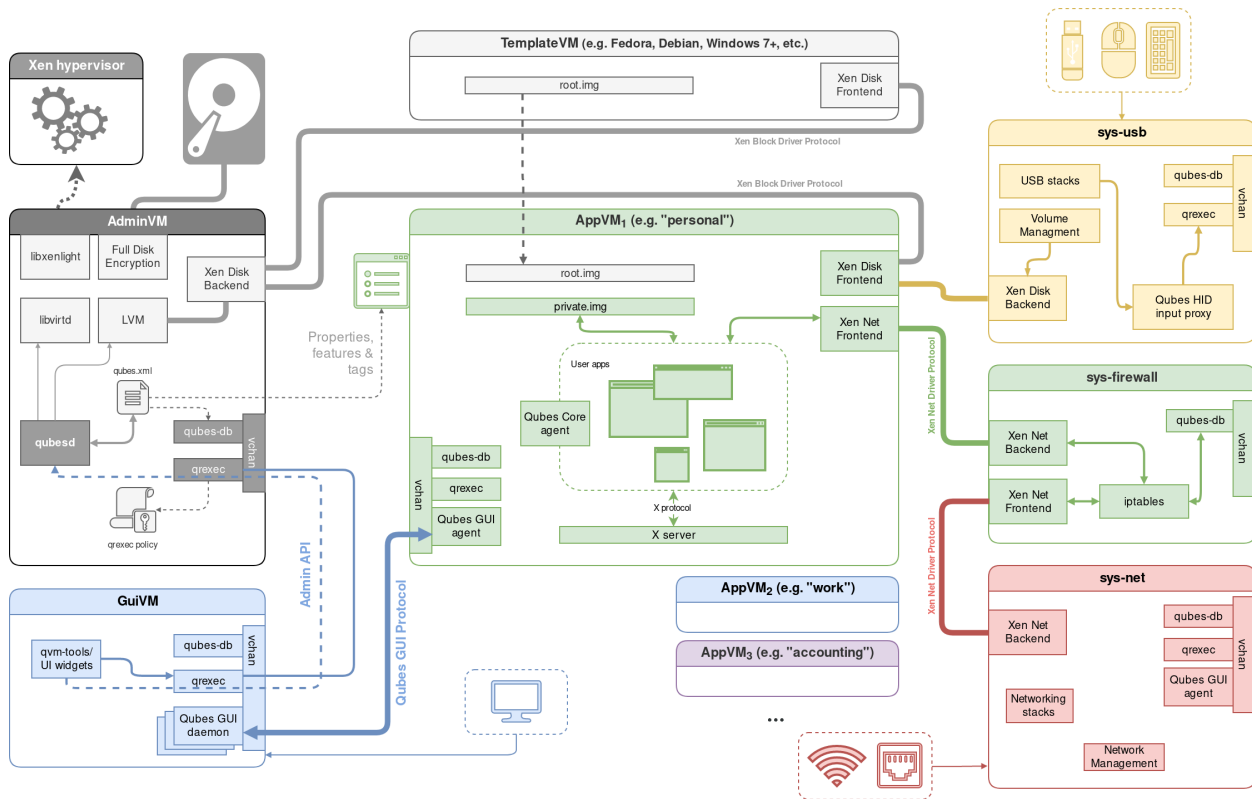
(For those interested in the history of the project, [Architecture Spec v0.3 \[PDF\]](#) is the original 2009 document that started this all. Please note that this document is for historical interest only. For the latest information, please see the rest of the [System Documentation](#).)

### Qubes Core Stack

Qubes Core Stack is, as the name implies, the core component of Qubes OS. It's the glue that connects all the other components together, and which allows users and admins to interact with and configure the system. The other components of the Qubes system include:

- VM-located core agents (implementing e.g. qrexec endpoints used by various Qubes services)
- VM-customizations (making the VMs lightweight and working well with seamless GUI virtualization)
- Qubes GUI virtualization (the protocol, VM-located agents, and daemons located in the GUI domain which, for now, happens to be the same as dom0),
- GUI domain customizations (Desktop Environment customizations, decoration coloring plugin, etc)
- The admin qube distribution (various customizations, special services, such as for receiving and verifying updates, in the future: custom distro)
- The Xen hypervisor (with a bunch of customization patches, occasional hardening) or - in the future - some other virtualising or containerizing software or technology
- Multiple “Qubes Apps” (various services built on top of Qubes qrexec infrastructure, such as: trusted PDF and Image converters, Split GPG, safe USB proxies for HID devices, USB proxy for offering USB devices (exposed via qvm-usb), Yubikey support, USB Armory support, etc)
- Various ready-to-use templates (e.g. Debian-, Whonix-based), which are used to create actual VMs, i.e. provide the root filesystem to the VMs
- Salt Stack integration

And all these components are “glued together” by the Qubes Core Stack.



This diagram illustrates the location of all these components in the overall system architecture. Unlike the other Qubes architecture diagram above, this one takes an app-qube-centric approach.

### 1.13.13 Security design goals

Qubes OS implements a security-by-isolation (or security-by-compartmentalization) approach by providing the ability to easily create many security domains. These domains are implemented as lightweight Virtual Machines (VMs) running under the Xen hypervisor. Qubes' main objective is to provide strong isolation between these domains, so that even if an attacker compromises one of the domains, the others are still safe. Qubes, however, does not attempt to provide any security isolation for applications running within the same domain. For example, a buggy web browser running in a Qubes domain could still be compromised just as easily as on a regular Linux distribution. The difference that Qubes makes is that now the attacker doesn't have access to all the software running in the other domains.

Qubes also provides features that make it easy and convenient to run these multiple domains, such as seamless GUI integration into one common desktop, secure clipboard copy and paste between domains, secure file transfer between domains, disposable VMs, and much more. Qubes also provides an advanced networking infrastructure that allows for the creation of multiple network VMs which isolate all the world-facing networking stacks and proxy VMs which can be used for advanced VPN configurations and tunneling over untrusted connections.

### 1.13.14 Security-critical code

Below is a list of security-critical (i.e., trusted) code components in Qubes OS. A successful attack against any of these components could compromise the system's security. This code can be thought of as the Trusted Computing Base (TCB) of Qubes OS. One of the main goals of the project is to keep the TCB to an absolute minimum. The size of the current TCB is on the order of hundreds of thousands of lines of C code, which is several orders of magnitude less than other OSes. (In Windows, Linux, and Mac OSes, the amount of trusted code is typically on the order of tens of *millions* of lines of C code.)

For more information, see [Qubes Security Goals](#).

#### Security-critical Qubes-specific Components

The following code components are security-critical in Qubes OS:

- Dom0-side of the libvchan library
- Dom0-side of the GUI virtualization code (`qubes-guid`)
- Dom0-side of the sound virtualization code (`pacat-simple-vchan`)
- Dom0-side in `qrexec`-related code (`qrexec_daemon`)
- VM memory manager (`qmemman`) that runs in Dom0
- Select Qubes RPC servers that run in Dom0: `qubes.ReceiveUpdates` and `qubes.SyncAppMenus`
- The `qubes.Filecopy` RPC server that runs in a VM (critical because it could allow one VM to compromise another if the user allows a file copy operation to be performed between them)

#### Security-critical Third-party Components

We did not create these components, but Qubes OS relies on them. At the current stage of the project, we cannot afford to spend the time to thoroughly review and audit them, so we more or less “blindly” trust that they are secure.

- The Xen hypervisor
- Xen's xenstore backend running in Dom0
- Xen's block backend running in Dom0's kernel
- The RPM program used in Dom0 for verifying signatures on dom0 updates
- Somewhat trusted: log viewing software in dom0 that parses VM-influenced logs

#### Attacks on Networking Components

Here are two examples of networking components that an adversary might seek to attack (or in which to exploit a vulnerability as part of an attack):

- Xen network PV frontends
- VMs' core networking stacks (core TCP/IP code)

Hypothetically, an adversary could compromise a NetVM, `sys-net-1`, and try to use it to attack the VMs connected to that NetVM. However, Qubes allows for the existence of more than one NetVM, so the adversary would not be able to use `sys-net-1` in order to attack VMs connected to a *different* NetVM, `sys-net-2` without also compromising `sys-net-2`. In addition, the adversary would not be able to use `sys-net-1` (or, for that matter, `sys-net-2`) to attack VMs that have networking disabled (i.e., VMs that are not connected to any NetVM).

## Buggy Code vs. Backdoored Code

There is an important distinction between buggy code and maliciously backdoored code. We could have the most secure architecture and the most bulletproof TCB that perfectly isolates all domains from each other, but it would all be pretty useless if all the code we ran inside our domains (e.g. the code in our email clients, word processors, and web browsers) were backdoored. In that case, only network-isolated domains would be somewhat trustworthy.

This means that we must trust at least some of the vendors that supply the code we run inside our domains. (We don't have to trust *all* of them, but we at least have to trust the few that provide the apps we use in the most critical domains.) In practice, we trust the software provided by the [Fedora Project](#). This software is signed by Fedora distribution keys, so it is also critical that the tools used in domains for software updates (`dnf` and `rpm`) are trustworthy.

### 1.13.15 GUI virtualization

#### `qubes-gui` and `qubes-guid` processes

All AppVM X applications connect to local (running in AppVM) Xorg servers that use the following “hardware” drivers:

- `dummyqsb_drv` - video driver, that paints onto a framebuffer located in RAM, not connected to real hardware
- `qubes_drv` - it provides a virtual keyboard and mouse (in fact, more, see below)

For each AppVM, there is a pair of `qubes-gui` (running in AppVM) and `qubes-guid` (running in the AppVM's GuiVM, dom0 by default) processes connected over vchan. The main responsibilities of `qubes-gui` are:

- call `XCompositeRedirectSubwindows` on the root window, so that each window has its own composition buffer
- instruct the local Xorg server to notify it about window creation, configuration and damage events; pass information on these events to dom0
- receive information about keyboard and mouse events from dom0, tell `qubes_drv` to fake appropriate events
- receive information about window size/position change, apply them to the local window

The main responsibilities of `qubes-guid` are:

- create a window in dom0 whenever an information on window creation in AppVM is received from `qubes-gui`
- whenever the local window receives `XEvent`, pass information on it to AppVM (particularly, mouse and keyboard data)
- whenever AppVM signals damage event, tell local Xorg server to repaint a given window fragment
- receive information about window size/position change, apply them to the local window

Note that keyboard and mouse events are passed to AppVM only if a window belonging to this AppVM has focus. AppVM has no way to get information on keystrokes fed to other AppVMs (e.g. XTEST extension will report the status of local AppVM keyboard only) or synthesize and pass events to other AppVMs.

## Window content updates implementation

Typical remote desktop applications, like VNC, pass information on all changed window content in-band (say, over tcp). As that channel has limited throughput, this impacts video performance. In the case of Qubes, `qubes-gui` does not transfer all changed pixels via `vchan`. Instead, for each window, upon its creation or size change:

- Old `qubes-gui` versions will ask `qubes-driv` driver for the list of physical memory frames that hold the composition buffer of a window, and pass this to `dom0` via the deprecated `MFNDUMP` message.
- New `qubes-gui` versions will rely on `qubes-driv` having allocated memory using `gntalloc`, and then pass the grant table indexes `gntalloc` has chosen to the GUI daemon using the `WINDOW_DUMP` message.

Now, `qubes-guid` has to tell the `dom0` Xorg server about the location of the buffer. There is no supported way (e.g. Xorg extension) to do this zero-copy style. The following method is used in Qubes:

- in `dom0`, the Xorg server is started with `LD_PRELOAD`-ed library named `shmoverride.so`. This library hooks all function calls related to shared memory.
- `qubes-guid` creates a shared memory segment, and then tells Xorg to attach it via MIT-SHM extension
- when Xorg tries to attach the segment (via `glibc shmat`) `shmoverride.so` intercepts this call and instead maps AppVM memory via `xc_map_foreign_pages` for the deprecated `MFNDUMP` message, or `xengnttab_map_domain_grant_refs` for the `WINDOW_DUMP` message.
- afterwards, we can use MIT-SHM functions, such as `XShmPutImage`, to draw onto a `dom0` window. `XShmPutImage` will paint with DRAM speed, and many drivers use DMA to make this even faster.

The important detail is that `xc_map_foreign_pages` verifies that a given mfn range actually belongs to a given domain id (and the latter is provided by trusted `qubes-guid`). Therefore, rogue AppVM cannot gain anything by passing crafted mnfs in the `MFNDUMP` message. Similarly, `xengnttab_map_domain_grant_refs` will only map grants from the specific domain ID specified by `qubes-guid`, so crafted `WINDOW_DUMP` messages are not helpful to an attacker.

To sum up, this solution has the following benefits:

- window updates at DRAM speed
- no changes to Xorg code
- minimal size of the supporting code

There are two reasons that `WINDOW_DUMP` is preferred over `MFNDUMP`:

1. `xc_map_foreign_pages` can only be used by `dom0`, as it allows accessing all memory of any VM. Allowing any VM other than `dom0` to do this would be a security vulnerability.
2. `xc_map_foreign_pages` requires the guest physical address of the pages to map, but normal userspace processes (such as `qubes-gui` or Xorg) do not have access to that information. Therefore, the translation is done via the `u2mfn` out-of-tree kernel module.

Currently, using `WINDOW_DUMP` does come at a performance cost, because the AppVM's X server must copy the pages from the application to the `gntalloc`-allocated memory. This will be solved by future improvements to `gntalloc`, which will allow exporting *any* page via `gntalloc`, including memory shared by another process.

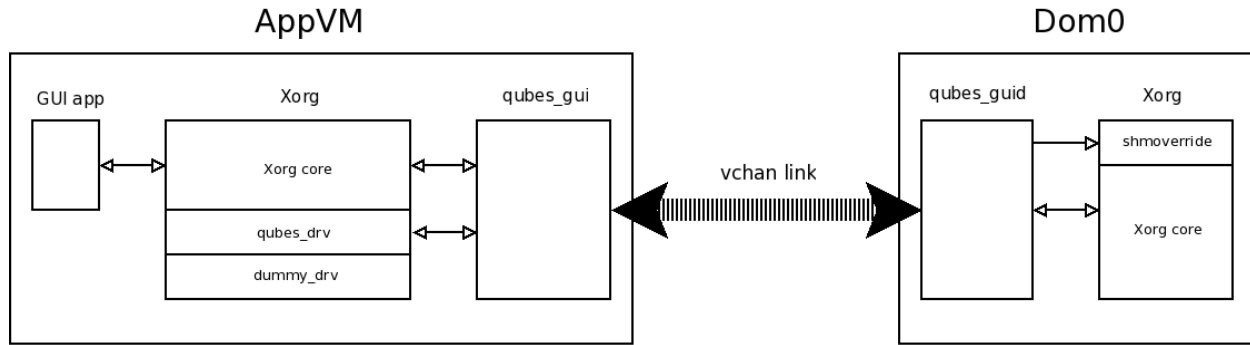


Fig. 13: gui.png

### Security markers on dom0 windows

It is important that the user knows which AppVM a given window belongs to. This prevents a rogue AppVM from painting a window pretending to belong to other AppVM or dom0 and trying to steal, for example, passwords.

In Qubes, a custom window decorator is used that paints a colourful frame (the colour is determined during AppVM creation) around decorated windows. Additionally, the window title always starts with **[name of the AppVM]**. If a window has an `override_redirect` attribute, meaning that it should not be treated by a window manager (typical case is menu windows), `qubes-gui` draws a two-pixel colourful frame inside it manually.

### Clipboard sharing implementation

Certainly, it would be insecure to allow AppVM to read/write the clipboards of other AppVMs unconditionally. Therefore, the following mechanism is used:

- there is a “qubes clipboard” in dom0 - its contents are stored in a regular file in dom0.
- if the user wants to copy local AppVM clipboard to qubes clipboard, she must focus on any window belonging to this AppVM, and press **Ctrl-Shift-C**. This combination is trapped by `qubes-guid`, and `CLIPBOARD_REQ` message is sent to AppVM. `qubes-gui` responds with `CLIPBOARD_DATA` message followed by clipboard contents.
- the user focuses on other AppVM window, presses **Ctrl-Shift-V**. This combination is trapped by `qubes-guid`, and `CLIPBOARD_DATA` message followed by qubes clipboard contents is sent to AppVM; `qubes-gui` copies data to the local clipboard, and then user can paste its contents to local applications normally.

This way, the user can quickly copy clipboards between AppVMs. This action is fully controlled by the user, it cannot be triggered/forced by any AppVM.

### qubes-gui and qubes-guid code notes

Both applications are structured similarly. They use `select` function to wait for any of these two event sources:

- messages from the local X server
- messages from the vchan connecting to the remote party

The XEvents are handled by the `handle_xevent_eventname` function, and messages are handled by `handle_message_name` function. One should be very careful when altering the actual `select` loop, because both XEvents and vchan messages are buffered, and `select` will not wake for each message.

If one changes the number/order/signature of messages, one should increase the `QUBES_GUID_PROTOCOL_VERSION` constant in `messages.h` include file.

qubes-guid writes debugging information to `/var/log/qubes/qubes.domain_id.log` file; qubes-gui writes debugging information to `/var/log/qubes/gui_agent.log`. Include these files when reporting a bug.

## AppVM -> GuiVM messages

Proper handling of the below messages is security-critical. Note that all messages except for CLIPBOARD, MFNDUMP, and WINDOW\_DUMP have fixed size, so the parsing code can be small.

The `override_redirect` window attribute is explained at [Override Redirect Flag](#). The `transient_for` attribute is explained at [transient\\_for attribute](#).

Window manager hints and flags are described in the [Extended Window Manager Hints \(EWMH\)](#) spec, especially under the `_NET_WM_STATE` section.

Each message starts with the following header:

```
struct msghdr {
    uint32_t type;
    uint32_t window;
    /* This field is intended for use by GUI agents to skip unknown
     * messages from the (trusted) GUI daemon. GUI daemon, on the other
     * hand, should never rely on this field to calculate the actual len
     * of message to be read, as the (untrusted) agent can put whatever
     * it wants here! */
    uint32_t untrusted_len;
};
```

This header is followed by message-specific data:



Table 1: i

Message name	Structure after header	Action
MSG_CLIPBOARD_DATA	Any blob (in protocol before 1.2, length determined by the “window” field, in 1.2 and later - by <code>untrusted_len</code> in the header)	Store the received clipboard content (not parsed in any way)
MSG_CREATE	<pre>struct msg_create {     uint32_t x;     uint32_t y;     uint32_t width;     uint32_t height;     uint32_t parent;     uint32_t     override_redirect; };</pre>	Create a window with given parameters
MSG_DESTROY	None	Destroy a window
MSG_MAP	<pre>struct msg_map_info {     uint32_t transient_for;     uint32_t     override_redirect; };</pre>	Map a window with given parameters
MSG_UNMAP	None	Unmap a window
MSG_CONFIGURE	<pre>struct msg_configure {     uint32_t x;     uint32_t y;     uint32_t width;     uint32_t height;     uint32_t     override_redirect; };</pre>	Change window position/size/type
MSG_MFNDUMP	<pre>struct shm_cmd {     uint32_t shmidx;     uint32_t width;     uint32_t height;     uint32_t bpp;     uint32_t off;     uint32_t num_mfn;     uint32_t domid;     uint32_t mfns[0]; };</pre>	Retrieve the array of mfns that constitute the composition buffer of a remote window. The “num_mfn” 32bit integers follow the <code>shm_cmd</code> structure; “off” is the offset of the composite buffer start in the first frame; “shmidx” and “domid” parameters are just placeholders (to be filled by <code>qubes-guid</code> ), so that we can use the same structure when talking to <code>shmoverride.so</code> .
<b>1.13. Developer Documentation</b>		<b>393</b>
MSG_SHMIMAGE	<pre>struct msg_shmimage {     uint32_t</pre>	Repaint the given window fragment

## **GuiVM -> AppVM messages**

Proper handling of the below messages is NOT security-critical.

Each message starts with the following header

```
struct msghdr {  
    uint32_t type;  
    uint32_t window;  
};
```

The header is followed by message-specific data:

Table 2: i

Message name	Structure after header	Action
MSG_KEYPRESS	<pre>struct msg_keypress {     uint32_t type;     uint32_t x;     uint32_t y;     uint32_t state;     uint32_t keycode; };</pre>	Tell qubes_drv driver to generate a keypress
MSG_BUTTON	<pre>struct msg_button {     uint32_t type;     uint32_t x;     uint32_t y;     uint32_t state;     uint32_t button; };</pre>	Tell qubes_drv> driver to generate mouseclick
MSG_MOTION	<pre>struct msg_motion {     uint32_t x;     uint32_t y;     uint32_t state;     uint32_t is_hint; };</pre>	Tell qubes_drv driver to generate motion event
MSG_CONFIGURE	<pre>struct msg_configure {     uint32_t x;     uint32_t y;     uint32_t width;     uint32_t height;     uint32_t     override_redirect; };</pre>	Change window position/size/type
MSG_MAP	<pre>struct msg_map_info {     uint32_t transient_for;     uint32_t     override_redirect; };</pre>	Map a window with given parameters
MSG_CLOSE	None	send wmDeleteMessage to the window
MSG_CROSSING		Notify window about enter/leave event
<b>1.13. Developer Documentation</b>	<pre>struct msg_crossing {     uint32_t type;     uint32_t x;     uint32_t y;     uint32_t state;     uint32_t is_hint; };</pre>	<b>395</b>

KEYPRESS, BUTTON, MOTION, FOCUS messages pass information extracted from dom0 XEvent; see appropriate event documentation.

## 1.13.16 Networking

### Overall description

In Qubes, the standard Xen networking is used, based on backend driver in the driver domain and frontend drivers in VMs. In order to eliminate layer 2 attacks originating from a compromised VM, routed networking is used instead of the default bridging of `vif` devices and NAT is applied at each network hop. The default `vif-route` script had some deficiencies (requires `eth0` device to be up, and sets some redundant iptables rules), therefore the custom `vif-route-qubes` script is used.

The IP address of `eth0` interface in AppVM, as well as two IP addresses to be used as nameservers (DNS1 and DNS2), are passed via QubesDB to AppVM during its boot (thus, there is no need for DHCP daemon in the network driver domain). DNS1 and DNS2 are private addresses; whenever an interface is brought up in the network driver domain, the `/usr/lib/qubes/qubes_setup_dnat_to_ns` script sets up the DNAT iptables rules translating DNS1 and DNS2 to the newly learned real dns servers. This way AppVM networking configuration does not need to be changed when configuration in the network driver domain changes (e.g. user switches to a different WLAN). Moreover, in the network driver domain, there is no DNS server either, and consequently there are no ports open to the VMs.

### Routing tables examples

VM routing table is simple:

Destina- tion	Gateway	Genmask	Flags	Metric	Ref	Use	lface
0.0.0.0	0.0.0.0	0.0.0.0	U	0	0	0	eth0

Network driver domain routing table is a bit longer:

Destina- tion	Gateway	Genmask	Flags	Metric	Ref	Use	lface
10.137.0.16	0.0.0.0	255.255.255.255	UH	0	0	0	vif4.0
10.137.0.7	0.0.0.0	255.255.255.255	UH	0	0	0	vif10.0
10.137.0.9	0.0.0.0	255.255.255.255	UH	0	0	0	vif9.0
10.137.0.8	0.0.0.0	255.255.255.255	UH	0	0	0	vif8.0
10.137.0.12	0.0.0.0	255.255.255.255	UH	0	0	0	vif3.0
192.168.0.0	0.0.0.0	255.255.255.0	U	1	0	0	eth0
0.0.0.0	192.168.0.1	0.0.0.0	UG	0	0	0	eth0

### IPv6

Starting with Qubes 4.0, there is opt-in support for IPv6 forwarding. Similar to the IPv4, traffic is routed and NAT is applied at each network gateway. This way we avoid reconfiguring every connected qube whenever uplink connection is changed, and even telling the qube what that uplink is - which may be complex when VPN or other tunneling services are employed. The feature can be enabled on any network-providing qube, and will be propagated down the network tree, so every qube connected to it will also have IPv6 enabled. To enable the `ipv6` feature use `qvm-features` tool and set the value to 1. For example to enable it on `sys-net`, execute in dom0:

```
qvm-features sys-net ipv6 1
```

It is also possible to explicitly disable IPv6 support for some qubes, even if it is connected to IPv6-providing one. This can be done by setting `ipv6` feature to empty value:

```
qvm-features ipv4-only-qube ipv6 ''
```

This configuration is presented below - green qubes have IPv6 access, red one does not.

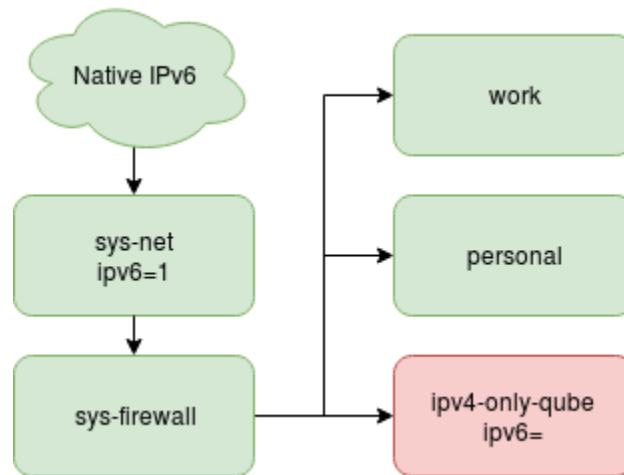


Fig. 14: ipv6-1

In that case, system uplink connection have native IPv6. But in some cases it may not be true. Then some tunneling solution can be used (for example teredo). The same will apply when the user is connected to VPN service providing IPv6 support, regardless of user's internet connection. Such configuration can be expressed by enabling `ipv6` feature only on some subset of Qubes networking, for example by creating separate qube to encapsulate IPv6 traffic and setting `ipv6` to 1 only there. See diagram below

Besides enabling IPv6 forwarding, standard Qubes firewall can be used to limit what network resources are available to each qube. Currently only `qvm-firewall` command support adding IPv6 rules, GUI firewall editor will have this ability later.

## Limitations

Currently only IPv4 DNS servers are configured, regardless of `ipv6` feature state. It is done this way to avoid reconfiguring all connected qubes whenever IPv6 DNS becomes available or not. Configuring qubes to always use IPv6 DNS and only fallback to IPv4 may result in relatively long timeouts and poor usability. But note that DNS using IPv4 does not prevent to return IPv6 addresses. In practice this is only a problem for IPv6-only networks.

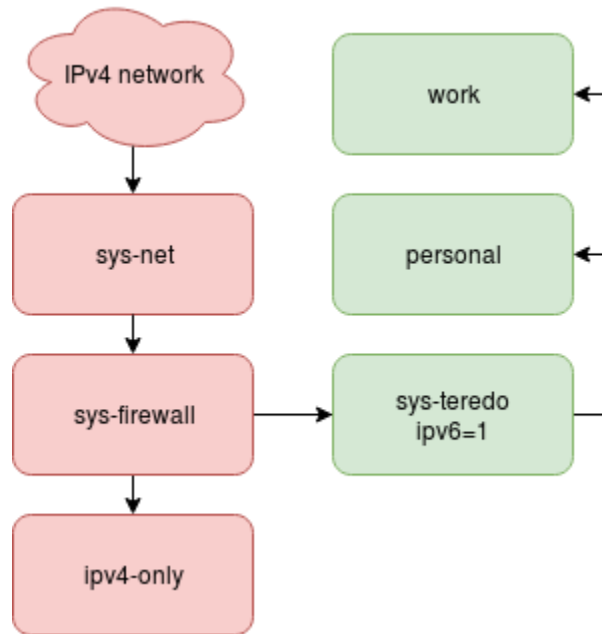


Fig. 15: ipv6-2

### 1.13.17 Template implementation

#### Block devices of a VM

Every VM has 4 block devices connected:

- **xvda** – base root device (/) – details described below
- **xvdb** – private.img – place where VM always can write.
- **xvdc** – volatile.img, discarded at each VM restart – here is placed swap and temporal “/” modifications (see below)
- **xvdd** – modules.img – kernel modules and firmware

#### private.img (xvdb)

This is mounted as /rw and here is placed all VM private data. This includes:

- /home – which is bind mounted to /rw/home
- /usr/local – which is symlink to /rw/usrlocal
- some config files (/rw/config) called by qubes core scripts (ex /rw/config/rc.local)

**Note:** Whenever a TemplateBasedVM is created, the contents of the /home directory of its parent TemplateVM *are not copied to the child TemplateBasedVM's /home*. The child TemplateBasedVM's /home is independent from its parent TemplateVM's /home, which means that any changes to the parent TemplateVM's /home will not affect the child TemplateBasedVM's /home. Once a TemplateBasedVM has been created, any changes in its /home, /usr/local, or /rw/config directories will be persistent across reboots, which means that any files stored there will still be available after restarting the TemplateBasedVM. No changes in any other directories in TemplateBasedVMs persist in this manner. If you would like to make changes in other directories which *do* persist in this manner, you must make those changes in the parent TemplateVM.

## modules.img (xvdd)

As the kernel is chosen in dom0, there must be some way to provide matching kernel modules to VM. Qubes kernel directory consists of 3 files:

- *vmlinuz* – actual kernel
- *initramfs* – initial ramdisk containing script to setup snapshot devices (see below) and mount `/lib/modules`
- *modules.img* – filesystem image of `/lib/modules` with matching kernel modules and firmware (`/lib/firmware/updates` is symlinked to `/lib/modules/firmware`)

Normally kernel “package” is common for many VMs (can be set using `qvm-prefs`). One of them can be set as default (`qvm-set-default-kernel`) to simplify kernel updates (by default all VMs use the default kernel). All installed kernels are placed in `/var/lib/qubes/vm-kernels` as separate subdirs. In this case, *modules.img* is attached to the VM as R/O device.

There is a special case when the VM can have a custom kernel – when it is updateable (StandaloneVM or TemplateVM) and the kernel is set to “none” (by `qvm-prefs`). In this case the VM uses the kernel from the “kernels” VM subdir and *modules.img* is attached as R/W device.

## Qubes TemplateVM implementation

TemplateVM has a shared *root.img* across all AppVMs that are based on it. This mechanism has some advantages over a simple common device connected to multiple VMs:

- *root.img* can be modified while there are AppVMs running – without corrupting the filesystem
- multiple AppVMs that are using different versions of *root.img* (from various points in time) can be running concurrently

There are two layers of the device-mapper snapshot device; the first one enables modifying *root.img* without stopping the AppVMs and the second one, which is contained in the AppVM, enables temporal modifications to its filesystem. These modifications will be discarded after a restart of the AppVM.

## Snapshot device in Dom0

This device consists of:

- *root.img* – real template filesystem
- *root-cow.img* – differences between the device as seen by AppVM and the current *root.img*

The above is achieved through creating device-mapper snapshots for each version of *root.img*. When an AppVM is started, a xen hotplug script (`/etc/xen/scripts/block-snapshot`) reads the inode numbers of *root.img* and *root-cow.img*; these numbers are used as the snapshot device’s name. When a device with the same name exists the new AppVM will use it – therefore, AppVMs based on the same version of *root.img* will use the same device. Of course, the device-mapper cannot use the files directly – it must be connected through `/dev/loop*`. The same mechanism detects if there is a loop device associated with a file determined by the device and inode numbers – or if creating a new loop device is necessary.

When an AppVM is stopped the xen hotplug script checks whether the device is still in use – if it is not, the script removes the snapshot and frees the loop device.

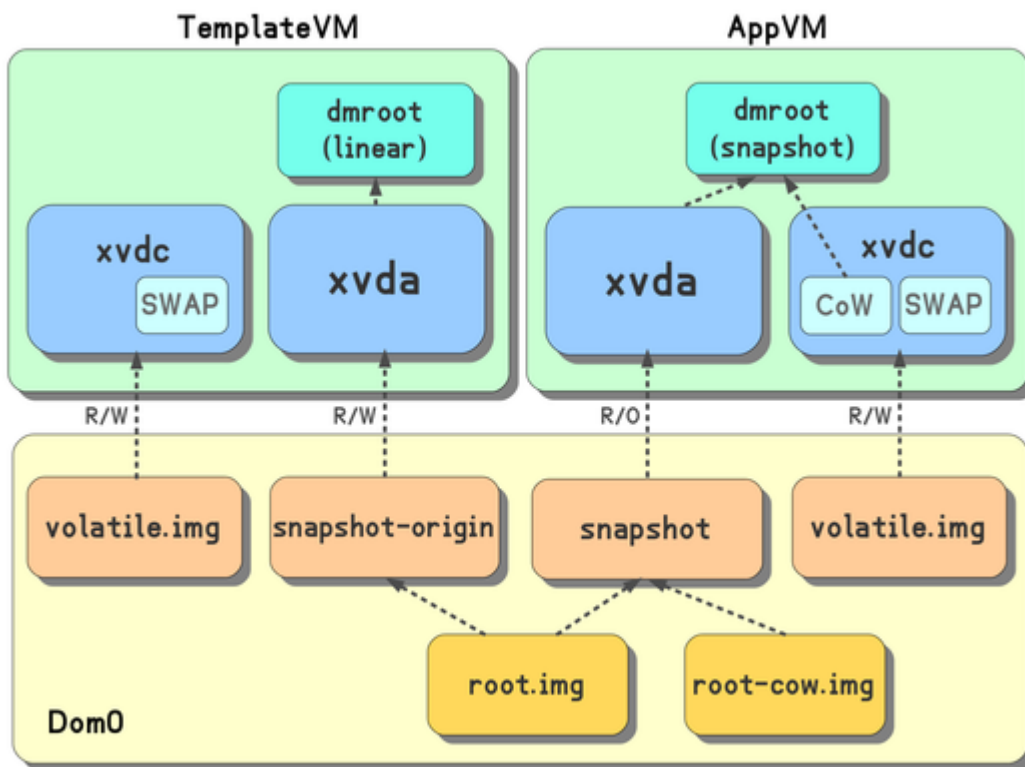


Fig. 16: TemplateSharing2.png



## Changes to template filesystem

In order for the full potential of the snapshot device to be realized, every change in `root.img` must save the original version of the modified block in `root-cow.img`. This is achieved by a snapshot-origin device.

When TemplateVM is started, it receives the snapshot-origin device connected as a root device (in read-write mode). Therefore, every change to this device is immediately saved in `root.img` – but remains invisible to the AppVM, which uses the snapshot.

When TemplateVM is stopped, the xen script moves `root-cow.img` to `root-cow.img.old` and creates a new one (using the `qvm-template-commit` tool). The snapshot device will remain untouched due to the loop device, which uses an actual file on the disk (by inode, not by name). Linux kernel frees the old `root-cow.img` files as soon as they are unused by all snapshot devices (to be exact, loop devices). The new `root-cow.img` file will get a new inode number, and so new AppVMs will get new snapshot devices (with different names).

## Rollback template changes

There is possibility to rollback last template changes. Saved `root-cow.img.old` contains all changes made during last TemplateVM run. Rolling back changes is done by reverting this “binary patch”.

This is done using snapshot-merge device-mapper target (available from 2.6.34 kernel). It requires that no other snapshot device uses underlying block devices (`root.img`, `root-cow.img` via loop device). Because of this all AppVMs based on this template must be halted during this operation.

Steps performed by **qvm-revert-template-changes**:

1. Ensure that no other VMs uses this template.
2. Prepare snapshot device with **root-cow.img.old** instead of `root-cow.img` (`/etc/xen/scripts/block-snapshot prepare`).
3. Replace `snapshot` device-mapper target with `snapshot-merge`, other parameters (chunk size etc) remains untouched. Now kernel starts merging changes stored in `root-cow.img.old` into `root.img`. d-m device can be used normally (if needed).
4. Waits for merge completed: `dmsetup status` shows used snapshot blocks – it should be equal to metadata size when completed.
5. Replace `snapshot-merge` d-m target back to `snapshot`.
6. Cleanup snapshot device (if nobody uses it at the moment).
7. Move `root-cow.img.old` to `root-cow.img` (overriding existing file).

## Snapshot device in AppVM

Root device is exposed to AppVM in read-only mode. AppVM can write only in:

- `private.img` – persistent storage (mounted in `/rw`) used for `/home`, `/usr/local` – in future versions, its use may be extended
- `volatile.img` – temporary storage, which is discarded after an AppVM restart

`volatile.img` is divided into two partitions:

1. changes to root device
2. swap partition

Inside of an AppVM, the root device is wrapped by the snapshot in the first partition of `volatile.img`. Therefore, the AppVM can write anything to its filesystem – however, such changes will be discarded after a restart.

## **StandaloneVM**

Standalone VM enables user to modify root filesystem persistently. It can be created using `–standalone` switch to `qvm-create`.

It is implemented just like TemplateVM (has own `root.img` connected as R/W device), but no other VMs can be based on it.

### **1.13.18 Audio virtualization**

VMs on Qubes OS have access to virtualized audio through the PulseAudio module. It consists of two parts:

- `pacat-simple-vchan` running in a `dom0/Audio` VM (standalone application, one per VM, connected to the PulseAudio daemon)
- `module-vchan-sink` running in a VM (loaded into the PulseAudio process)

## **Protocol**

The protocol between these two parts is designed to be as simple as possible. Specifically, there is no audio format negotiation, no compression, etc. All the audio data is transferred as raw 44.1kHz samples in S16LE format, 2 channels.

These two parts are connected with two vchan links:

1. Connection on vchan port 4713 used to transfer audio from VM to `dom0/Audio` VM. There is no negotiation, no headers, etc. Only raw samples are sent over this channel.
2. Connection on vchan port 4714 used to transfer audio from `dom0/Audio` VM to VM. Like the previous one, audio data is sent as raw samples.

Additionally, the second vchan connection (on port 4714) is used in the opposite direction (VM to `dom0`) to send simple notifications/commands about the audio state. Each such notification is a 4-byte number in little-endian format.

List of defined codes:

- `0x00010001` – VM wants to receive audio input (some process is listening); prior to this message, `pacat-simple-vchan` will not send any audio samples to the VM.
- `0x00010000` – VM does not want to receive audio input (no process is listening anymore); after this message, `pacat-simple-vchan` will not send any audio samples to the VM.
- `0x00020000` – VM does not want to send audio output; informational for `dom0`, to avoid buffer under runs (may affect PulseAudio calculated delays).
- `0x00020001` – VM does want to send audio output.

## pacat-simple-vchan

This is the dom0 (or Audio VM) part of the audio virtualization. It is responsible for transferring audio samples between the PulseAudio daemon in dom0/Audio VM (which has access to the actual audio hardware) and a VM playing/recording sounds. A separate `pacat-simple-vchan` process runs in dom0 for each VM with audio. Each of them opens separate input and output streams to their local PulseAudio, which allows for controlling the volume of each VM separately (using the `pavucontrol` tool, for example). In order to (re)create a stream for a VM in dom0, `pacat-simple-vchan` can be used. In order to find the needed parameters, domid and domain name, the command `xl list` can be used.

Audio input to the VM is not sent by default. It needs to be both requested by the VM part and explicitly enabled in `pacat-simple-vchan`. The mechanism to do this differs between Qubes versions. In Qubes before R4.1, `pacat-simple-vchan` is controlled over system D-Bus:

- destination: `org.qubesos.Audio.VMNAME` (where `VMNAME` is the VM's name)
- object path: `/org/qubesos/audio`
- interface: `org.qubesos.Audio`
- property: `RecAllowed` (which can be set using the `org.freedesktop.DBus.Properties` interface)

In Qubes R4.1 and later, `pacat-simple-vchan` is controlled over a UNIX socket at `/var/run/qubes/audio-control.VMNAME` (where `VMNAME` is the VM's name). Supported commands:

- `audio-input 1` - enable audio input
- `audio-input 0` - disable audio input

These commands can be sent using the `qubes.AudioInputEnable+VMNAME` and `qubes.AudioInputDisable+VMNAME` grexec services, respectively. The current status is written into QubesDB at `/audio-input/VMNAME` (where `VMNAME` is the VM's name) with either 1 or 0 values. The lack of a key means that the `pacat-simple-vchan` for a given VM is not running.

In either version, it is exposed to the user as device of class `mic`, which can be attached to a VM (for example, using the `qvm-device mic` command).

## 1.13.19 Template manager

This document discusses the designs and technical details of `qvm-template`, a template manager application. The goal of the project is to design a new mechanism for template distribution and a unified tool for template management.

### Motivation

This project was originally proposed in the 2020 Google Summer of Code program.

Previously, templates were distributed by RPM packages and managed by `yum/dnf`. However, tracking inherently dynamic VM images with a package manager suited for static files creates some challenges. For example, users may accidentally update the images, overriding local changes ([#996](#), [#1647](#)). (Or in the case of [#2061](#), want to specifically override the changes.) Other operations that work well on normal VMs are also somewhat inconsistent on RPM-managed templates. This includes actions such as renaming ([#839](#)), removal ([#5509](#)) and backup/restore ([#1385](#), [#1453](#), [discussion thread 1](#), [discussion thread 2](#)). In turn, this creates inconveniences and confusion for users ([#1403](#), [#4518](#)).

Also, the usage of RPM packages meant that installing a template results in arbitrary code execution, which is not ideal.

Besides distribution, users may also wish to have an integrated template management application ([#2062](#), [#2064](#), [#2534](#), [#3040](#)), as opposed to the situation where multiple programs are required for different purposes, e.g., `qubes-dom0-update`, `dnf`, `qvm-remove`, `qubes-manager`.

To tackle these issues, `qvm-template` was created. It strives to provide not only a better mechanism for handling template installation but also a consistent user-facing interface to deal with template management.

### Features

- Install/reinstall/downgrade/upgrade templates, either from local packages or remote repositories
  - Ability to install templates in alternative pools
  - Possibility for the template package to specify options such as the kernel or `virt_mode` used by the resulting template
- List and show information about local and available templates
  - Machine-readable output for easy extension
- Search for templates
- Remove templates
  - Optionally, VMs based on the template to be removed can be either removed or “disassociated” – namely, have their templates changed to a “dummy” one.
- Show available repositories
- Works in both dom0 and management VMs by utilizing the Admin API
- Works well with existing tools
- Command-line interface with DNF-like usage
- A graphical interface also available

### Package format

The RPM package format is still used. However, the contents are manually extracted instead of installing the whole package. This allows us to take advantage of existing tools for things like repository management. We can also avoid the burden of dealing with verification, reducing the risk of issues like [QSB-028](#).

The package name should be in the form `qubes-template-<TEMPLATE_NAME>`.

The package metadata (summary, description, etc.) should not contain the `|` character to avoid possibly cryptic errors. This is because of its use as an internal separator. Note that as we already consider the repository metadata untrusted. This should not result in security issues.

The file structure should be quite similar to previous template RPMs. Namely, there should be the following files in the package:

- `var/lib/qubes/vm-templates/<TEMPLATE_NAME>/root.img.part.[00,01,...]`
  - Split tarball of template `root.img`
  - Note that the file is still split due to tools such as `rpm2cpio` not supporting large files. (Notably, the `cpio` format does not support files over 4GiB.)
- `var/lib/qubes/vm-templates/<TEMPLATE_NAME>/template.conf`
  - Stores custom package metadata (as RPM does not support custom attributes).
  - Uses `KEY=VALUE` format.
  - Fields (corresponding to `qvm-prefs` and `qvm-features` tags with the same names)
    - \* `virt_mode`

- Setting this to `pv` requires user confirmation.
- Permitted values: `pv`, `pvh`, `hvm`.
- \* `kernel`
  - Only allowed to be set to "" (without quotes), i.e., "none", for PVGrub.
- \* Network-related flags: (Must be set to IPv4 addresses in the form of `x.x.x.x`.)
  - `net.fake-ip`
  - `net.fake-gateway`
  - `net.fake-netmask`
- \* Boolean flags: (Permitted values are "1" and "0", denoting "true" and "false" respectively.)
  - `no-monitor-layout`
  - `pci-e820-host`
  - `linux-stubdom`
  - `gui`
  - `gui-emulated`
  - `qrexec`
- `var/lib/qubes/vm-templates/<TEMPLATE_NAME>/whitelisted-appmenus.list`
  - Contains default app menu entries of the template itself.
- `var/lib/qubes/vm-templates/<TEMPLATE_NAME>/vm-whitelisted-appmenus.list`
  - Contains default app menu entries of VMs based on the template.
- `var/lib/qubes/vm-templates/<TEMPLATE_NAME>/netvm-whitelisted-appmenus.list`
  - Contains default app menu entries of NetVMs based on the template.
  - These three files are the same as the current format.
  - Note that the contents of these files are stored in `qvm-features` upon installation. See the section below for details.

## Metadata storage

The template manager needs to keep metadata of installed templates such as versions and origin. This data can be stored via `qvm-features` to keep things consistent when, e.g., `qvm-remove` is used. Besides, backups are also more easily handled this way.

Also, the fields can serve as an indicator of whether a template is installed by `qvm-template`.

### Fields

Most of the fields should be fairly self-explanatory.

- `template-name`
  - Note that this field needs to be consistent with the template name to be considered valid.
- `template-epoch`
- `template-version`
- `template-release`
- `template-reponame`
- `template-buildtime`
- `template-installtime`
  - The times are in UTC, and are of the format `YYYY-MM-DD HH:MM:SS`.
- `template-license`
- `template-url`
- `template-summary`
- `template-description`
  - Note that the newlines in this field are converted to `|` to work better with existing tools like `qvm-features`.
- `menu-items`
- `default-menu-items`
- `netvm-menu-items`
  - The `*menu-items` entries store the contents of `var/lib/qubes/vm-templates/<TEMPLATE_NAME>/whitelisted-appmenus.list`, `var/lib/qubes/vm-templates/<TEMPLATE_NAME>/vm-whitelisted-appmenus.list`, `var/lib/qubes/vm-templates/<TEMPLATE_NAME>/netvm-whitelisted-appmenus.list` respectively.
  - Note that newlines are converted to spaces, again for it to work better with existing tools. This should not cause ambiguity as [the FreeDesktop specifications](#) forbid spaces in `.desktop` file names.

### Repository management

For UpdateVMs to access the repository configuration, the package `qubes-repo-templates` is created with the following contents:

- `/etc/qubes/repo-templates/*.repo`: repository configuration
- `/etc/qubes/repo-templates/keys`: PGP keys

As template keys may be less trusted, they are *not* added to the system RPM keychain but instead managed separately.

## Qrexec protocol

Dom0 and management VMs without network access also need to interact with template repositories. The following qrexec calls that list and download templates are thus proposed.

- `qubes.TemplateSearch`: wraps `dnf repoquery`
- `qubes.TemplateDownload`: wraps `dnf download`

## Input

Both calls accept the following format from standard input:

```
arg1
arg2
...
argN
package-file-spec
---
repo config
```

In other words, the input consists of two parts separated by the line `---`. The first part contains some arguments and `package-file-spec` that indicates the pattern to be queried or downloaded. The following arguments are allowed:

- `--enablerepo=<repoid>`
- `--disablerepo=<repoid>`
- `--repoid=<repoid>`
- `--releasever=<release>`
- `--refresh`

where the usage is identical to that of DNF.

For the exact definition of `package-file-spec`, refer to the DNF documentation.

The second part contains the repository configurations in `yum.repos.d` format.

## Output

`qubes.TemplateSearch` prints each package in `%{name} %{epoch} %{version} %{release} %{reponame} %{downloadsize} %{description}` format to standard output, separated by newlines. Note that there is a `|` at the end of the line. This is because `%{description}` may contain newlines, and doing so allows us to split the entries by `|\n`. (As we are using `dnf repoquery --qf`, we are unable to escape the newlines in advance.)

`qubes.TemplateDownload`, on the other hand, directly outputs the downloaded content to standard output.

## Machine-readable output

The commands `qvm-template list` and `qvm-template info` provide machine-readable output in both pipe(|)-separated and JSON format. See the `qvm-template` man page for details.

## Interactions with existing tools

### `qvm-remove`

The existing `qvm-remove` tool should behave identically to `qvm-template remove` – albeit without fancy features like disassociation. This is unlike the previous situation where `qvm-remove` cannot remove RPM-installed templates.

Notably, the metadata needs no special handling as it is stored in VM features and thus automatically consistent.

## Renaming and cloning

A template is treated as non-manager-installed once renamed or cloned. However, relevant metadata in the VM features is still retained for future extension and to serve as a hint for the user.

## Further reading

Initial Google Summer of Code (2020) project proposal:

- <https://hackmd.io/aYauztkGR0iOIoh8fJLecw>

Previous design document:

- <https://gist.github.com/WillyPillow/b8a643ddbd9235a97bc187e6e44b16e4>

Discussion threads:

- [https://groups.google.com/forum/#!topic/qubes-devel/6Zb\\_WLy3GY4](https://groups.google.com/forum/#!topic/qubes-devel/6Zb_WLy3GY4)
- <https://groups.google.com/forum/#!topic/qubes-devel/PyJogqT1TUg>
- <https://groups.google.com/forum/#!topic/qubes-devel/2XaMP4Us3kg>
- [https://groups.google.com/forum/#!topic/qubes-devel/wF\\_84b1BR0A](https://groups.google.com/forum/#!topic/qubes-devel/wF_84b1BR0A)
- <https://groups.google.com/forum/#!topic/qubes-devel/pYHnihVCBM0>

### 1.13.20 Inter-qube file copying (qfilecopy)

There are two cases when we need a mechanism to copy files between VMs:

- “regular” file copy - when user instructs file manager to copy a given files/directories to a different VM
- DispVM copy - user selects “open in DispVM” on a file; this file must be copied to a DisposableVM, edited by user, and possibly a modified file copied back from DispVM to VM.

In the early days of Qubes OS, for both cases, a block device (backed by a file in dom0 with a vfat filesystem on it) was attached to VM, file(s) copied there, and then the device was detached and attached to target VM. In the DispVM case, if a edited file has been modified, another block device is passed to requester VM in order to update the source file.

This has the following disadvantages:

- performance - dom0 has to prepare and attach/detach block devices, which is slow because of hotplug scripts involvement.



- security - VM kernel parses partition table and filesystem metadata from the block device; they are controlled by (potentially untrusted) sender VM.

In modern Qubes OS releases, we have reimplemented interVM file copy using `qrexec`, which addresses the above mentioned disadvantages. Nowadays, even more generic solution (`qubes rpc`) is used. See the developer docs on `qrexec` and `qubes rpc`. In a nutshell, the file sender and the file receiver just read/write from `stdin/stdout`, and the `qubes rpc` layer passes data properly - so, no block devices are used.

The `rpc` action for regular file copy is `qubes.Filecopy`, the `rpc` client is named `qfile-agent`, the `rpc` server is named `qfile-unpacker`. For `DispVM` copy, the `rpc` action is `qubes.OpenInVM`, the `rpc` client is named `qopen-in-vm`, `rpc` server is named `vm-file-editor`. Note that the `qubes.OpenInVM` action can be done on a normal `AppVM`, too.

Being a `rpc` server, `qfile-unpacker` must be coded securely, as it processes potentially untrusted data format. Particularly, we do not want to use external `tar` or `cpio` and be prone to all vulnerabilities in them; we want a simplified, small utility, that handles only directory/file/symlink file type, permissions, `mtime/atime`, and assume user/user ownership. In the current implementation, the code that actually parses the data from `srcVM` has ca 100 lines of code and executes `chrooted` to the destination directory. The latter is hardcoded to `~user/QubesIncoming/srcVM`; because of `chroot`, there is no possibility to alter files outside of this directory.

### 1.13.21 Qubes memory manager (qmemman)

#### Rationale

Traditionally, Xen VMs are assigned a fixed amount of memory. It is not the optimal solution, as some VMs may require more memory than assigned initially, while others underutilize memory. Thus, there is a need for solution capable of shifting free memory from VM to another VM.

The `tmem` project provides a “pseudo-RAM” that is assigned on per-need basis. However this solution has some disadvantages:

- It does not provide real RAM, just an interface to copy memory to/from fast, RAM-based storage. It is perfect for swap, good for file cache, but not ideal for many tasks.
- It is deeply integrated with the Linux kernel. When Qubes will support Windows guests natively, we would have to port `tmem` to Windows, which may be challenging.

Therefore, in Qubes another solution is used. There is the `qmemman` `dom0` daemon. All VMs report their memory usage (via `xenstore`) to `qmemman`, and it makes decisions on whether to balance memory across domains. The actual mechanism to add/remove memory from a domain (`xc.domain_set_target_mem`) is already supported by both PV Linux guests and Windows guests (the latter via PV drivers).

Similarly, when there is need for Xen free memory (for instance, in order to create a new VM), traditionally the memory is obtained from `dom0` only. When `qmemman` is running, it offers an interface to obtain memory from all domains.

To sum up, `qmemman` pros and cons. Pros:

- provides automatic balancing of memory across participating PV and HVM domains, based on their memory demand
- works well in practice, with less than 1% CPU consumption in the idle case
- simple, concise implementation

Cons:

- the algorithm to calculate the memory requirement for a domain is necessarily simple, and may not closely reflect reality

- *qmemman* is notified by a VM about memory usage change not more often than 10 times per second (to limit CPU overhead in VM). Thus, there can be up to 0.1s delay until *qmemman* starts to react to the new memory requirements
- it takes more time to obtain free Xen memory, as all participating domains need to be instructed to yield memory

## Interface

*qmemman* listens for the following events:

- writes to `/local/domain/domid/memory/meminfo` xenstore keys by *meminfo-writer* process in VM. The content of this key is taken from the VM's `/proc/meminfo` pseudofile; *meminfo-writer* just strips some unused lines from it. Note that *meminfo-writer* writes its xenstore key only if the VM memory usage has changed significantly enough since the last update (by default 30MB), to prevent flooding with almost identical data
- commands issued over Unix socket `/var/run/qubes/qmemman.sock`. Currently, the only command recognized is to free the specified amount of memory. The *QMemmanClient* class implements the protocol.
- if the `/var/run/qubes/do-not-membalance` file exists, *qmemman* suspends memory balancing. It is primarily used when allocating memory for a to-be-created domain, to prevent using up the free Xen memory by the balancing algorithm before the domain creation is completed.

## Algorithms basics

The core VM property is `prefmem`. It denotes the amount of memory that should be enough for a domain to run efficiently in the nearest future. All *qmemman* algorithms will never shrink domain memory below `prefmem`. Currently, `prefmem` is simply 130% of current memory usage in a domain (without buffers and cache, but including swap). Naturally, `prefmem` is calculated by *qmemman* based on the information passed by *meminfo-writer*.

Whenever *meminfo-writer* running in domain A provides new data on memory usage to *qmemman*, the `prefmem` value for A is updated and the following balance algorithm (*qmemman\_algo.balance*) is triggered. Its output is the list of (domain\_id, new\_memory\_target\_to\_be\_set) pairs:

1. TOTAL\_PREFMEM = sum of `prefmem` of all participating domains
2. TOTAL\_MEMORY = sum of all memory assigned to participating domains plus Xen free memory
3. if TOTAL\_MEMORY > TOTAL\_PREFMEM, then redistribute TOTAL\_MEMORY across all domains proportionally to their `prefmem`
4. if TOTAL\_MEMORY < TOTAL\_PREFMEM, then
  1. for all domains whose `prefmem` is less than actual memory, shrink them to their `prefmem`
  2. redistribute memory reclaimed in the previous step between the rest of domains, proportionally to their `prefmem`

In order to avoid too frequent memory redistribution, it is actually executed only if one of the below conditions hold:

- the sum of memory size changes for all domains is more than MIN\_TOTAL\_MEMORY\_TRANSFER (150MB)
- one of the domains is below its `prefmem`, and more than MIN\_MEM\_CHANGE\_WHEN\_UNDER\_PREF (15MB) would be added to it

Additionally, the balance algorithm is tuned so that XEN\_FREE\_MEM\_LEFT (50MB) is always left as Xen free memory, to make coherent memory allocations in driver domains work.

Whenever *qmemman* is asked to return X megabytes of memory to Xen free pool, the following algorithm (*qmemman\_algo.balloon*) is executed:

1. find all domains ("donors") whose actual memory is greater than its `prefmem`

2. calculate how much memory can be reclaimed by shrinking donors to their `prefmem`. If it is less than X, return error.
3. shrink donors, proportionally to their `prefmem`, so that X MB should become free
4. wait `BALLOON_DELAY` (0.1s)
5. if some domain have not given back any memory, remove it from the donors list, and go to step 2, unless we already did `MAX_TRIES` (20) iterations (then return error).

## Notes

Conventional means of viewing the memory available to Qubes will give incorrect values for `dom0` since commands such as `free` will only show the memory allocated for `dom0`. Run the `xl info` command in `dom0` and read the `total_memory` field to see the total memory available to Qubes.

### 1.13.22 Disposable implementation

**Note:** The content below applies to Qubes R3.2.

#### DisposableVM image preparation

DisposableVM is not started like other VMs, by executing equivalent of `xl create` - it would be too slow. Instead, DisposableVM are started by restore from a savefile.

Preparing a savefile is done by `/usr/lib/qubes/qubes_prepare_saved_domain.sh` script. It takes two mandatory arguments, appvm name (APPVM) and the savefile name, and optional path to “prerun” script. The script executes the following steps:

1. APPVM is started by `qvm-start`
2. xenstore key `/local/domain/appvm_domain_id/qubes_save_request` is created
3. if prerun script was specified, copy it to `qubes_save_script` xenstore key
4. wait for the `qubes_used_mem` key to appear
5. (in APPVM) APPVM boots normally, up to the point in `/etc/init.d/qubes_core` script when the presence of `qubes_save_request` key is tested. If it exists, then
  1. (in APPVM) if exists, prerun script is retrieved from the respective xenstore key and executed. This preloads filesystem cache with useful applications, so that they will start faster.
  2. (in APPVM) the amount of used memory is stored to `qubes_used_mem` xenstore key
  3. (in APPVM) busy-waiting for `qubes_restore_complete` xenstore key to appear
6. when `qubes_used_mem` key appears, the domain memory is reduced to this amount, to make the savefile smaller.
7. APPVM private image is detached
8. the domain is saved via `xl save`
9. the COW file `volatile.img` (cow for root fs and swap) is packed to `saved_cows.tar` archive

The `qubes_prepare_saved_domain.sh` script is lowlevel. It is usually called by `qvm-create-default-dvm` script, that takes care of creating a special AppVM (named `template_name-dvm`) to be passed to `qubes_prepare_saved_domain.sh`, as well as copying the savefile to `/dev/shm` (the latter action is not done if the `/var/lib/qubes/dvmdata/dont_use_shm` file exists).

### Restoring a DisposableVM from the savefile

Normally, disposable VM is created when qubes rpc request with target *\$dispvm* is received. Then, as a part of rpc connection setup, the *qfile-daemon-dvm* program is executed; it executes */usr/lib/qubes/qubes\_restore* program. It is crucial that this program executes quickly, to make DisposableVM creation overhead bearable for the user. Its main steps are:

1. modify the savefile so that the VM name, VM UUID, MAC address and IP address are unique
2. restore the COW files from the *saved\_cows.tar*
3. create the */var/run/qubes/fast\_block\_attach* file, whose presence tells the */etc/xen/scripts/block* script to bypass some redundant checks and execute as fast as possible.
4. execute *xl restore* in order to restore a domain.
5. create the same xenstore keys as normally created when AppVM boots (e.g. *qubes\_ip*)
6. create the *qubes\_restore\_complete* xenstore key. This allows the boot process in DisposableVM to continue.

The actual passing of files between AppVM and a DisposableVM is implemented via qubes rpc.

### Validating the DisposableVM savefile

DisposableVM savefile contains references to template rootfs and to COW files. The COW files are restored before each DisposableVM start, so they cannot change. On the other hand, if templateVM is started, the template rootfs will change, and it may not be coherent with the COW files.

Therefore, the check for template rootfs modification time being older than DisposableVM savefile modification time is required. It is done in *qfilexchgd* daemon, just before restoring DisposableVM. If necessary, an attempt is made to recreate the DisposableVM savefile, using the last template used (or default template, if run for the first time) and the default prerun script, residing at */var/lib/qubes/vm-templates/templatename/dispvm\_prerun.sh*. Unfortunately, the prerun script takes a lot of time to execute - therefore, after template rootfs modification, the next DisposableVM creation can be longer by about 2.5 minutes.

## 1.13.23 Dom0 secure updates

### Reasons for Dom0 updates

Normally there should be few reasons for updating software in Dom0. This is because there is no networking in Dom0, which means that even if some bugs will be discovered e.g. in the Dom0 Desktop Manager, this really is not a problem for Qubes, because all the third-party software running in Dom0 is not accessible from VMs or network in any way. Some exceptions to the above include: Qubes GUI daemon, Xen store daemon, and disk back-ends (we plan to move the disk backends to untrusted domain in Qubes 2.0). Of course we believe this software is reasonably secure and we hope it will not need patching.

However, we anticipate some other situations when updating Dom0 software might be required:

- Updating drivers/libs for new hardware support
- Correcting non-security related bugs (e.g. new buttons for qubes manager)
- Adding new features (e.g. GUI backup tool)

## Problems with traditional network-based update mechanisms

Dom0 is the most trusted domain on Qubes OS, and for this reason we decided to design Qubes in such a way that Dom0 is not connected to any network. In fact only certain domains can be connected to a network via so-called network domains. There can also be more than one network domain, e.g. in case the user is connected to more than one physically or logically separated networks.

## Secure update mechanism we use in Qubes (starting from Beta 2)

Keeping Dom0 not connected to any network makes it hard, however, to provide updates for software in Dom0. For this reason we have come up with the following mechanism for Dom0 updates, which minimizes the amount of untrusted input processed by Dom0 software:

The update process is initiated by `qubes-dom0-update` script, running in Dom0.

Updates (\*.rpm files) are checked and downloaded by UpdateVM, which by default is the same as the firewall VM, but can be configured to be any other, network-connected VM. This is done by `qubes-download-dom0-updates.sh` script (this script is executed using `qrexec` by the previously mentioned `qubes-dom0-update`). Note that we assume that this script might get compromised and fetch maliciously compromised downloads – this is not a problem as Dom0 verifies digital signatures on updates later. The downloaded rpm files are placed in a `/var/lib/qubes/dom0-updates` directory on UpdateVM filesystem (again, they might get compromised while being kept there, still this isn't a problem). This directory is passed to yum using the `-installroot=` option.

Once updates are downloaded, the update script that runs in UpdateVM requests an RPM service `qubes.ReceiveUpdates` to be executed in Dom0. This service is implemented by `qubes-receive-updates` script running in Dom0. The Dom0's `qubes-dom0-update` script (which originally initiated the whole update process) waits until `qubes-receive-updates` finished.

The `qubes-receive-updates` script processes the untrusted input from Update VM: it first extracts the received \*.rpm files (that are sent over `qrexec` data connection) and then verifies digital signature on each file. The `qubes-receive-updates` script is a security-critical component of the Dom0 update process (as is the `qfile-dom0-unpacker.c` and the `rpm` utility, both used by the `qubes-receive-updates` for processing the untrusted input).

Once `qubes-receive-updates` finished unpacking and verifying the updates, the updates are placed in `qubes-receive-updates` directory in Dom0 filesystem. Those updates are now trusted. Dom0 is configured (see `/etc/yum.conf` in Dom0) to use this directory as a default (and only) `yum` repository.

Finally, `qubes-dom0-update` runs `yum update` that fetches the RPMs from `qubes-cached repo` and installs them as usual.

## Security benefit of our update mechanism

The benefit of the above scheme is that one doesn't need to trust the TCP/IP stack, the HTTP stack, and `wget` implementation in order to safely deliver and install updates in Dom0. One only needs to trust a few hundred lines of code as discussed above, as well as the `rpm` utility for properly checking digital signature (but this is required in any update scheme).

### 1.13.24 Qrexec: secure communication across domains

(This page is about *qrexec* v3. For *qrexec* v2, see [here](#).)

The **qrexec framework** is used by core Qubes components to implement communication between domains. Qubes domains are strictly isolated by design. However, the OS needs a mechanism to allow the administrative domain (dom0) to force command execution in another domain (VM). For instance, when a user selects an application from the KDE menu, it should start in the selected VM. Also, it is often useful to be able to pass stdin/stdout/stderr from an application running in a VM to dom0 (and the other way around). (For example, so that a VM can notify dom0 that there are updates available for it). By default, Qubes allows VMs initiate such communications in specific circumstances. The qrexec framework generalizes this process by providing a remote procedure call (RPC) protocol for the Qubes architecture. It allows users and developers to use and design secure inter-VM tools.

#### Qrexec basics: architecture and examples

Qrexec is built on top of *vchan*, a Xen library providing data links between VMs. During domain startup, a process named *qrexec-daemon* is started in dom0, and a process named *qrexec-agent* is started in the VM. They are connected over a **vchan** channel. *qrexec-daemon* listens for connections from a dom0 utility named *qrexec-client*. Let's say we want to start a process (call it *VMprocess*) in a VM (*someVM*). Typically, the first thing that a *qrexec-client* instance does is to send a request to the *qrexec-daemon*, which in turn relays it to *qrexec-agent* running in *someVM*. *qrexec-daemon* assigns unique vchan connection details and sends them to both *qrexec-client* (in dom0) and *qrexec-agent* (in *someVM*). *qrexec-client* starts a vchan server, which *qrexec-agent* then connects to. Once this channel is established, stdin/stdout/stderr from the *VMprocess* is passed between *qrexec-agent* and the *qrexec-client* process.

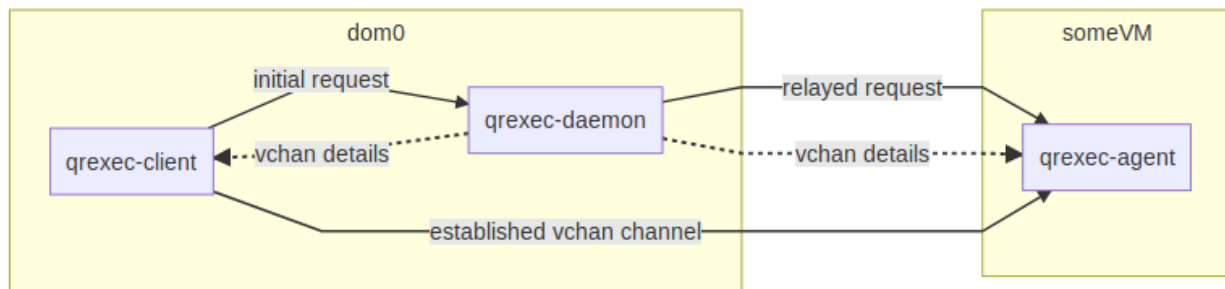


Fig. 17: qrexec basics diagram

The *qrexec-client* command is used to make connections to VMs from dom0. For example, the following command creates an empty file called *hello-world.txt* in the home folder of *someVM*:

```
$ qrexec-client -e -d someVM user:'touch hello-world.txt'
```

The string before the colon specifies what user to run the command as. The *-e* flag tells *qrexec-client* to exit immediately after sending the execution request and receiving a status code from *qrexec-agent* (whether the process creation succeeded). With this option, no further data is passed between the domains. By contrast, the following command demonstrates an open channel between dom0 and *someVM* (in this case, a remote shell):

```
$ qrexec-client -d someVM user:bash
```

The *qvm-run* command is heavily based on *qrexec-client*. It also takes care of additional activities, e.g. starting the domain if it is not up yet and starting the GUI daemon. Thus, it is usually more convenient to use *qvm-run*.

There can be an almost arbitrary number of *qrexec-client* processes for a given domain. The limiting factor is the number of available vchan channels, which depends on the underlying hypervisor, as well the domain's OS.

For more details on the qrexec framework and protocol, see “*Qubes RPC internals*.”

## Qubes RPC services

Some common tasks (like copying files between VMs) have an RPC-like structure: a process in one VM (say, the file sender) needs to invoke and send/receive data to some process in other VM (say, the file receiver). The Qubes RPC framework was created to securely facilitate a range of such actions.

Obviously, inter-VM communication must be tightly controlled to prevent one VM from taking control of another, possibly more privileged, VM. Therefore the design decision was made to pass all control communication via dom0, that can enforce proper authorization. Then, it is natural to reuse the already-existing qrexec framework.

Also, note that bare qrexec provides VM <-> dom0 connectivity, but the command execution is always initiated by dom0. There are cases when VM needs to invoke and send data to a command in dom0 (e.g. to pass information on newly installed .desktop files). Thus, the framework allows dom0 to be the RPC target as well.

Thanks to the framework, RPC programs are very simple – both RPC client and server just use their stdin/stdout to pass data. The framework does all the inner work to connect these processes to each other via `qrexec-daemon` and `qrexec-agent`. Additionally, disposable VMs are tightly integrated – RPC to a DisposableVM is identical to RPC to a normal domain, all one needs is to pass `@dispvm` as the remote domain name.

## Qubes RPC administration

### Policy files

The dom0 directory `/etc/qubes/policy.d/` contains files that set policy for each available RPC action that a VM might call. For example, `/etc/qubes/policy.d/90-default.policy` contains the default policy settings.

When making changes to existing policies it is recommended that you create a *new* policy file starting with a lower number, like `/etc/qubes/policy.d/30-user.policy`.

You may keep your custom policies in one file like `/etc/qubes/policy.d/30-user.policy`, or you may choose to have multiple files, like `/etc/qubes/policy.d/10-copy.policy`, `/etc/qubes/policy.d/10-open.policy`.

Together the contents of these files make up the RPC access policy database: the files are merged, with policies in lower number files overriding policies in higher numbered files.

Policies are defined in lines with the following format:

```
service-name|* +argument|* source destination action [options]
```

You can specify the source and destination by name or by one of the reserved keywords such as `*`, `@dispvm`, or `dom0`. (Of these three, only `*` keyword makes sense in the source field. Service calls from dom0 are currently always allowed, and `@dispvm` means “new VM created for this particular request,” so it is never a source of request.) Other methods using *tags* and *types* are also available (and discussed below).

Whenever a RPC request for an action is received, the domain checks the first matching line of the files in `/etc/qubes/policy.d/` to determine access: whether to allow the request, what VM to redirect the execution to, and what user account the program should run under. Note that if the request is redirected (`target=` parameter), policy action remains the same – even if there is another rule which would otherwise deny such request. If no policy rule is matched, the action is denied.

In the target VM, a file in either of the following locations must exist, containing the file name of the program that will be invoked, or being that program itself – in which case it must have executable permission set (`chmod +x`):  
 - `/etc/qubes-rpc/RPC_ACTION_NAME` when you make it in the template qube; - `/usr/local/etc/qubes-rpc/RPC_ACTION_NAME` for making it only in an app qube.



## Making an RPC call

From outside of dom0, RPC calls take the following form:

```
$ qrexec-client-vm target_vm_name RPC_ACTION_NAME rpc_client_path client arguments
```

For example:

```
$ qrexec-client-vm work qubes.StartApp+firefox
```

Note that only stdin/stdout is passed between RPC server and client – notably, no command line arguments are passed. By default, stderr of client and server is logged in the syslog/journald of the VM where the process is running.

It is also possible to call service without specific client program – in which case server stdin/out will be connected with the terminal:

```
$ qrexec-client-vm target_vm_name RPC_ACTION_NAME
```

## Specifying VMs: tags, types, targets, etc.

There are several methods for specifying source/target VMs in RPC policies.

- @tag:some-tag - meaning a VM with tag some-tag
- @type:type - meaning a VM of type (like AppVM, TemplateVM etc)

Target VM can be also specified as @default, which matches the case when calling VM didn't specified any particular target (either by using @default target, or empty target). For DisposableVMs, @dispvm:DISP\_VM is very similar to @dispvm but forces using a particular VM (DISP\_VM) as a base VM to be started as DisposableVM. For example:

```
* * anon-whonix @dispvm:anon-whonix-dvm allow
```

Adding such policy itself will not force usage of this particular DISP\_VM - it will only allow it when specified by the caller. But @dispvm:DISP\_VM can also be used as target in request redirection, so *it is possible* to force particular DISP\_VM usage, when caller didn't specify it:

```
* * anon-whonix @dispvm allow target=@dispvm:anon-whonix-dvm
```

Note that without redirection, this rule would allow using default Disposable VM (default\_dispvm VM property, which itself defaults to global default\_dispvm property). Also note that the request will be allowed (allow action) even if there is no second rule allowing calls to @dispvm:anon-whonix-dvm, or even if there is a rule explicitly denying it. This is because the redirection happens *after* considering the action.

The policy confirmation dialog (ask action) allows the user to specify target VM. User can choose from VMs that, according to policy, would lead to ask or allow actions. It is not possible to select VM that policy would deny. By default no VM is selected, even if the caller provided some, but policy can specify default value using default\_target= parameter. For example:

```
* * work-mail work-archive allow
* * work-mail @tag:work ask default_target=work-files
* * work-mail @default ask default_target=work-files
```

The first rule allows calls from work-mail to work-archive, without any confirmation. The second rule will ask the user about calls from work-mail VM to any VM with tag work. And the confirmation dialog will have work-files VM chosen by default, regardless of the VM specified by the caller (work-mail VM). The third rule allows the caller



to not specify target VM at all and let the user choose, still - from VMs with tag `work` (and `work-archive`, regardless of tag), and with `work-files` as default.

## RPC services and security

Be very careful when coding and adding a new RPC service. Unless the offered functionality equals full control over the target (it is the case with e.g. `qubes.VMShell` action), any vulnerability in an RPC server can be fatal to Qubes security. On the other hand, this mechanism allows to delegate processing of untrusted input to less privileged (or disposable) AppVMs, thus wise usage of it increases security.

For example, this command will run the `firefox` command in a DisposableVM based on `work`:

```
$ qvm-run --dispvm=work firefox
```

By contrast, consider this command:

```
$ qvm-run --dispvm=work --service qubes.StartApp+firefox
```

This will look for a `firefox.desktop` file in a standard location in a DisposableVM based on `work`, then launch the application described by that file. The practical difference is that the bare `qvm-run` command uses the `qubes.VMShell` service, which allows you to run an arbitrary command with arbitrary arguments, essentially providing full control over the target VM. By contrast, the `qubes.StartApp` service allows you to run only applications that are advertised in `/usr/share/applications` (or other standard locations) *without* control over the arguments, so giving a VM access to `qubes.StartApp` is much safer. While there isn't much practical difference between the two commands above when starting an application from dom0 in Qubes 4.0, there is a significant security risk when launching applications from a domU (e.g., from a separate GUI domain). This is why `qubes.StartApp` uses our standard `qrexec` argument grammar to strictly filter the permissible grammar of the `Exec=` lines in `.desktop` files that are passed from untrusted domUs to dom0, thereby protecting dom0 from command injection by maliciously-crafted `.desktop` files.

## Service policies with arguments

Sometimes a service name alone isn't enough to make reasonable `qrexec` policy. One example of such a situation is *qrexec-based USB passthrough*. Using just a service name would make it difficult to express the policy "allow access to devices X and Y, but deny to all others." It isn't feasible to create a separate service for every device: we would need to change the code in multiple files any time we wanted to update the service.

For this reason it is possible to specify a service argument, which will be subject to a policy. A service argument can make service policies more fine-grained. With arguments, it is easier to write more precise policies using the "allow" and "deny" actions, instead of relying on the "ask" method. (Writing too many "ask" policies offloads additional decisions to the user. Generally, the fewer choices the user must make, the lower the chance to make a mistake.)

The argument is specified in the second column of the policy line, as `+ARGUMENT`. If the policy uses "\*" as an argument, then it will match any argument (including no argument). As rules are processed in order, any lines with a specific argument below the line with the wildcard argument will be ignored. So for instance, we might have policies which are different depending on the argument:

```
Device +device1 * * allow
Device +device2 * * deny
Device *          * * ask
```

When calling a service that takes an argument, just add the argument to the service name separated with `+`.

```
$ qrexec-client-vm target_vm_name RPC_ACTION_NAME+ARGUMENT
```

The script will receive `ARGUMENT` as its argument. The argument will also become available as the `QREXEC_SERVICE_ARGUMENT` environment variable. This means it is possible to install a different script for a particular service argument.

See [below](#) for an example of an RPC service using an argument.

### Qubes RPC examples

To demonstrate some of the possibilities afforded by the `qrexec` framework, here are two examples of custom RPC services.

#### Simple RPC service (addition)

We can create an RPC service that adds two integers in a target domain (the server, call it “anotherVM”) and returns back the result to the invoker (the client, “someVM”). In someVM, create a file with the following contents and save it with the path `/usr/bin/our_test_add_client`:

```
#!/bin/sh
echo $1 $2           # pass data to RPC server
exec cat >&${SAVED_FD_1} # print result to the original stdout, not to the other RPC_
↪endpoint
```

Our server will be anotherVM at `/usr/bin/our_test_add_server`. The code for this file is:

```
#!/bin/sh
read arg1 arg2       # read from stdin, which is received from the RPC client
echo $((($arg1+$arg2)) # print to stdout, which is passed to the RPC client
```

We’ll need to create a service called `test.Add` with its own definition and policy file in `dom0`. Now we need to define what the service does. In this case, it should call our addition script. We define the service with a symlink at `/etc/qubes-rpc/test.Add` pointing to our server script (the script can be also placed directly in `/etc/qubes-rpc/test.Add` - make sure the file has executable bit set!):

```
ln -s /usr/bin/our_test_add_server /etc/qubes-rpc/test.Add
```

The administrative domain will direct traffic based on the current RPC policies. In `dom0`, create a file at `/etc/qubes/policy.d/30-test.policy` containing the following:

```
test.Add * * * ask
```

This will allow our client and server to communicate.

Before we make the call, ensure that the client and server scripts have executable permissions. Finally, invoke the RPC service.

```
$ qrexec-client-vm anotherVM test.Add /usr/bin/our_test_add_client 1 2
```

We should get “3” as answer. (`dom0` will ask for confirmation first.)

**Note:** For a real world example of writing a `qrexec` service, see this [blog post](#).

## RPC service with argument (file reader)

Here we create an RPC call that reads a specific file from a predefined directory on the target. This example uses an *argument* to the policy. In this example a simplified workflow will be used. The service code is placed directly in the service definition file on the target VM. No separate client script will be needed.

First, on your target VM, create two files in the home directory: `testfile1` and `testfile2`. Have them contain two different “Hello world!” lines.

Next, we define the RPC service. On the target VM, place the code below at `/etc/qubes-rpc/test.File`:

```
#!/bin/sh
argument="$1" # service argument, also available as $QREXEC_SERVICE_ARGUMENT
if [ -z "$argument" ]; then
    echo "ERROR: No argument given!"
    exit 1
fi
cat "/home/user/$argument"
```

Make sure the file is executable! (The service argument is already sanitized by qrexec framework. It is guaranteed to not contain any spaces or slashes, so there should be no need for additional path sanitization.)

Now we create the policy file in dom0, at `/etc/qubes/policy.d/30-test.policy`. The contents of the file are below. Replace “source\_vm1” and others with the names of your own chosen domains.

```
test.File +testfile1 source_vm1 target_vm allow
test.File +testfile2 source_vm2 target_vm allow
test.File * * * deny
```

With this done, we can run some tests. Invoke RPC from `source_vm1` via

```
[user@source_vm1] $ qrexec-client-vm target_vm test.File+testfile1
```

We should get the contents of `/home/user/testfile1` printed to the terminal. Invoking the service from `source_vm2` should result in a denial, but `testfile2` should work.

```
[user@source_vm2] $ qrexec-client-vm target_vm test.File+testfile1
Request refused
[user@source_vm2] $ qrexec-client-vm target_vm test.File+testfile2
```

And when invoked with other arguments or from a different VM, it should also be denied.

### 1.13.25 Qrexec v2 (deprecated)

(This page is about qrexec v2. For qrexec v3, see [here](#).)

Qubes **qrexec** is a framework for implementing inter-VM (incl. Dom0-VM) services. It offers a mechanism to start programs in VMs, redirect their stdin/stdout, and a policy framework to control this all.

## Qrexec basics

During each domain creation a process named `qrexec-daemon` is started in `dom0`, and a process named `qrexec-agent` is started in the VM. They are connected over `vchan` channel.

Typically, the first thing that a `qrexec-client` instance does is to send a request to `qrexec-agent` to start a process in the VM. From then on, the `stdin/stdout/stderr` from this remote process will be passed to the `qrexec-client` process.

E.g., to start a primitive shell in a VM type the following in `Dom0` console:

```
[user@dom0 ~]$ /usr/lib/qubes/qrexec-client -d <vm name> user:bash
```

The string before first semicolon specifies what user to run the command as.

Adding `-e` on the `qrexec-client` command line results in mere command execution (no data passing), and `qrexec-client` exits immediately after sending the execution request.

There is also the `-l <local program>` flag, which directs `qrexec-client` to pass `stdin/stdout` of the remote program not to its `stdin/stdout`, but to the (spawned for this purpose) `<local program>`.

The `qvm-run` command is heavily based on `qrexec-client`. It also takes care of additional activities (e.g., starting the domain, if it is not up yet, and starting the GUI daemon). Thus, it is usually more convenient to use `qvm-run`.

There can be almost arbitrary number of `qrexec-client` processes for a domain (i.e., `qrexec-client` processes connected to the same `qrexec-daemon`); their data is multiplexed independently.

There is a similar command line utility available inside Linux AppVMs (note the `-vm` suffix): `qrexec-client-vm` that will be described in subsequent sections.

## Qubes RPC services

Apart from simple `Dom0->VM` command executions, as discussed above, it is also useful to have more advanced infrastructure for controlled inter-VM RPC/services. This might be used for simple things like inter-VM file copy operations, as well as more complex tasks like starting a `DispVM`, and requesting it to do certain operations on a handed file(s).

Instead of implementing complex RPC-like mechanisms for inter-VM communication, Qubes takes a much simpler and pragmatic approach and aims to only provide simple *pipes* between the VMs, plus ability to request *pre-defined* programs (servers) to be started on the other end of such pipes, and a centralized policy (enforced by the `qrexec-policy` process running in `dom0`) which says which VMs can request what services from what VMs.

Thanks to the framework and automatic `stdin/stdout` redirection, RPC programs are very simple; both the client and server just use their `stdin/stdout` to pass data. The framework does all the inner work to connect these file descriptors to each other via `qrexec-daemon` and `qrexec-agent`. Additionally, `DispVMs` are tightly integrated; RPC to a `DispVM` is a simple matter of using a magic `$dispvm` keyword as the target VM name.

All services in Qubes are identified by a single string, which by convention takes a form of `qubes.ServiceName`. Each VM can provide handlers for each of the known services by providing a file in `/etc/qubes-rpc/` directory with the same name as the service it is supposed to handle. This file will then be executed by the `qrexec` service, if the `dom0` policy allowed the service to be requested (see below). Typically, the files in `/etc/qubes-rpc/` contain just one line, which is a path to the specific binary that acts as a server for the incoming request, however they might also be the actual executable themselves. Qrexec framework is careful about connecting the `stdin/stdout` of the server process with the corresponding `stdin/stdout` of the requesting process in the requesting VM (see example Hello World service described below).

## Qubes RPC administration

Besides each VM needing to provide explicit programs to serve each supported service, the inter-VM service RPC is also governed by a central policy in Dom0.

In dom0, there is a bunch of files in `/etc/qubes-rpc/policy/` directory, whose names describe the available RPC actions; their content is the RPC access policy database. Some example of the default services in Qubes are:

```
qubes.Filecopy
qubes.OpenInVM
qubes.ReceiveUpdates
qubes.SyncAppMenus
qubes.VMShell
qubes.ClipboardPaste
qubes.Gpg
qubes.NotifyUpdates
qubes.PdfConvert
```

These files contain lines with the following format:

```
srcvm destvm (allow|deny|ask) [,user=user_to_run_as] [,target=VM_to_redirect_to]
```

You can specify `srcvm` and `destvm` by name, or by one of `$anyvm`, `$dispv`, `dom0` reserved keywords (note string `dom0` does not match the `$anyvm` pattern; all other names do). Only `$anyvm` keyword makes sense in the `srcvm` field (service calls from dom0 are currently always allowed, `$dispv` means “new VM created for this particular request” - so it is never a source of request). Currently, there is no way to specify source VM by type, but this is planned for Qubes R3.

Whenever a RPC request for service named “XYZ” is received, the first line in `/etc/qubes-rpc/policy/XYZ` that matches the actual `srcvm/destvm` is consulted to determine whether to allow RPC, what user account the program should run in target VM under, and what VM to redirect the execution to. If the policy file does not exist, user is prompted to create one *manually*; if still there is no policy file after prompting, the action is denied.

On the target VM, the `/etc/qubes-rpc/XYZ` must exist, containing the file name of the program that will be invoked.

## Requesting VM-VM (and VM-Dom0) services execution

In a src VM, one should invoke the `qrexec` client via the following command:

```
/usr/lib/qubes/qrexec-client-vm <target vm name> <service name> <local program path>
↪ [local program arguments]
```

Note that only `stdin/stdout` is passed between RPC server and client – notably, no `cmdline` argument are passed.

The source VM name can be accessed in the server process via `QREXEC_REMOTE_DOMAIN` environment variable. (Note the source VM has *no* control over the name provided in this variable—the name of the VM is provided by dom0, and so is trusted.)

By default, `stderr` of client and server is logged to respective `/var/log/qubes/qrexec.XID` files, in each of the VM.

Be very careful when coding and adding a new RPC service! Any vulnerability in a RPC server can be fatal to security of the target VM!

If requesting VM-VM (and VM-Dom0) services execution *without cmdline helper*, connect directly to `/var/run/qubes/qrexec-agent-fdpass` socket as described [below](#).

## Revoking “Yes to All” authorization

Qubes RPC policy supports an “ask” action, that will prompt the user whether a given RPC call should be allowed. It is set as default for services such as inter-VM file copy. A prompt window launches in dom0, that gives the user option to click “Yes to All”, which allows the action and adds a new entry to the policy file, which will unconditionally allow further calls for given (service, srcVM, dstVM) tuple.

In order to remove such authorization, issue this command from a Dom0 terminal (example below for `qubes.Filecopy` service):

```
sudo nano /etc/qubes-rpc/policy/qubes.Filecopy
```

and then remove any line(s) ending in “allow” (before the first `##` comment) which are the “Yes to All” results.

A user might also want to set their own policies in this section. This may mostly serve to prevent the user from mistakenly copying files or text from a trusted to untrusted domain, or vice-versa.

## Qubes RPC “Hello World” service

We will show the necessary files to create a simple RPC call that adds two integers on the target VM and returns back the result to the invoking VM.

- Client code on source VM (`/usr/bin/our_test_add_client`)

```
#!/bin/sh
echo $1 $2      # pass data to rpc server
exec cat >&${SAVED_FD_1} # print result to the original stdout, not to the other rpc_
↪endpoint
```

- Server code on target VM (`/usr/bin/our_test_add_server`)

```
#!/bin/sh
read arg1 arg2 # read from stdin, which is received from the rpc client
echo $((($arg1+$arg2)) # print to stdout - so, pass to the rpc client
```

- Policy file in dom0 (`/etc/qubes-rpc/policy/test.Add`)

```
$anyvm $anyvm ask
```

- Server path definition on target VM (`/etc/qubes-rpc/test.Add`)

```
/usr/bin/our_test_add_server
```

- To test this service, run the following in the source VM:

```
/usr/lib/qubes/qrexec-client-vm <target VM> test.Add /usr/bin/our_test_add_client 1_
↪2
```

and we should get “3” as answer, provided dom0 policy allows the call to pass through, which would happen after we click “Yes” in the popup that should appear after the invocation of this command. If we changed the policy from “ask” to “allow”, then no popup should be presented, and the call will always be allowed.

**Note:** For a real world example of writing a qrexec service, see this [blog post](#).

## More high-level RPCs?

As previously noted, Qubes aims to provide mechanisms that are very simple and thus with very small attack surface. This is the reason why the inter-VM RPC framework is very primitive and doesn't include any serialization or other function arguments passing, etc. We should remember, however, that users/app developers are always free to run more high-level RPC protocols on top of qrexec. Care should be taken, however, to consider potential attack surfaces that are exposed to untrusted or less trusted VMs in that case.

## Qubes RPC internals

*(This is about the implementation of qrexec v2. For the implementation of qrexec v3, see [here](#). Note that the user API in v3 is backward compatible: qrexec apps written for Qubes R2 should run without modification on Qubes R3.)*

## Dom0 tools implementation

Players:

- `/usr/lib/qubes/qrexec-daemon`: started by mgmt stack (`qubes.py`) when a VM is started.
- `/usr/lib/qubes/qrexec-policy`: internal program used to evaluate the policy file and making the 2nd half of the connection.
- `/usr/lib/qubes/qrexec-client`: raw command line tool that talks to the daemon via unix socket (`/var/run/qubes/qrexec.XID`)

**Note:** None of the above tools are designed to be used by users.

## Linux VMs implementation

Players:

- `/usr/lib/qubes/qrexec-agent`: started by VM bootup scripts, a daemon.
- `/usr/lib/qubes/qubes-rpc-multiplexer`: executes the actual service program, as specified in VM's `/etc/qubes-rpc/qubes.XYZ`.
- `/usr/lib/qubes/qrexec-client-vm`: raw command line tool that talks to the agent.

**Note:** None of the above tools are designed to be used by users. `qrexec-client-vm` is designed to be wrapped up by Qubes apps.

## Windows VMs implementation

`%QUBES_DIR%` is the installation path (`c:\Program Files\Invisible Things Lab\Qubes OS Windows Tools` by default).

- `%QUBES_DIR%\bin\qrexec-agent.exe`: runs as a system service. Responsible both for raw command execution and interpreting RPC service requests.
- `%QUBES_DIR%\qubes-rpc`: directory with `qubes.XYZ` files that contain commands for executing RPC services. Binaries for the services are contained in `%QUBES_DIR%\qubes-rpc-services`.
- `%QUBES_DIR%\bin\qrexec-client-vm`: raw command line tool that talks to the agent.

**Note:** None of the above tools are designed to be used by users. `qrexec-client-vm` is designed to be wrapped up by Qubes apps.

## All the pieces together at work

**Note:** This section is not needed to use `qrexec` for writing Qubes apps. Also note the *qrexec framework implementation in Qubes R3* significantly differs from what is described in this section.

The VM-VM channels in Qubes R2 are made via “gluing” two VM-Dom0 and Dom0-VM vchan connections:

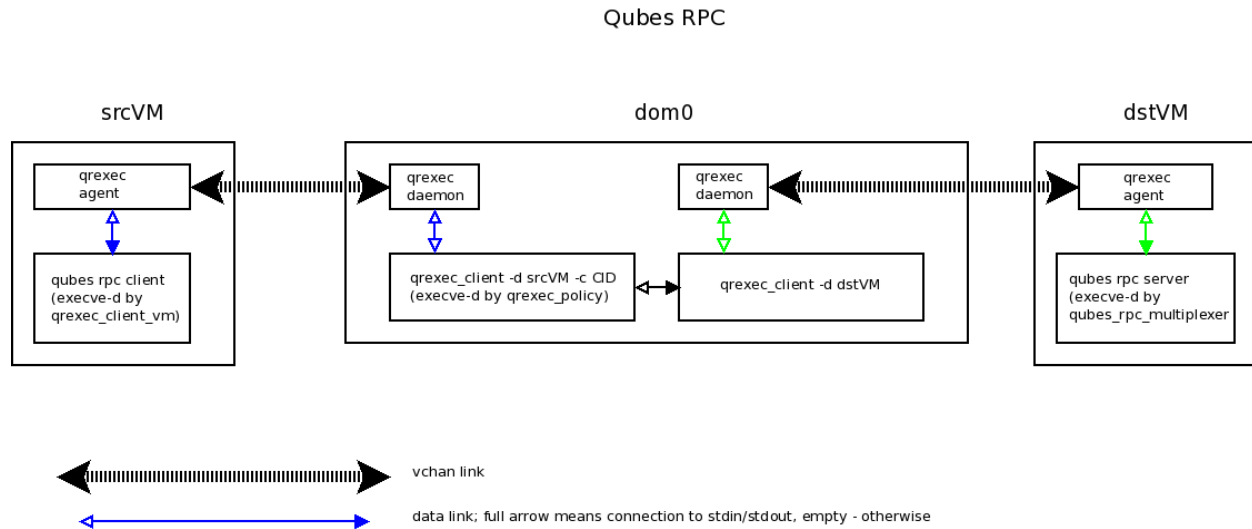


Fig. 18: qrexec2-internals.png

Note that Dom0 never examines the actual data flowing in neither of the two vchan connections.

When a user in a source VM executes `qrexec-client-vm` utility, the following steps are taken:

- `qrexec-client-vm` connects to `qrexec-agent`'s `/var/run/qubes/qrexec-agent-fdpass` unix socket 3 times. Reads 4 bytes from each of them, which is the fd number of the accepted socket in agent. These 3 integers, in text, concatenated, form “connection identifier” (CID)
- `qrexec-client-vm` writes to `/var/run/qubes/qrexec-agent` fifo a blob, consisting of target vmname, rpc action, and CID
- `qrexec-client-vm` executes the rpc client, passing the above mentioned unix sockets as process stdin/stdout, and optionally stderr (if the `PASS_LOCAL_STDERR` env variable is set)
- `qrexec-agent` passes the blob to `qrexec-daemon`, via `MSG_AGENT_TO_SERVER_TRIGGER_CONNECT_EXISTING` message over vchan
- `qrexec-daemon` executes `qrexec-policy`, passing source vmname, target vmname, rpc action, and CID as cmdline arguments
- `qrexec-policy` evaluates the policy file. If successful, creates a pair of `qrexec-client` processes, whose stdin/stdout are cross-connected.
  - The first `qrexec-client` connects to the src VM, using the `-c ClientID` parameter, which results in not creating a new process, but connecting to the existing process file descriptors (these are the fds of unix socket created in step 1).
  - The second `qrexec-client` connects to the target VM, and executes `qubes-rpc-multiplexer` command there with the rpc action as the cmdline argument. Finally, `qubes-rpc-multiplexer` executes the correct rpc server on the target.
- In the above step, if the target VM is `$dispvm`, the DispVM is created via the `qfile-daemon-dvm` program. The latter waits for the `qrexec-client` process to exit, and then destroys the DispVM.



*TODO: Protocol description (“wire-level” spec)*

### 1.13.26 Qrexec: Qubes RPC internals

*(This page details the current implementation of qrexec (qrexec3). A [general introduction](#) to qrexec is also available. For the implementation of qrexec2, see [here](#).)*

The qrexec framework consists of a number of processes communicating with each other using a common IPC protocol, described in detail below.

Components residing in the same domain (qrexec-client-vm to qrexec-agent, qrexec-client to qrexec-daemon) use local sockets as the underlying transport medium. Components in separate domains (qrexec-daemon to qrexec-agent, data channel between qrexec-agent) use vchan links. Because of [vchan limitation](#), it is not possible to establish qrexec connection back to the source domain.

#### Dom0 tools implementation

The following programs handle parts of the framework: sending and receiving requests, verifying permissions, and administering connections. These tools are not designed to be used by users directly.

##### qrexec-daemon

/usr/sbin/qrexec-daemon

One instance is required for every active domain. qrexec-daemon is responsible for both:

- handling execution and service requests from **dom0** (source: qrexec-client); and
- handling service requests from the associated domain (source: qrexec-client-vm, then qrexec-agent).

Command line usage:

qrexec-daemon domain-id domain-name [default user]

- **domain-id**: Numeric Qubes ID assigned to the associated domain.
- **domain-name**: Associated domain name.
- **default user**: Optional. If passed, qrexec-daemon uses this user as default for all execution requests that don't specify one.

##### qrexec-client

/usr/bin/qrexec-client

Used to pass execution and service requests to qrexec-daemon.

Command line usage:

- **-d target-domain-name**: Specifies the target for the execution/service request.
- **-l local-program**: Optional. If present, local-program is executed and its stdout/stdin are used when sending/receiving data to/from the remote peer.
- **-e**: Optional. If present, stdout/stdin are not connected to the remote peer. Only process creation status code is received.

- `-c <request-id,src-domain-name,src-domain-id>`: used for connecting a VM-VM service request by `qrexec-policy`. Details described below in the service example.
- `cmdline`: Command line to pass to `qrexec-daemon` as the execution/service request. Service request format is described below in the service example.

## VM tools implementation

### qrexec-agent

`/usr/lib/qubes/qrexec-agent`

One instance runs in each active domain. Responsible for:

- Handling service requests from `qrexec-client-vm` and passing them to connected `qrexec-daemon` in `dom0`.
- Executing associated `qrexec-daemon` execution/service requests.

The `qrexec-agent` command takes no parameters.

### qrexec-client-vm

`/usr/bin/qrexec-client-vm`

Runs in an active domain. Used to pass service requests to `qrexec-agent`.

Command line usage:

`qrexec-client-vm target-domain-name service-name local-program [local program arguments]`

- `target-domain-name`: Target domain for the service request. Source is the current domain.
- `service-name`: Requested service name.
- `local-program`: `local-program` is executed locally and its `stdin/stdout` are connected to the remote service endpoint.

## Qrexec protocol details

The qrexec protocol is message-based. All messages share a common header followed by an optional data packet.

```
/* uniform for all peers, data type depends on message type */
struct msg_header {
    uint32_t type;           /* message type */
    uint32_t len;           /* data length */
};
```

When two peers establish connection, the server sends `MSG_HELLO` followed by `peer_info` struct:

```
struct peer_info {
    uint32_t version; /* qrexec protocol version */
};
```

The client then should reply with its own `MSG_HELLO` and `peer_info`. The lower of two versions define protocol used for this connection. If either side does not support this version, the connection is closed.

Details of all possible use cases and the messages involved are described below.

**dom0: request execution of cmd in domX**

- **dom0:** `qrexec-client` is invoked in **dom0** as follows:

```
qrexec-client -d domX [-l local_program] user:cmd
```

(If `local_program` is set, `qrexec-client` executes it and uses that child's stdin/stdout in place of its own when exchanging data with `qrexec-agent` later.) `qrexec-client` translates that request into a `MSG_EXEC_CMDLINE` message sent to `qrexec-daemon`, with `connect_domain` set to 0 (connect to **dom0**) and `connect_port` also set to 0 (allocate a port).

- **dom0:** `qrexec-daemon` allocates a free port (in this case 513), and sends a `MSG_EXEC_CMDLINE` back to the client with connection parameters (**domX** and 513) and with command field empty. `qrexec-client` disconnects from the daemon, starts a vchan server on port 513 and awaits connection. Then, `qrexec-daemon` passes on the request as `MSG_EXEC_CMDLINE` message to the `qrexec-agent` running in **domX**. In this case, the connection parameters are **dom0** and 513.
- **domX:** `qrexec-agent` receives `MSG_EXEC_CMDLINE`, and starts the command (`user:cmd`, or `cmd` as user `user`). If possible, this is actually delegated to a separate server (`qrexec-fork-server`) also running on **domX**. After starting the command, `qrexec-fork-server` connects to `qrexec-client` in **dom0** over the provided vchan port 513.
- Data is forwarded between the `qrexec-client` in **dom0** and the command executed in **domX** using `MSG_DATA_STDIN`, `MSG_DATA_STDOUT` and `MSG_DATA_STDERR`. Empty messages (with data len field set to 0 in `msg_header`) are an EOF marker. Peer receiving such message should close the associated input/output pipe. When `cmd` terminates, **domX**'s `qrexec-fork-server` sends `MSG_DATA_EXIT_CODE` header to `qrexec-client` followed by the exit code (`int`).

**domX: request execution of service admin.Service in dom0**

- **domX:** `qrexec-client-vm` is invoked as follows:

```
qrexec-client-vm dom0 admin.Service [local_program] [params]
```

(If `local_program` is set, it will be executed in **domX** and connected to the remote command's stdin/stdout). `qrexec-client-vm` connects to `qrexec-agent` and requests service execution (`admin.Service`) in **dom0**. `qrexec-agent` assigns an internal identifier to the request. It's based on a file descriptor of the connected `qrexec-client-vm`: in this case, `SOCKET11`. `qrexec-agent` forwards the request (`MSG_TRIGGER_SERVICE3`) to its corresponding `qrexec-daemon` running in **dom0**.

- **dom0:** `qrexec-daemon` receives the request and triggers `qrexec-policy` program, passing all necessary parameters: source domain **domX**, target domain **dom0**, service `admin.Service` and identifier `SOCKET11`. `qrexec-policy` evaluates if the RPC should be allowed or denied, possibly also launching a GUI confirmation prompt. (If the RPC is denied, it returns with exit code 1, in which case `qrexec-daemon` sends a `MSG_SERVICE_REFUSED` back).
- **dom0:** If the RPC is allowed, `qrexec-policy` will launch a `qrexec-client` with the right command:

```
qrexec-client -d dom0 -c domX,X,SOCKET11 "QUBESRPC admin.Service domX name dom0"
```

The `-c domX,X,SOCKET11` are parameters indicating how connect back to **domX** and pass its input/output. The command parameter describes the RPC call: it contains service name (`admin.Service`), source domain (**domX**) and target description (`name dom0`, could also be e.g. keyword `@dispv`). The target description is important in case the original target wasn't **dom0**, but the service is executing in **dom0**. `qrexec-client` connects to a `qrexec-daemon` for **domX** and sends a `MSG_SERVICE_CONNECT` with connection parameters (**dom0**, and port 0, indicating a port should be allocated) and request identifier (`SOCKET11`). `qrexec-daemon` allocates a free

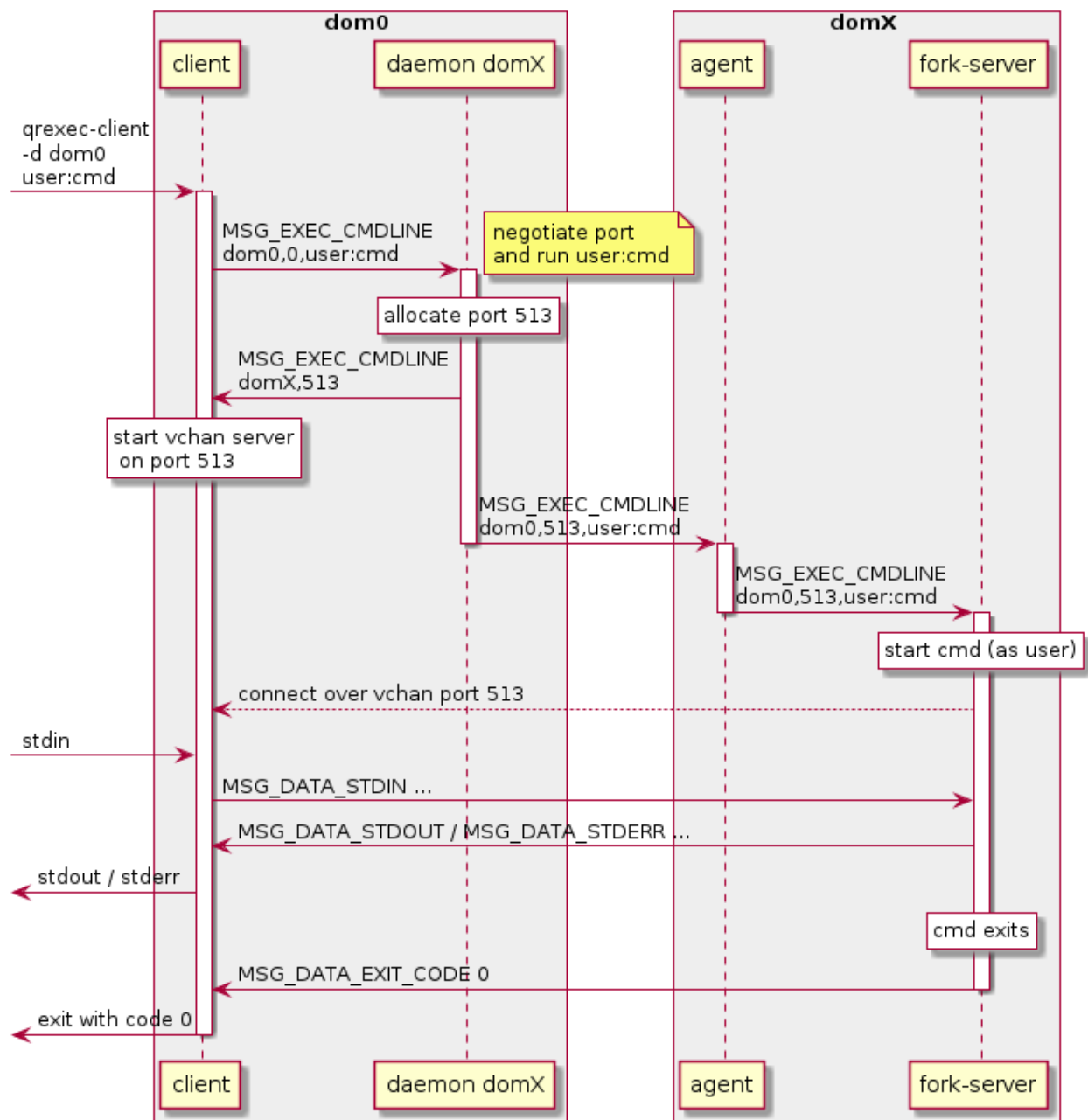


Fig. 19: qrexec internals diagram dom0-vm

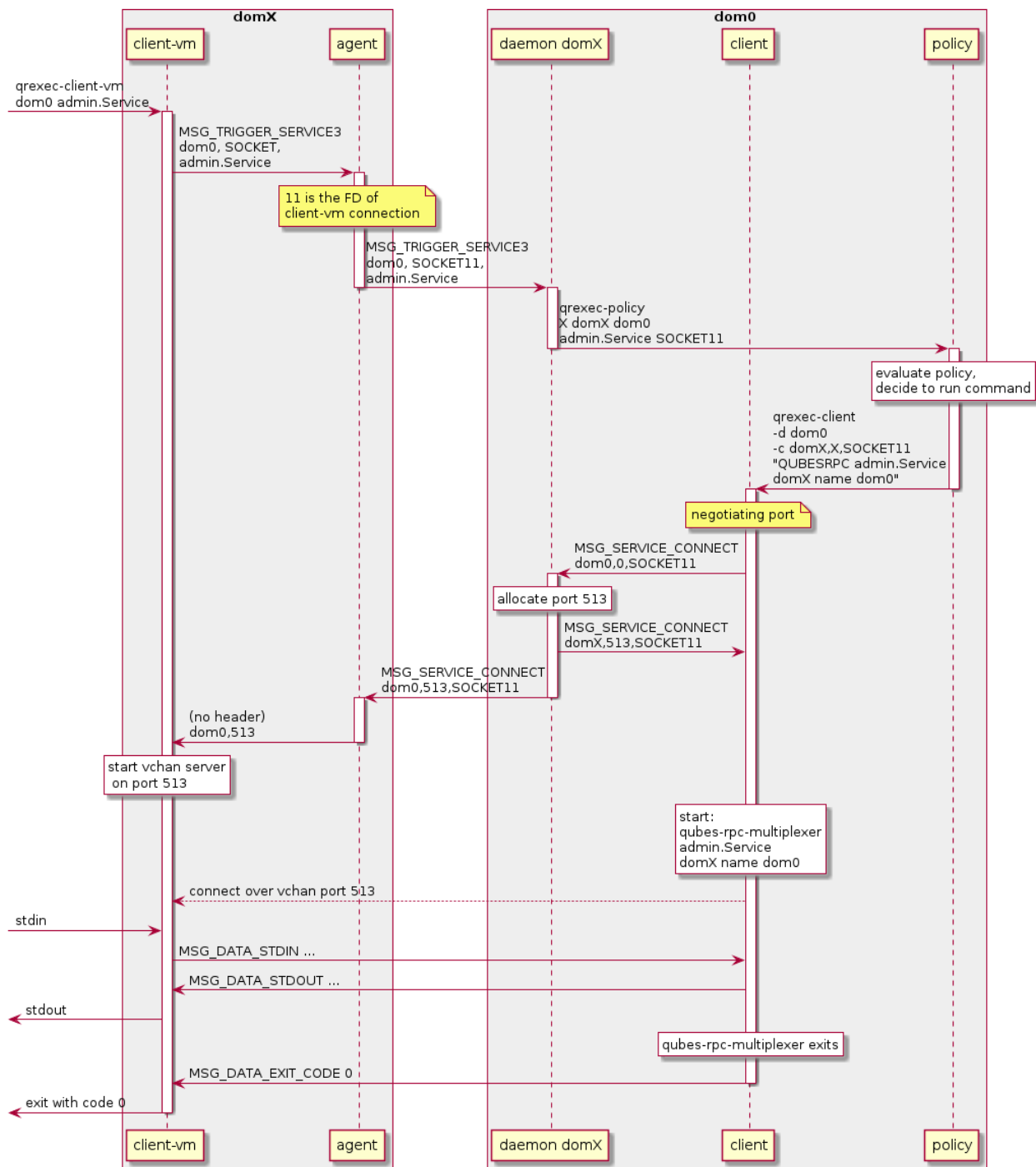


Fig. 20: qrexec internals diagram vm-dom0

port (513) and sends back connection parameters to `qrexec-client` (**domX** port 513). `qrexec-client` starts the command, and tries to connect to **domX** over the provided port 513. Then, `qrexec-daemon` forwards the connection request (`MSG_SERVICE_CONNECT`) to `qrexec-agent` running in **domX**, with the right parameters (**dom0** port 513, request `SOCKET11`).

- **dom0**: Because the command has the form `QUBESRPC: ...`, it is started through the `qubes-rpc-multiplexer` program with the provided parameters (`admin.Service domX name dom0`). That program finds and executes the necessary script in `/etc/qubes-rpc/`.
- **domX**: `qrexec-agent` receives the `MSG_SERVICE_CONNECT` and passes the connection parameters back to the connected `qrexec-client-vm`. It identifies the `qrexec-client-vm` by the request identifier (`SOCKET11` means file descriptor 11). `qrexec-client-vm` starts a vchan server on 513 and receives a connection from `qrexec-client`.
- Data is forwarded between **dom0** and **domX** as in the previous example (`dom0-VM`).

### **domX: invoke execution of qubes service `qubes.Service` in `domY`**

- **domX**: `qrexec-client-vm` is invoked as follows:

```
qrexec-client-vm domY qubes.Service [local_program] [params]
```

(If `local_program` is set, it will be executed in **domX** and connected to the remote command's stdin/stdout).

- The request is forwarded as `MSG_TRIGGER_SERVICE3` to `qrexec-daemon` running in **dom0**, then to `qrexec-policy`, then (if allowed) to `qrexec-client`. This is the same as in the previous example (`VM-dom0`).
- **dom0**: If the RPC is allowed, `qrexec-policy` will launch a `qrexec-client` with the right command:

```
qrexec-client -d domY -c domX,X,SOCKET11 user:cmd "DEFAULT:QUBESRPC qubes.Service_
↪domX"
```

The `-c domX,X,SOCKET11` are parameters indicating how connect back to **domX** and pass its input/output. The command parameter describes the service call: it contains the username (or `DEFAULT`), service name (`qubes.Service`) and source domain (**domX**). `qrexec-client` will then send a `MSG_EXEC_CMDLINE` message to `qrexec-daemon` for **domY**. The message will be with port number 0, requesting port allocation. `qrexec-daemon` for **domY** will allocate a port (513) and send it back. It will also send a `MSG_EXEC_CMDLINE` to its corresponding agent. (It will also translate `DEFAULT` to the configured default username). Then, `qrexec-client` will also send `MSG_SERVICE_CONNECT` message to **domX**'s agent, indicating that it should connect to **domY** over port 513. Having notified both domains about a connection, `qrexec-client` now exits.

- **domX**: `qrexec-agent` receives a `MSG_SERVICE_CONNECT` with connection parameters (**domY** port 513) and request identifier (`SOCKET11`). It sends the connection parameters back to the right `qrexec-client-vm`. `qrexec-client-vm` starts a vchan server on port 513. note that this is different than in the other examples: `MSG_SERVICE_CONNECT` means you should start a server, `MSG_EXEC_CMDLINE` means you should start a client.
- **domY**: `qrexec-agent` receives a `MSG_EXEC_CMDLINE` with the command to execute (`user:QUBESRPC...`) and connection parameters (**domX** port 513). It forwards the request to `qrexec-fork-server`, which handles the command and connects to **domX** over the provided port. Because the command is of the form `QUBESRPC ...`, `qrexec-fork-server` starts it using `qubes-rpc-multiplexer` program, which finds and executes the necessary script in `/etc/qubes-rpc/`.
- After that, the data is passed between **domX** and **domY** as in the previous examples (`dom0-VM`, `VM-dom0`).

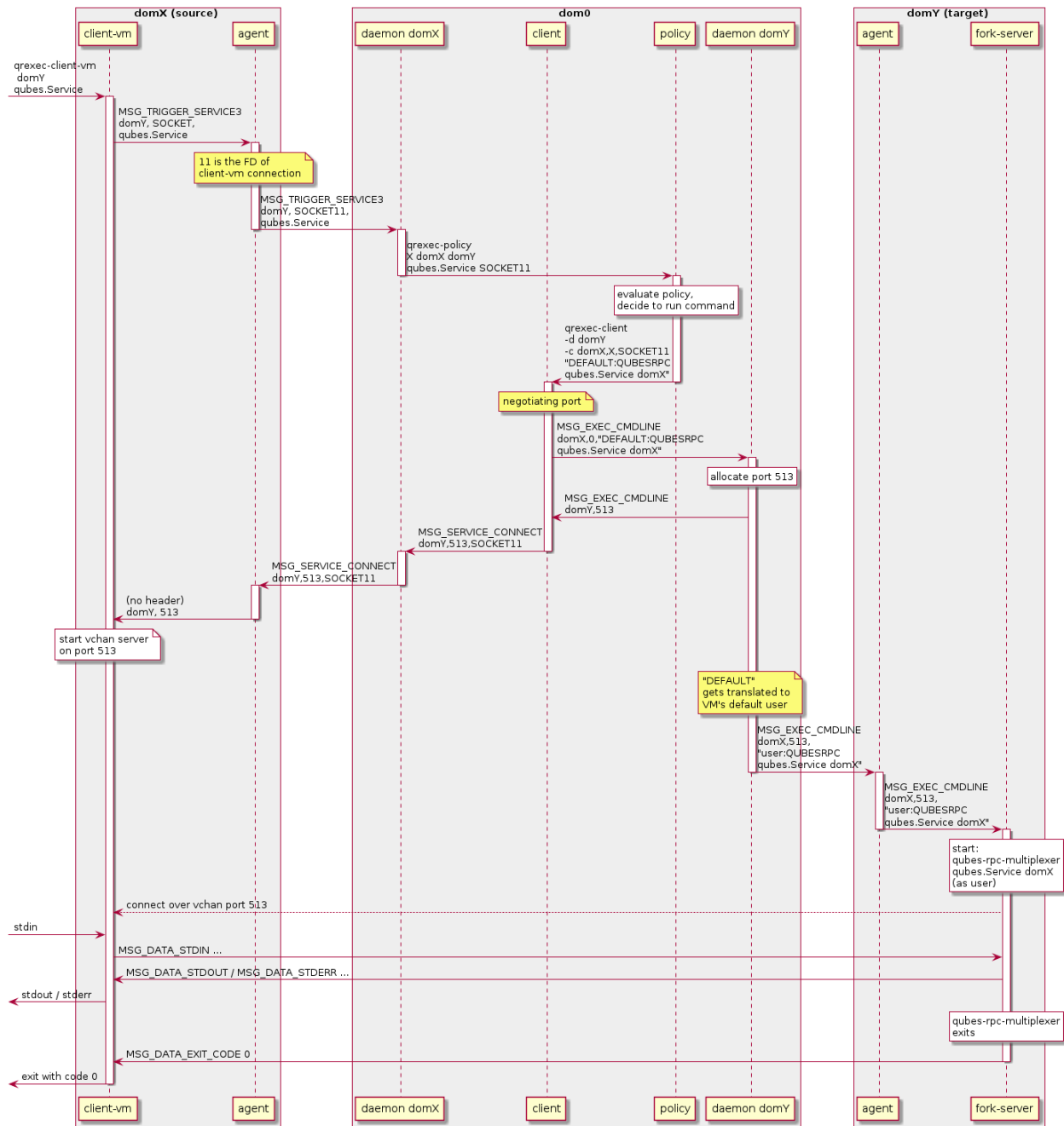


Fig. 21: qrexec internals diagram vm-vm

## qrexec-policy implementation

qrexec-policy is a mechanism for evaluating whether an RPC call should be allowed. For introduction, see [Qubes RPC administration](#).

### qrexec-policy-daemon

This is a service running in dom0. It is called by qrexec-daemon and is responsible for evaluating the request and possibly launching an action.

The daemon listens on a socket (/var/run/qubes/policy.sock). It accepts requests in the format described in [qrexec-policy-daemon.rst](#) and replies with result=allow/deny.

A standalone version is called qrexec-policy-exec and is available as a fallback.

### qrexec-policy-agent

This is a service running in the GuiVM. It is called by qrexec-policy-daemon in order to display prompts and notifications to the user.

It is a *socket-based Qubes RPC service*. Requests are in JSON format, and response is simple ASCII.

There are two endpoints:

- policy.Ask - ask the user about whether to execute a given action
- policy.Notify - notify the user about about an action.

See [qrexec-policy-agent.rst](#) for protocol details.

## 1.13.27 Qrexec: socket-based services

*This page describes how to implement and use new socket-backed services for qrexec. See [qrexec](#) for general overview of the qrexec framework.*

As of Qubes 4.1, qrexec allows implementing services not only as executable files, but also as Unix sockets. This allows Qubes RPC requests to be handled by a server running in a VM and listening for connections.

### How it works

When a Qubes RPC service is invoked, qrexec searches for a file that handles it in the qubes-rpc directories (/etc/qubes-rpc or /usr/local/etc/qubes-rpc). If the file is a Unix socket, qrexec will try to connect to it.

Before passing user input, the socket service will receive a null-terminated service descriptor, i.e. the part after QUBESRPC. When running in a VM, this is:

```
<service_name> <source>\0
```

When running in dom0, it is:

```
<service_name> <source> <target_type> <target>\0
```

(The target type can be name, in which case target is a domain name, or keyword, in which the target is a keyword like @dispvm).



Afterwards, data provided by the service's user (as stdin) is sent into the socket, and data received from the socket is sent back to the user (as stdout). When the service closes the socket, an exit code of 0 is sent back to the user.

## Differences from executable-based services

From the user point of view, the socket-based service behaves almost like an executable-based one. Here are the differences:

- There is no stderr (the socket provides only one output stream). Currently, that means stderr will also never be closed on user's end.
- There is no exit code. When the socket connection is closed, exit code 0 is sent to the user.

## Recommended use

Create a program that binds to path *outside* `/etc/qubes-rpc`, such as `/var/run/my-daemon.sock`. Put a symlink in `/etc/qubes-rpc`, e.g. `ln -s /var/run/my-daemon.sock /etc/qubes-rpc/qubes.Service`.

If your program handles multiple services, create multiple symlinks. You can dispatch based on the service descriptor.

Do not run the program as root.

You can use systemd and socket activation so that the program is started only when the service is invoked. See the below example.

## Example: qrexec-policy-agent

`qrexec-policy-agent` is the program that handles “ask” prompts for Qubes RPC calls. It is a good example of an application that: \* Uses Python and asyncio. \* Runs as a daemon, to save some overhead on starting process. \* Runs as a normal user. This is achieved using user's instance of systemd. \* Uses systemd socket activation. This way it can be installed in all VMs, but started only if it's ever needed.

See the [qubes-core-qrexec](#) repository for details.

## Systemd unit files

`/lib/systemd/user/qubes-qrexec-policy-agent.service`: This is the service configuration.

```
[Unit]
Description=Qubes remote exec policy agent
ConditionUser=!root
ConditionGroup=qubes
Requires=qubes-qrexec-policy-agent.socket

[Service]
Type=simple
ExecStart=/usr/bin/qrexec-policy-agent

[Install]
WantedBy=default.target
```

`/lib/systemd/user/qubes-qrexec-policy-agent.socket`: This is the socket file that will activate the service.

```
[Unit]
Description=Qubes remote exec policy agent socket
ConditionUser=!root
ConditionGroup=qubes
PartOf=qubes-qrexec-policy-agent.service

[Socket]
ListenStream=/var/run/qubes/policy-agent.sock

[Install]
WantedBy=sockets.target
```

Note the ConditionUser and ConditionGroup that ensure that the socket and service is started only as the right user. Start the socket using `systemctl --user start`. Enable it using `systemctl --user enable`, so that it starts automatically.

```
systemctl --user start qubes-qrexec-policy-agent.socket
systemctl --user enable qubes-qrexec-policy-agent.socket
```

Alternatively, you can enable the service by creating a symlink:

```
sudo ln -s /lib/systemd/user/qubes-qrexec-policy-agent.socket /lib/systemd/user/sockets.
↳target.wants/
```

## Link in qubes-rpc

qrexec-policy-agent will handle a Qubes RPC service called `policy.Ask`, so we add a link:

```
sudo ln -s /var/run/qubes/policy-agent.sock /etc/qubes-rpc/policy.Ask
```

## Python server with socket activation

Socket activation in systemd works by starting our program with the socket file already bound at a specific file descriptor. It's a simple mechanism based on a few environment variables, but the canonical way is to use the `sd_listen_fds()` function from systemd library (or, in our case, its Python version).

Install the Python systemd library:

```
sudo dnf install python3-systemd
```

Here is the server code:

```
import os
import asyncio
import socket

from systemd.daemon import listen_fds

class SocketService:
    def __init__(self, socket_path, socket_activated=False):
```

(continues on next page)

(continued from previous page)

```

self._socket_path = socket_path
self._socket_activated = socket_activated

async def run(self):
    server = await self.start()
    async with server:
        await server.serve_forever()

async def start(self):
    if self._socket_activated:
        fds = listen_fds()
        if fds:
            assert len(fds) == 1, 'too many listen_fds: {}'.format(
                listen_fds)
            sock = socket.socket(fileno=fds[0])
            return await asyncio.start_unix_server(self._client_connected,
                                                    sock=sock)

    if os.path.exists(self._socket_path):
        os.unlink(self._socket_path)
    return await asyncio.start_unix_server(self._client_connected,
                                            path=self._socket_path)

async def _client_connected(self, reader, writer):
    try:
        data = await reader.read()
        assert b'\0' in data, data

        service_descriptor, data = data.split(b'\0', 1)

        response = await self.handle_request(service_descriptor, data)

        writer.write(response)
        await writer.drain()
    finally:
        writer.close()
        await writer.wait_closed()

async def handle_request(self, service_descriptor, data):
    # process params, return response

    return response

def main():
    socket_path = '/var/run/qubes/policy-agent.sock'
    service = SocketService(socket_path)

    loop = asyncio.get_event_loop()
    loop.run_until_complete(service.run())

```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':  
    main()
```

You can also use `qrexec/server.py` from [qubes-core-qrexec](#) repository, which is a variant of the above code - but note that currently it's somewhat more specific (JSON requests and ASCII responses; no target handling in service descriptors).

## Using the service

The service is invoked in the same way as a standard Qubes RPC service:

```
echo <input_data> | qrexec-client -d domX 'DEFAULT:QUBESRPC policy.Ask'
```

You can also connect to it locally, but remember to include the service descriptor:

```
echo -e 'policy.Ask dom0\0<input data>' | nc -U /etc/qubes-rpc/policy.Ask
```

## Further reading

- [Qrexec overview](#)
- [Qrexec internals](#)
- [qubes-core-qrexec](#) repository - contains the above example
- `systemd.socket` - socket unit configuration
- [Streams in Python asyncio](#)

## 1.13.28 Admin API

*You may also be interested in the article [Introducing the Qubes Admin API](#).*

### Goals

The goals of the Admin API system is to provide a way for the user to manage the domains without direct access to dom0.

Foreseen benefits include:

- Ability to remotely manage the Qubes OS.
- Possibility to create multi-user system, where different users are able to use different sets of domains, possibly overlapping. This would also require to have separate GUI domain.

The API would be used by:

- Qubes OS Manager (or any tools that would replace it)
- CLI tools, when run from another VM (and possibly also from dom0)
- remote management tools
- any custom tools

## Threat model

TBD

## Components

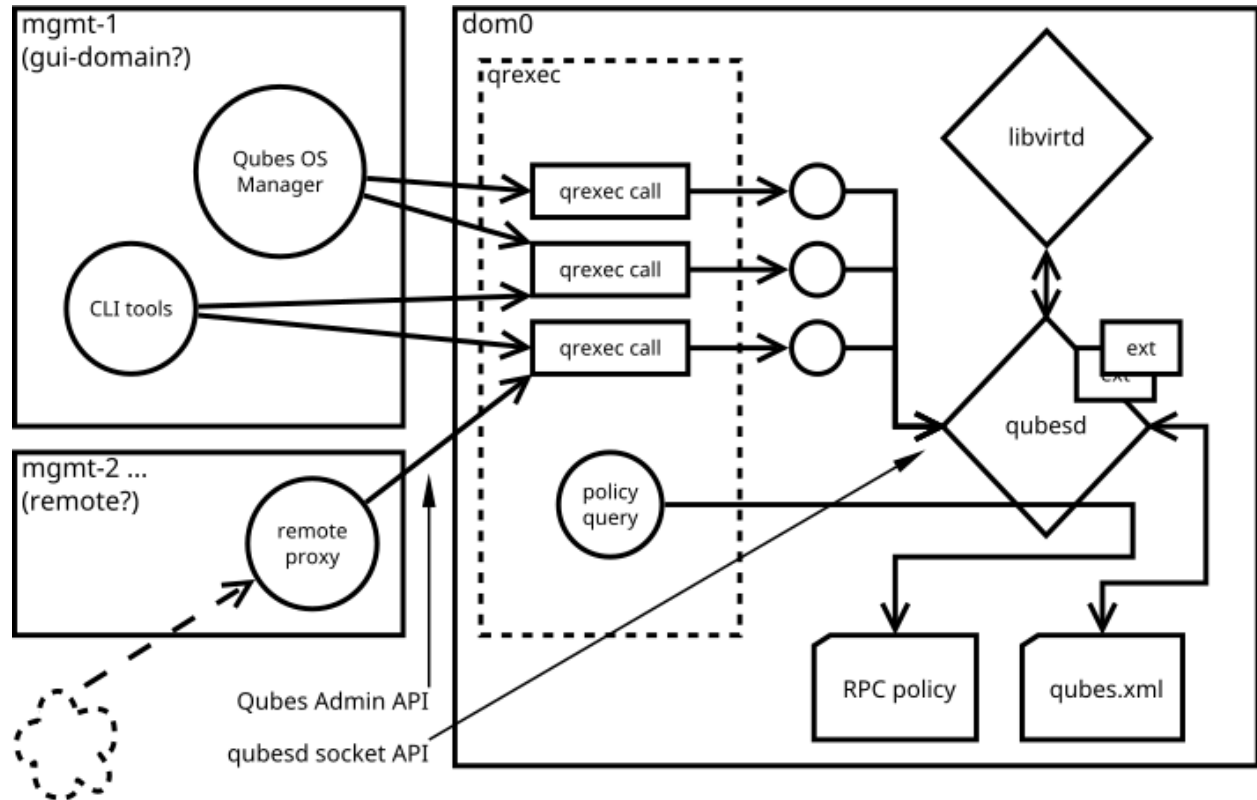


Fig. 22: Admin API Architecture

A central entity in the Qubes Admin API system is a **qubesd** daemon, which holds information about all domains in the system and mediates all actions (like starting and stopping a qube) with **libvirt**. The **qubesd** daemon also manages the **qubes.xml** file, which stores all persistent state information and dispatches events to extensions. Last but not least, **qubesd** is responsible for querying the RPC policy for **qrexec** daemon.

The **qubesd** daemon may be accessed from other domains through a set of **qrexec** API calls called the “Admin API”. This API is the intended management interface supported by the Qubes OS. The API is stable. When called, the RPC handler performs basic validation and forwards the request to the **qubesd** via UNIX domain socket. The socket API is private, unstable, and not yet documented.

## The calls

The API should be implemented as a set of qrexec calls. This is to make it easy to set the policy using current mechanism.

Table 3: i

call	dest	argument	inside	return	note
admin.vmmclass.List	dom0	-	-	<class>	
admin.vmmclass.List	dom0 <vm>		-	<name> class=<class> state=<state>	
admin.vmmclass.Create.<class>	dom0	template	name=<name> label=<label>		
admin.vmmclass.CreateInPool.<class>	dom0	template	name=<name> label=<label>, pool=<pool> pool:<volume>=<pool>		either use pool= to put all volumes there, or pool:<volume>= for individual volumes - both forms are not allowed at the same time
admin.vmmclass.CreateDisposable	template	-	-	name	Create new DisposableVM, template is any AppVM with dispvm_allowed set to True, or dom0 to use default defined in default_dispvm property of calling VM; VM created with this call will be automatically removed after its shutdown; the main difference from admin.vmmclass.Create.DispVM is automatic (random) name generation.
admin.vmm.Remove	vm	-	-	-	
admin.label.List	dom0	-	-	<property>	
admin.label.Create	dom0	label	0xRRGGBB	-	
admin.label.Get	dom0	label	-	0xRRGGBB	
admin.label.Index	dom0	label	-	<label-index>	
admin.label.Remove	dom0	label	-	-	
admin.property.List	dom0	-	-	<property>	
admin.property.Get	dom0	property	-	default={True False} type={str int bool vmclass typestr} <value>	Type list is added in R4.1. Value is suffixed with newline character.

continues on next page

Table 3 – continued from previous page

call	dest	argu- ment	inside	return	note
admin. property. GetAll	dom0	-	-	<property-name> <full-value-as-in-property- Get>	Get all the properties in one call. Each property is returned on a separate line and use the same value encoding as property.Get method, with an exception that newlines are encoded as literal \n and literal \ are encoded as \\.
admin. property. GetDefault	dom0	propety	-	type={str int bool vm label list} <value>	Types listed in R4.1. Values are of type str and each entry is suffixed with newline character.
admin. property. Help	dom0	prop- erty	-	help	
admin. property. HelpRst	dom0	prop- erty	-	help.rst	
admin. property. Reset	dom0	prop- erty	-	-	
admin. property.Set	dom0	prop- erty	value	-	
admin.vm. property. List	vm	-	-	<property>	
admin.vm. property.Get	vm	prop- erty	-	default={True False} type={str int bool vm label list} <value>	
admin.vm. property. GetAll	vm	-	-	<property-name> <full-value-as-in-property- Get>	Get all the properties in one call. Each property is returned on a separate line and use the same value encoding as property.Get method, with an exception that newlines are encoded as literal \n and literal \ are encoded as \\.
admin.vm. property. GetDefault	vm	prop- erty	-	type={str int bool vm label list} <value>	Types listed in R4.1. Each list entry is suffixed with a newline character
admin.vm. property. Help	vm	prop- erty	-	help	
admin.vm. property. HelpRst	vm	prop- erty	-	help.rst	
admin.vm. property. Reset	vm	prop- erty	-	-	
admin.vm. property.Set	vm	prop- erty	value	-	

continues on next page

Table 3 – continued from previous page

call	dest	argu- ment	inside	return	note
admin.vm. feature.List	vm	-	-	<feature>	
admin.vm. feature.Get	vm	feature	-	value	
admin.vm. feature. CheckWithTemplate	vm	feature	-	value	
admin.vm. feature. CheckWithNetvm	vm	feature	-	value	
admin.vm. feature. CheckWithAdminVM	vm	feature	-	value	
admin.vm. feature. CheckWithTemplateAndAdminVM	vm	feature	-	value	
admin.vm. feature. Remove	vm	feature	-	-	
admin.vm. feature.Set	vm	feature	value	-	
admin.vm. tag.List	vm	-	-	<tag>	
admin.vm. tag.Get	vm	tag	-	0 or 1	retcode?
admin.vm. tag.Remove	vm	tag	-	-	
admin.vm. tag.Set	vm	tag	-	-	
admin.vm. firewall.Get	vm	-	-	<rule>	rules syntax as in <i>firewall interface</i> (Firewall Rules in 4x) with addition of <code>expire=</code> and <code>comment=</code> options; <code>comment=</code> (if present) must be the last option
admin.vm. firewall.Set	vm	-	<rule>	-	set firewall rules, see admin.vm. firewall.Get for syntax
admin.vm. firewall. Reload	vm	-	-	-	force reload firewall without chang- ing any rule
admin.vm. device. <class>. Attach	vm	device	options	-	device is in form <backend-name>+<device-ident> optional options given in key=value format, separated with spaces; options can include <code>persistent=True</code> to “persis- tently” attach the device (default is temporary)

continues on next page



Table 3 – continued from previous page

call	dest	argument	inside	return	note
admin.vm. device. <class>. Detach	vm	device	-	-	device is in form <backend-name>+<device-ident>
admin.vm. device. <class>. Set. persistent	vm	device	True False	-	device is in form <backend-name>+<device-ident>
admin.vm. device. <class>. List	vm	-	-	<device> <options>	options can include persistent=True for “per- sistently” attached devices (default is temporary)
admin.vm. device. <class>. Available	vm	device- ident	-	<device-ident> <properties> description=<desc>	optional service argument may be used to get info about a single de- vice, optional (device class specific) properties are in key=value form, <i>description</i> must be the last one and is the only one allowed to contain spaces
admin.pool. List	dom0	-	-	<pool>	
admin.pool. ListDrivers	dom0	-	-	<pool-driver> <property> ...	Properties allowed in admin. pool.Add
admin.pool. Info	dom0	pool	-	<property>=<value>	
admin.pool. Add	dom0	driver	<property>=<value>		
admin. pool.Set. revisions_to_keep	dom0	pool	<value>	-	
admin.pool. Remove	dom0	pool	-	-	
admin.pool. volume.List	dom0	pool	-	volume id	
admin.pool. volume.Info	dom0	pool	vid	<property>=<value>	
admin.pool. volume.Set. revisions_to_keep	dom0	pool	<vid> <value>	-	
admin.pool. volume. ListSnapshots	dom0	pool	vid	<snapshot>	
admin.pool. volume. Snapshot	dom0	pool	vid	snapshot	
admin.pool. volume. Revert	dom0	pool	<vid> <snapshot>	-	

continues on next page

Table 3 – continued from previous page

call	dest	argument	inside	return	note
admin.pool. volume. Resize	dom0	pool	<vid> <size_in_bytes>	-	
admin.pool. volume. Import	dom0	pool	<vid> <raw volume data>	-	
admin.pool. volume. CloneFrom	dom0	pool	vid	token, to be used in admin.pool. volume.CloneTo	obtain a token to copy volume vid in pool; the token is one time use only, it's invalidated by admin. pool.volume.CloneTo, even if the operation fails
admin.pool. volume. CloneTo	dom0	pool	<vid> <token>	-	copy volume pointed by a token to volume vid in pool
admin.vm. volume.List	vm	-	-	<volume>	<volume> is per-VM volume name (root, private, etc), <vid> is pool-unique volume id
admin.vm. volume.Info	vm	vol- ume	-	<property>=<value>	
admin.vm. volume.Set. revisions_to_keep	vm	vol- ume	value	-	
admin.vm. volume. ListSnapshots	vm	vol- ume	-	snapshot	duplicate of admin.pool. volume., but with other call params
admin.vm. volume. Snapshot	vm	vol- ume	-	snapshot	id.
admin.vm. volume. Revert	vm	vol- ume	snapshot	-	id.
admin.vm. volume. Resize	vm	vol- ume	size_in_bytes		id.
admin.vm. volume. Import	vm	vol- ume	raw volume data	-	id.
admin.vm. volume. ImportWithSize	vm	vol- ume	<size_in_bytes> <raw volume data>		new version of admin.vm. volume.Import, allows new volume to be different size
admin.vm. volume.Clear	vm	vol- ume	-	-	clear contents of volume
admin.vm. volume. CloneFrom	vm	vol- ume	-	token, to be used in admin.vm.volume. CloneTo	obtain a token to copy volume of vm; the token is one time use only, it's invalidated by admin.vm. volume.CloneTo, even if the oper- ation fails

continues on next page

Table 3 – continued from previous page

call	dest	argument	inside	return	note
admin.vm. volume. CloneTo	vm	volume	token, obtained with admin. vm. volume. CloneFrom	-	copy volume pointed by a token to volume of vm
admin.vm. CurrentState	vm	-	-	<state-property>=<value>	state properties: power_state, mem, mem_static_max, cputime
admin.vm. Start	vm	-	-	-	
admin.vm. Shutdown	vm	-	-	-	
admin.vm. Pause	vm	-	-	-	
admin.vm. Unpause	vm	-	-	-	
admin.vm. Kill	vm	-	-	-	
admin. backup. Execute	dom0	config id	-	-	config in /etc/qubes/backup/ <id>.conf, only one backup operation of given config id can be running at once
admin. backup.Info	dom0	config id	-	backup info	info what would be included in the backup
admin. backup. Cancel	dom0	config id	-	-	cancel running backup operation
admin.Events	dom0   vm	-	-	events	
admin.vm. Stats	dom0   vm	-	-	vm-stats events, see below	emit VM statistics (CPU, memory usage) in form of events

Volume properties:

- pool
- vid
- size
- usage
- rw
- source
- save\_on\_stop
- snap\_on\_start
- revisions\_to\_keep
- is\_outdated

Method admin.vm.Stats returns vm-stats events every stats\_interval seconds, for every running VM. Parameters of vm-stats events:

- `memory_kb` - memory usage in kB
- `cpu_time` - absolute CPU time (in milliseconds) spent by the VM since its startup, normalized for one CPU
- `cpu_usage` - CPU usage in percents

## Returned messages

First byte of a message is a message type. This is 8 bit non-zero integer. Values start at 0x30 (48, '0', zero digit in ASCII) for readability in hexdump. Next byte must be 0x00 (a separator).

This alternatively can be thought of as zero-terminated string containing single ASCII digit.

### OK (0)

```
30 00 <content>
```

Server will close the connection after delivering single message.

### EVENT (1)

```
31 00 <subject> 00 <event> 00 ( <key> 00 <value> 00 )* 00
```

Events are returned as stream of messages in selected API calls. Normally server will not close the connection.

A method yielding events will not ever return a OK or EXCEPTION message.

When calling such method, it will produce an artificial event `connection-established` just after connection, to help avoiding race conditions during event handler registration.

### EXCEPTION (2)

```
32 00 <type> 00 ( <traceback> )? 00 <format string> 00 ( <field> 00 )*
```

Server will close the connection.

Traceback may be empty, can be enabled server-side as part of debug mode. Delimiting zero-byte is always present.

Fields are should substituted into %-style format string, possibly after client-side translation, to form final message to be displayed unto user. Server does not by itself support translation.

## Tags

The tags provided can be used to write custom policies. They are not used in a default Qubes OS installation. However, they are created anyway.

- `created-by-<vm>` — Created in an extension to qubesd at the moment of creation of the VM. Cannot be changed via API, which is also enforced by this extension.
- `managed-by-<vm>` — Can be used for the same purpose, but it is not created automatically, nor is it forbidden to set or reset this tag.

## Backup profile

Backup-related calls do not allow (yet) to specify what should be included in the backup. This needs to be configured separately in dom0, with a backup profile, stored in `/etc/qubes/backup/<profile>.conf`. The file use yaml syntax and have following settings:

- **include** - list of VMs to include, can also contains tags using `$tag:some-tag` syntax or all VMs of given type using `$type:AppVM`, known from qrexec policy
- **exclude** - list of VMs to exclude, after evaluating **include** setting
- **destination\_vm** - VM to which the backup should be send
- **destination\_path** - path to which backup should be written in **destination\_vm**. This setting is given to `qubes.Backup` service and technically it's up to it how to interpret it. In current implementation it is interpreted as a directory where a new file should be written (with a name based on the current timestamp), or a command where the backup should be piped to
- **compression** - should the backup be compressed (default: True)? The value can be either `False` or `True` for default compression, or a compression command (needs to accept `-d` argument for decompression)
- **passphrase\_text** - passphrase used to encrypt and integrity protect the backup
- **passphrase\_vm** - VM which should be asked what backup passphrase should be used. The asking is performed using `qubes.BackupPassphrase+profile_name` service, which is expected to output chosen passphrase to its stdout. Empty output cancel the backup operation. This service can be used either to ask the user interactively, or to have some automated passphrase handling (for example: generate randomly, then encrypt with a public key and send somewhere)

Not all settings needs to be set.

Example backup profile:

```
# Backup only selected VMs
include:
- work
- personal
- vault
- banking

# Store the backup on external disk
destination_vm: sys-usb
destination_path: /media/my-backup-disk

# Use static passphrase
passphrase_text: "My$Very!@Strong23Passphrase"
```

And slightly more advanced one:

```
# Include all VMs with a few exceptions
include:
- $type:AppVM
- $type:TemplateVM
- $type:StandaloneVM
exclude:
- untrusted
- $tag:do-not-backup
```

(continues on next page)

(continued from previous page)

```
# parallel gzip for faster backup
compression: pigz

# ask 'vault' VM for the backup passphrase
passphrase_vm: vault

# send the (encrypted) backup directly to remote server
destination_vm: sys-net
destination_path: ncftpput -u my-ftp-username -p my-ftp-pass -c my-ftp-server /directory/
↳ for/backups
```

## General notes

- there is no provision for `qvm-run`, but there already exists `qubes.VMShell` call
- generally actions `*.List` return a list of objects and have “object identifier” as first word in a row. Such action can be also called with “object identifier” in argument to get only a single entry (in the same format).
- closing `qrexec` connection normally does *not* interrupt running operation; this is important to avoid leaving the system in inconsistent state
- actual operation starts only after caller send all the parameters (including a payload), signaled by sending EOF mark; there is no support for interactive protocols, to keep the protocol reasonable simple

## Policy admin API

There is also an API to view and update *Qubes RPC policy files* in `dom0`. All of the following calls have `dom0` as destination:

Table 4: i

call	argument	inside	return
<code>policy.List</code> <code>policy.include.List</code>	-	-	<code>&lt;name1&gt;\n&lt;name2&gt;...</code>
<code>policy.Get</code> <code>policy.include.Get</code>	name	-	<code>&lt;token&gt;\n&lt;content&gt;</code>
<code>policy.Get</code> <code>policy.include.Get</code>	name	<code>&lt;token&gt;\n&lt;content&gt;</code>	-
<code>policy.Remove</code> <code>policy.include.Remove</code>	name	<code>&lt;token&gt;</code>	-

The `policy.*` calls refer to main policy files (`/etc/qubes/policy.d/`), and the `policy.include.*` calls refer to

the include directory (`/etc/qubes/policy.d/include/`). The `.policy` extension for files in the main directory is always omitted.

The responses do not follow admin API protocol, but signal error using an exit code and a message on stdout.

The changes are validated before saving, so that the policy cannot end up in an invalid state (e.g. syntax error, missing include file).

In addition, there is a mechanism to prevent concurrent modifications of the policy files:

- A `*.Get` call returns a file along with a *token* (currently implemented as a hash of the file).
- When calling `Replace` or `Remove`, you need to include the current token as first line. If the token does not match, the modification will fail.
- When adding a new file using `Replace`, pass `new` as token. This will ensure that the file does not exist before adding.
- To skip the check, pass `any` as token.

## TODO

- notifications
  - how to constrain the events?
  - how to pass the parameters? maybe XML, since this is trusted anyway and parser may be complicated
- how to constrain the possible values for `admin.vm.property.Set` etc, like “you can change `netvm`, but you have to pick from this set”; this currently can be done by writing an extension
- a call for executing `*.desktop` file from `/usr/share/applications`, for use with appmenus without giving access to `qubes.VMShell`; currently this can be done by writing custom `qrexec` calls
- maybe some generator for `.desktop` for appmenus, which would wrap calls in `qrexec-client-vm`

### 1.13.29 Qfileexchgd (deprecated)

**This mechanism is obsolete as of Qubes Beta 1!**

Please see this [page](#) instead.

## Overview

*qfilexchgd* is a dom0 daemon responsible for managing exchange of block devices (“virtual pendrives”) between VMs. It is used for

- copying files between AppVMs
- copying a single file between an AppVM and a DisposableVM

*qfilexchgd* is started after first *qubes\_guid* has been started, so that it has access to X display in dom0 to present dialog messages.

*qfilexchgd* is event driven. The sources of events are:

- trigger of xenstore watch for the changes in `/local/domain` xenstore hierarchy - to detect start/stop of VMs, and maintain `vmname->vm_xid` dictionary
- trigger of xenstore watch for a change in `/local/domain/domid/device/qpen` key - VMs write to this key to request service from *qfilexchgd*

## Copying files between AppVMs

1. AppVM1 user runs *qvm-copy-to-vm* script (accessible from Dolphin file manager by right click on a “file(s)->Actions->Send to VM” menu). It calls */usr/lib/qubes/qubes\_pencil new*, and it writes “new” request to its device/*qpen* xenstore key. *qfilexchg*d creates a new 1G file, makes *vfat* fs on it, and does block-attach so that this file is attached as */dev/xvdg* in AppVM1.
2. AppVM1 mounts */dev/xvdg* on */mnt/outgoing* and copies requested files there, then unmounts it.
3. AppVM1 writes “send DestVM” request to its device/*qpen* xenstore key (calling */usr/lib/qubes/qubes\_pencil send DestVM*). After getting confirmation by displaying a dialog box in dom0 display, *qfilexchg*d detaches */dev/xvdg* from AppVM1, attaches it as */dev/xvdh* to DestVM.
4. In DestVM, udev script for */dev/xvdh* named *qubes\_add\_pendrive\_script* (see */etc/udev/rules.d/qubes.rules*) mounts */dev/xvdh* on */mnt/incoming*, and then waits for */mnt/incoming* to become unmounted. A file manager running in DestVM shows a new volume, and user in DestVM may copy files from it. When user in DestVM is done, then user unmounts */mnt/incoming*. *qubes\_add\_pendrive\*\_script* then tells *\*qfilexchg*d to detach */dev/xvdh* and terminates.

## Copying a single file between AppVM and a DisposableVM

In order to minimize attack surface presented by necessity to process virtual pendrive metadata sent by (potentially compromised and malicious) DisposableVM, AppVM<->DisposableVM file exchange protocol does not use any filesystem.

1. User in AppVM1 runs *qvm-open-in-dvm* (accessible from Dolphin file manager by right click on a “file->Actions->Open in DisposableVM” menu). *qvm-open-in-dvm*
  1. gets a new */dev/xvdg* (just as described in previous paragraph)
  2. computes a new unique transaction seq SEQ by incrementing */home/user/.dvm/seq* contents,
  3. writes the requested file name (say, */home/user/document.txt*) to */home/user/.dvm/SEQ*
  4. creates a *dvm\_header* (see *core.git/appvm/dvm.h*) on */dev/xvdg*, followed by file contents
  5. writes the “send disposable SEQ” command to its device/*qpen* xenstore key.
2. *qfilexchg*d sees that “send” argument==“disposable”, and creates a new DisposableVM by calling */usr/lib/qubes/qubes\_restore*. It adds the new DisposableVM to *qubesDB* via *qvm\_collection.add\_new\_disposablevm*. Then it attaches the virtual pendrive (previously attached as */dev/xvdg* at AppVM1) as */dev/xvdh* in DisposableVM.
3. In DisposableVM, *qubes\_add\_pendrive\_script* sees non-zero *qubes\_transaction\_seq* key in xenstore, and instead processing the virtual pendrive as in the case of normal copy, treats it as DVM transaction (a request, because we run in DisposableVM). It retrieves the body of the file passed in */dev/xvdh*, copies to */tmp*, and runs *mime-open* utility to open appropriate executable to edit it. When *mime-open* returns, if the file was modified, it is sent back to AppVM1 (by writing “send AppVM1 SEQ” to device/*qpen* xenstore key). Then DisposableVM destroys itself.
4. In AppVM1, a new */dev/xvdh* appears (because DisposableVM has sent it). *qubes\_add\_pendrive\_script* sees non-zero *qubes\_transaction\_seq* key, and treats it as DVM transaction (a response, because we run in AppVM, not DisposableVM). It retrieves the filename from */home/user/.dvm/SEQ*, and copies data from */dev/xvdh* to it.



### 1.13.30 How to set up a test bench

This guide shows how to set up simple test bench that automatically test your code you're about to push. It is written especially for core3 branch of `core-admin.git` repo, but some ideas are universal.

We will set up a spare machine (bare metal, not a virtual) that will be hosting our experimental Dom0. We will communicate with it via Ethernet and SSH. This tutorial assumes you are familiar with *QubesBuilder* and you have it set up and running flawlessly.

**Notice:** This setup intentionally weakens some security properties in the testing system. So make sure you understand the risks and use exclusively for testing.

#### Setting up the Machine

##### Install ISO

First, do a clean install from the `.iso` *you built* or grabbed elsewhere (for example [here](#)).

##### Enabling Network Access in Dom0

Internet access is intentionally disabled by default in dom0. But to ease the deployment process we will give it access. The following steps should be done in dom0.

**Note:** the following assume you have only one network card. If you have two, pick one and leave the other attached to `sys-net`.

1. Remove the network card (PCI device) from `sys-net`
2. Restart your computer (for the removal to take effect)
3. Install `dhcp-client` and `openssh-server` on your testbench's dom0.
4. Save the following script in `/home/user/bin/dom0_network.sh` and make it executable. It should enable your network card in dom0. *Be sure to adjust the script's variables to suit your needs.*

```
#!/bin/sh

# adjust this for your NIC (run lspci)
BDF=0000:02:00.0

# adjust this for your network driver
DRIVER=e1000e

prog=$(basename $0)

pciunbind() {
    local path
    path=/sys/bus/pci/devices/${1}/driver/unbind
    if ! [ -w ${path} ]; then
        echo "${prog}: Device ${1} not bound"
        return 1
    fi
    echo -n ${1} >${path}
}
```

(continues on next page)

(continued from previous page)

```

pcibind() {
    local path
    path=/sys/bus/pci/drivers/${2}/bind
    if ! [ -w ${path} ]; then
        echo "${prog}: Driver ${2} not found"
        return 1
    fi
    echo ${1} >${path}
}

pciunbind ${BDF}
pcibind ${BDF} ${DRIVER}

sleep 1
dhclient

```

5. Configure your DHCP server so your testbench gets static IP and connect your machine to your local network. You should ensure that your testbench can reach the Internet.
6. You'll need to run the above script on every startup. To automate this save the following systemd service `/etc/systemd/system/dom0-network-direct.service`

```

[Unit]
Description=Connect network to dom0

[Service]
Type=oneshot
ExecStart=/home/user/bin/dom0_network.sh

[Install]
WantedBy=multi-user.target

```

7. Then, enable and start the SSH Server and the script on boot:

```

sudo systemctl enable sshd
sudo systemctl start sshd

sudo systemctl enable dom0-network-direct
sudo systemctl start dom0-network-direct

```

**Note:** If you want to install additional software in dom0 and your only network card was assigned to dom0, then *instead* of the usual `sudo qubes-dom0-update <PACKAGE>` now you run `sudo dnf --setopt=reposdir=/etc/yum.repos.d install <PACKAGE>`.

## Install Tests and Their Dependencies

A regular Qubes installation isn't ready to run the full suite of tests. For example, in order to run the [Split GPG tests](#) you need to have the `qubes-gpg-split-tests` package installed in your app qubes.

Because of the above reason, some additional configurations need to be done to your testing environment. This can be done in an automated manner with the help of the [Salt](#) configuration that provisions the [automated testing environment](#).

The following commands should work for you, but do keep in mind that the provisioning scripts are designed for the [openQA environment](#) and not your specific local testing system. Run the following in `dom0`:

```
# For future reference the following commands are an adaptation of
# https://github.com/marmarek/openqa-tests-qubesos/blob/master/tests/update.pm

# Install git
sudo qubes-dom0-update git || sudo dnf --setopt=reposdir=/etc/yum.repos.d install git

# Download the openQA automated testing environment Salt configuration
git clone https://github.com/marmarek/openqa-tests-qubesos/
cd openqa-tests-qubesos/extra-files
sudo cp -a system-tests/ /srv/salt/
sudo qubesctl top.enable system-tests

# Install the same configuration as the one in openQA
QUBES_VERSION=4.1
PILLAR_DIR=/srv/pillar/base/update
sudo mkdir -p $PILLAR_DIR
printf 'update:\n  qubes_ver: '$QUBES_VERSION'\n' | sudo tee $PILLAR_DIR/init.sls
printf "base:\n  '*':\n    - update\n" | sudo tee $PILLAR_DIR/init.top
sudo qubesctl top.enable update pillar=True

# Apply states to dom0 and VMs
# NOTE: These commands can take several minutes (if not more) without showing output
sudo qubesctl --show-output state.highstate
sudo qubesctl --max-concurrency=2 --skip-dom0 --templates --show-output state.highstate
```

## Development VM

### SSH

Arrange firewall so you can reach the testbench from your `qubes-dev` VM. Generate SSH key in `qubes-dev`:

```
ssh-keygen -t ecdsa -b 521
```

Add the following section in `.ssh/config` in `qubes-dev`:

```
Host testbench
  # substitute username in testbench
  User user
  # substitute address of your testbench
  HostName 192.168.123.45
```

## Passwordless SSH Login

To log to your testbench without entering password every time, copy your newly generated public key (`id_ecdsa.pub`) to `~/.ssh/authorized_keys` on your testbench. You can do this easily by running this command on `qubes-dev`: `ssh-copy-id -i ~/.ssh/id_ecdsa.pub user@192.168.123.45` (substituting with the actual username address of your testbench).

## Scripting

This step is optional, but very helpful. Put these scripts somewhere in your `${PATH}`, like `/usr/local/bin`.

qtb-runtests:

```
#!/bin/sh

ssh testbench python -m qubes.tests.run
```

qtb-install:

```
#!/bin/sh

TMPDIR=/tmp/qtb-rpms

if [ $# -eq 0 ]; then
    echo "usage: $(basename $0) <rpmfile> ..."
    exit 2
fi

set -e

ssh testbench mkdir -p "${TMPDIR}"
scp "${@}" testbench:"${TMPDIR}" || echo "check if you have 'scp' installed on your ↵
↵testbench"

while [ $# -gt 0 ]; do
    ssh testbench sudo rpm -i --replacepkgs --replacefiles "${TMPDIR}/${(basename ${1})}
    ↵)"
    shift
done
```

qtb-iterate:

```
#!/bin/sh

set -e

# substitute path to your builder installation
pushd ${HOME}/builder >/dev/null

# the following are needed only if you have sources outside builder
#rm -rf qubes-src/core-admin
#make COMPONENTS=core-admin get-sources
```

(continues on next page)

(continued from previous page)

```
make core-admin
qtb-install qubes-src/core-admin/rpm/x86_64/qubes-core-dom0-*.rpm
qtb-runtests
```

## Hooking git

I (woju) have those two git hooks. They ensure tests are passing (or are marked as expected failure) when committing and pushing. For committing it is only possible to run tests that may be executed from git repo (even if the rest were available, I probably wouldn't want to do that). For pushing, I also install RPM and run tests on testbench.

core-admin/.git/hooks/pre-commit: (you may retain also the default hook, here omitted for readability)

```
#!/bin/sh

set -e

python -c "import sys, qubes.tests.run; sys.exit(not qubes.tests.run.main())"
```

core-admin/.git/hooks/pre-push:

```
#!/bin/sh

exec qtb-iterate
```

### 1.13.31 Automated tests

#### Unit and Integration Tests

Starting with Qubes R3 we use `python unittest` to perform automatic tests of Qubes OS. Despite the name, we use it for both `unit tests` and `integration tests`. The main purpose is, of course, to deliver much more stable releases.

The integration tests must be run in dom0, but some unit tests can run inside a VM as well.

#### Integration & unit testing in dom0

Integration tests are written with the assumption that they will be executed on dedicated hardware and must be run in dom0. All other unit tests can also be run in dom0.

**Do not run the tests on installations with important data, because you might lose it.**

All the VMs with a name starting with `test-` on the installation are removed during the process, and all the tests are recklessly started from dom0, even when testing (& possibly breaking) VM components.

First you need to build all packages that you want to test. Please do not mix branches as this will inevitably lead to failures. Then setup Qubes OS with these packages installed.

For testing you'll have to stop the `qubesd` service as the tests will use its own custom variant of the service: `sudo systemctl stop qubesd`

Don't forget to start it after testing again.

To start testing you can then use the standard python unittest runner:

```
sudo -E python3 -m unittest -v qubes.tests
```

Alternatively, use the custom Qubes OS test runner:

```
sudo -E python3 -m qubes.tests.run -v
```

Our test runner runs mostly the same as the standard one, but it has some nice additional features like colored output and not needing the “qubes.test” prefix.

You can use `python3 -m qubes.tests.run -h` to get usage information:

```
[user@dom0 ~]$ python3 -m qubes.tests.run -h
usage: run.py [-h] [--verbose] [--quiet] [--list] [--failfast] [--no-failfast]
              [--do-not-clean] [--do-clean] [--loglevel LEVEL]
              [--logfile FILE] [--syslog] [--no-syslog] [--kmsg] [--no-kmsg]
              [TESTNAME [TESTNAME ...]]

positional arguments:
  TESTNAME              list of tests to run named like in description
                        (default: run all tests)

optional arguments:
  -h, --help            show this help message and exit
  --verbose, -v         increase console verbosity level
  --quiet, -q           decrease console verbosity level
  --list, -l            list all available tests and exit
  --failfast, -f        stop on the first fail, error or unexpected success
  --no-failfast         disable --failfast
  --loglevel LEVEL, -L LEVEL
                        logging level for file and syslog forwarding (one of:
                        NOTSET, DEBUG, INFO, WARN, WARNING, ERROR, CRITICAL;
                        default: DEBUG)
  --logfile FILE, -o FILE
                        if set, test run will be also logged to file
  --syslog              reenable logging to syslog
  --no-syslog           disable logging to syslog
  --kmsg, --very-brave-or-very-stupid
                        log most important things to kernel ring-buffer
  --no-kmsg, --i-am-smarter-than-kay-sievers
                        do not abuse kernel ring-buffer
  --allow-running-along-qubesd
                        allow running in parallel with qubesd; this is
                        DANGEROUS and WILL RESULT IN INCONSISTENT SYSTEM STATE
  --break-to-repl       break to REPL after tests
```

When running only specific tests, write their names like **in** log, **in** format:  
 MODULE+“/”+CLASS+“/”+FUNCTION. MODULE should omit initial “qubes.tests.”.  
 Example: basic/TC\_00\_Basic/test\_000\_create

For instance, to run only the tests for the fedora-21 template, you can use the `-l` option, then filter the list:

```
[user@dom0 ~]$ python3 -m qubes.tests.run -l | grep fedora-21
network/VmNetworking_fedora-21/test_000_simple_networking
network/VmNetworking_fedora-21/test_010_simple_proxyvm
network/VmNetworking_fedora-21/test_020_simple_proxyvm_nm
network/VmNetworking_fedora-21/test_030_firewallvm_firewall
network/VmNetworking_fedora-21/test_040_inter_vm
```

(continues on next page)

(continued from previous page)

```

vm_qrexec_gui/TC_00_AppVM_fedora-21/test_000_start_shutdown
vm_qrexec_gui/TC_00_AppVM_fedora-21/test_010_run_gui_app
vm_qrexec_gui/TC_00_AppVM_fedora-21/test_050_qrexec_simple_eof
vm_qrexec_gui/TC_00_AppVM_fedora-21/test_051_qrexec_simple_eof_reverse
vm_qrexec_gui/TC_00_AppVM_fedora-21/test_052_qrexec_vm_service_eof
vm_qrexec_gui/TC_00_AppVM_fedora-21/test_053_qrexec_vm_service_eof_reverse
vm_qrexec_gui/TC_00_AppVM_fedora-21/test_060_qrexec_exit_code_dom0
vm_qrexec_gui/TC_00_AppVM_fedora-21/test_065_qrexec_exit_code_vm
vm_qrexec_gui/TC_00_AppVM_fedora-21/test_100_qrexec_filecopy
vm_qrexec_gui/TC_00_AppVM_fedora-21/test_110_qrexec_filecopy_deny
vm_qrexec_gui/TC_00_AppVM_fedora-21/test_120_qrexec_filecopy_self
vm_qrexec_gui/TC_20_DispatchVM_fedora-21/test_000_prepare_dvm
vm_qrexec_gui/TC_20_DispatchVM_fedora-21/test_010_simple_dvm_run
vm_qrexec_gui/TC_20_DispatchVM_fedora-21/test_020_gui_app
vm_qrexec_gui/TC_20_DispatchVM_fedora-21/test_030_edit_file
[user@dom0 ~]$ sudo -E python3 -m qubes.tests.run -v `python3 -m qubes.tests.run -l |
↵grep fedora-21`

```

Example test run:

```

File Edit View Bookmarks Settings Help
[user@dom0 qubes]$ python -m qubes.tests.run -v
dom0: basic/TC_00_Basic/test_000_create ... ok
dom0: basic/TC_01_Properties/test_000_rename ... ok
dom0: basic/TC_01_Properties/test_010_netvm ... ok
dom0: basic/TC_01_Properties/test_020_dispvnetvm ... ok
dom0: basic/TC_01_Properties/test_030_clone ... ok
dom0: dom0_update/TC_00_Dom0Upgrade/test_000_update ... skipped (rpm-sign and/or rpm-build not installed)
dom0: dom0_update/TC_00_Dom0Upgrade/test_010_install ... skipped (rpm-sign and/or rpm-build not installed)
dom0: dom0_update/TC_00_Dom0Upgrade/test_020_install_wrong_sign ... skipped (rpm-sign and/or rpm-build not installed)
dom0: dom0_update/TC_00_Dom0Upgrade/test_030_install_unsigned ... skipped (rpm-sign and/or rpm-build not installed)
dom0: network/VmNetworking_fedora-21/test_000_simple_networking ... ok
dom0: network/VmNetworking_fedora-21/test_010_simple_proxyvm ... ok
dom0: network/VmNetworking_fedora-21/test_020_simple_proxyvm_nm ... ok
dom0: network/VmNetworking_fedora-21/test_030_firewallvm_firewall ... ok
dom0: network/VmNetworking_fedora-21/test_040_inter_vm ... ok
dom0: network/VmNetworking_fedora-21/test_050_spoof_ip
  Test if VM IP spoofing is blocked ... ok
dom0: vm_qrexec_gui/TC_10_HVM/test_000_create_start ... ok
dom0: vm_qrexec_gui/TC_10_HVM/test_010_create_start_template ... ok
dom0: vm_qrexec_gui/TC_10_HVM/test_020_create_start_template_vm ... ok
dom0: vm_qrexec_gui/TC_10_HVM/test_030_prevent_simultaneous_start ... ok
dom0: vm_qrexec_gui/TC_30_Gui_daemon/test_000_clipboard ... ok
dom0: vm_qrexec_gui/TC_00_AppVM_fedora-21/test_000_start_shutdown ... ok
dom0: vm_qrexec_gui/TC_00_AppVM_fedora-21/test_010_run_gui_app ... ok
dom0: vm_qrexec_gui/TC_00_AppVM_fedora-21/test_050_qrexec_simple_eof
  Test for data and EOF transmission dom0->VM ... ok
dom0: vm_qrexec_gui/TC_00_AppVM_fedora-21/test_051_qrexec_simple_eof_reverse
  Test for EOF transmission VM->dom0 ... ok
dom0: vm_qrexec_gui/TC_00_AppVM_fedora-21/test_052_qrexec_vm_service_eof
  Test for EOF transmission VM(src)->VM(dst) ... ok
dom0: vm_qrexec_gui/TC_00_AppVM_fedora-21/test_053_qrexec_vm_service_eof_reverse
  Test for EOF transmission VM(src)<-VM(dst) ... unexpected success
dom0: vm_qrexec_gui/TC_00_AppVM_fedora-21/test_055_qrexec_dom0_service_abort ... ok
dom0: vm_qrexec_gui/TC_00_AppVM_fedora-21/test_060_qrexec_exit_code_dom0 ... ok
dom0: vm_qrexec_gui/TC_00_AppVM_fedora-21/test_065_qrexec_exit_code_vm ... expected failure
dom0: vm_qrexec_gui/TC_00_AppVM_fedora-21/test_100_qrexec_filecopy ... ok
dom0: vm_qrexec_gui/TC_00_AppVM_fedora-21/test_110_qrexec_filecopy_deny ... ok
dom0: vm_qrexec_gui/TC_00_AppVM_fedora-21/test_120_qrexec_filecopy_self ... skipped (Xen gntalloc driver crashes when page is mapped in the same domain)
dom0: vm_qrexec_gui/TC_00_AppVM_fedora-21/test_200_timezone
  Test whether timezone setting is properly propagated to the VM ... ok
dom0: vm_qrexec_gui/TC_00_AppVM_fedora-21/test_210_time_sync
  Test time synchronization mechanism ... Time sync failed, aborting!
Thu Oct 1 04:04:51 UTC 2015
FAIL
dom0: vm_qrexec_gui/TC_20_DispatchVM_fedora-21/test_000_prepare_dvm ... /qubes-used-mem
Disk detached successfully

Domain fedora-21-dvm saved to /var/lib/qubes/appvms/fedora-21-dvm/dvm-savefile
DVM savefile created successfully.
ok
dom0: vm_qrexec_gui/TC_20_DispatchVM_fedora-21/test_010_simple_dvm_run ... ok
dom0: vm_qrexec_gui/TC_20_DispatchVM_fedora-21/test_020_gui_app ... FAIL
dom0: vm_qrexec_gui/TC_20_DispatchVM_fedora-21/test_030_edit_file ... ok
dom0: backup/TC_00_Backup/test_000_basic_backup ... ok
dom0: backup/TC_00_Backup/test_001_compressed_backup ... ok
dom0: backup/TC_00_Backup/test_002_encrypted_backup ... ok

```

Fig. 23: snapshot-tests2.png

Tests are also compatible with nose2 test runner, so you can use this instead:

```
sudo systemctl stop qubesd; sudo -E nose2 -v --plugin nose2.plugins.loader.loadtests_
↪qubes.tests; sudo systemctl start qubesd
```

This may be especially useful together with various nose2 plugins to store tests results (for example `nose2.plugins.junitxml`), to ease presenting results. This is what we use on [OpenQA](#).

## Unit testing inside a VM

Many unit tests will also work inside a VM. However all of the tests requiring a dedicated VM to be run (mostly the integration tests) will be skipped.

Whereas integration tests are mostly stored in the [qubes-core-admin](#) repository, unit tests can be found in each of the Qubes OS repositories.

To for example run the `qubes-core-admin` unit tests, you currently have to clone at least [qubes-core-admin](#) and its dependency [qubes-core-qrexec](#) repository in the branches that you want to test.

The below example however will assume that you set up a build environment as described in the [Qubes Builder documentation](#).

Assuming you cloned the `qubes-builder` repository to your home directory inside a fedora VM, you can use the following commands to run the unit tests:

```
cd ~
sudo dnf install python3-pip lvm2 python35 python3-virtualenv
virtualenv -p /usr/bin/python35 python35
source python35/bin/activate
python3 -V
cd ~/qubes-builder/qubes-src/core-admin
pip3 install -r ci/requirements.txt
export PYTHONPATH=./core-qrexec:test-packages
./run-tests
```

To run only the tests related to e.g. `lvm`, you may use:

```
./run-tests -v $(python3 -m qubes.tests.run -l | grep lvm)
```

You can later re-use the created virtual environment including all of the via `pip3` installed packages with `source ~/python35/bin/activate`.

We recommend to run the unit tests with the Python version that the code is meant to be run with in `dom0` (3.5 was just an example above). For instance, the `release4.0` (Qubes 4.0) branch is intended to be run with Python 3.5 whereas the Qubes 4.1 branch (`master` as of 2020-07) is intended to be run with Python 3.7 or higher. You can always check your `dom0` installation for the Python version of the current stable branch.

## Tests configuration

Test runs can be altered using environment variables:

- `DEFAULT_LVM_POOL` - LVM thin pool to use for tests, in `VolumeGroup/ThinPool` format
- `QUBES_TEST_PCIDEV` - PCI device to be used in PCI passthrough tests (for example sound card)
- `QUBES_TEST_TEMPLATES` - space separated list of templates to run tests on; if not set, all installed templates are tested



- QUBES\_TEST\_LOAD\_ALL - load all tests (including tests for all templates) when relevant test modules are imported; this needs to be set for test runners not supporting `load_tests` protocol

### Adding a new test to core-admin

After adding a new unit test to `core-admin/qubes/tests` you'll have to include it in `core-admin/qubes/tests/__init__.py`

### Editing `__init__.py`

You'll also need to add your test at the bottom of the `__init__.py` file, in the method `def load_tests`, in the for loop with `modname`. Again, given the hypothetical `example.py` test:

```
for modname in (
    'qubes.tests.basic',
    'qubes.tests.dom0_update',
    'qubes.tests.network',
    'qubes.tests.vm_qrexec_gui',
    'qubes.tests.backup',
    'qubes.tests.backupcompatibility',
    'qubes.tests.regressions',
    'qubes.tests.example', # This is our newly added test
):
```

### Testing PyQt applications

When testing (Py)QT applications, it's useful to create a separate `QApplication` object for each test. But QT framework does not allow multiple `QApplication` objects in the same process at the same time. This means it's critical to reliably cleanup the previous instance before creating a new one. This turns out to be a non-trivial task, especially if *any* test uses the event loop. Failure to perform proper cleanup in many cases results in `SEGV`. Below you can find steps for the proper cleanup:

```
import asyncio
import quamash
import unittest
import gc

class SomeTestCase(unittest.TestCase):
    def setUp(self):
        [...]

        # force "cleanlooks" style, the default one on Xfce (GtkStyle) use
        # static variable internally and caches pointers to later destroyed
        # objects (result: SEGV)
        self.qtapp = QtGui.QApplication(["test", "-style", "cleanlooks"])

        # construct event loop even if this particular test doesn't use it,
        # otherwise events with qtapp references will be queued there anyway and the
        # first test that actually use event loop will try to dereference (already
        # destroyed) objects, resulting in SEGV
        self.loop = quamash.QEventLoop(self.qtapp)
```

(continues on next page)

(continued from previous page)

```

def tearDown(self):
    [...]
    # process any pending events before destroying the object
    self.qtapp.processEvents()

    # queue destroying the QApplication object, do that for any other QT
    # related objects here too
    self.qtapp.deleteLater()

    # process any pending events (other than just queued destroy), just in case
    self.qtapp.processEvents()

    # execute main loop, which will process all events, _including just queued_
    ↪destroy_
    self.loop.run_until_complete(asyncio.sleep(0))

    # at this point it QT objects are destroyed, cleanup all remaining references;
    # del other QT object here too
    self.loop.close()
    del self.qtapp
    del self.loop
    gc.collect()

```

## Automated tests with openQA

**URL:** <https://openqa.qubes-os.org/> **Tests:** <https://github.com/marmarek/openqa-tests-qubesos>

Manually testing Qubes OS and its installation is a time-consuming process. We use **OpenQA** to automate this process. It works by installing Qubes in KVM and interacting with it as a user would, including simulating mouse clicks and keyboard presses. Then, it checks the output to see whether various tests were passed, e.g. by comparing the virtual screen output to screenshots of a successful installation.

Using openQA to automatically test the Qubes installation process works as of Qubes 4.0-rc4 on 2018-01-26, provided that the versions of KVM and QEMU are new enough and the hardware has VT-x and EPT. KVM also supports nested virtualization, so HVM should theoretically work. In practice, however, either Xen or QEMU crashes when this is attempted. Nonetheless, PV works well, which is sufficient for automated installation testing.

Thanks to present and past donors who have provided the infrastructure for Qubes' openQA system with hardware that meets these requirements.

## Looking for patterns in tests

In order to better visualize patterns in tests the `openqa_investigator` script can be used. It feeds off of the openQA test data to make graph plots. Here is an example:

Some outputs: - plot by tests - plot by errors - markdown

Some filters: - filter by error - filter by test name

Check out the script's help with `python3 openqa_investigator.py --help` to see all available options.

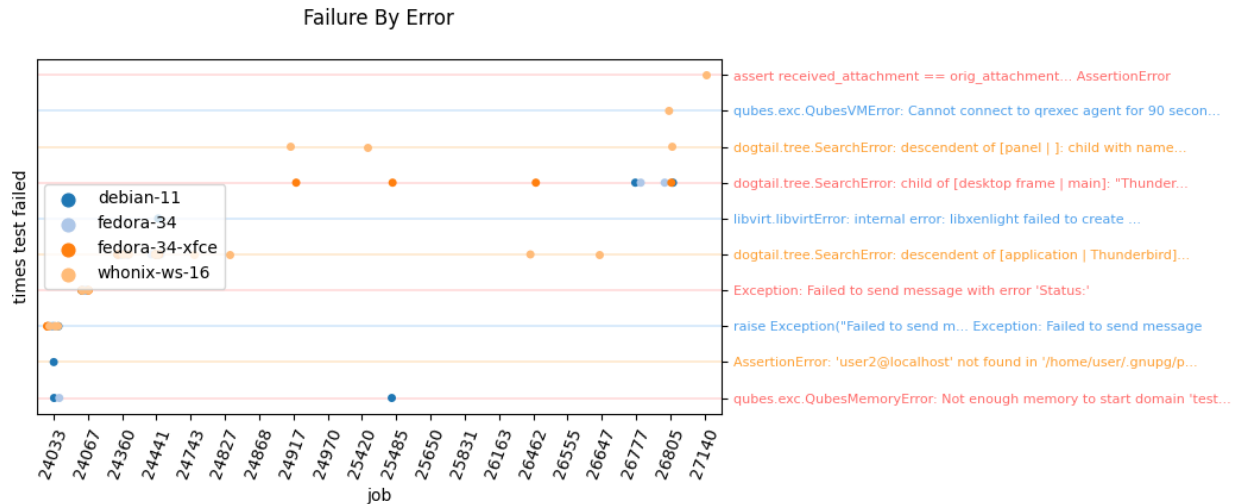


Fig. 24: openqa-investigator-splitgpg-example.png

### 1.13.32 Qube configuration interface

Qubes VM have some settings set by dom0 based on VM settings. There are multiple configuration channels, which includes:

- QubesDB
- XenStore (in Qubes 2, data the same as in QubesDB, keys without leading /)
- Qubes RPC (called at VM startup, or when configuration changed)
- GUI protocol

#### QubesDB

##### Keys exposed by dom0 to VM

- /qubes-vm-type - VM type, the same as type field in qvm-prefs. One of AppVM, ProxyVM, NetVM, TemplateVM, HVM, TemplateHVM
- /qubes-vm-updatable - flag whether VM is updatable (whether changes in root.img will survive VM restart). One of True, False
- /qubes-vm-persistence - what data do persist between VM restarts:
  - full - all disks
  - rw-only - only /rw disk
  - none - none
- /qubes-timezone - name of timezone based on dom0 timezone. For example Europe/Warsaw
- /qubes-keyboard (deprecated in R4.1) - keyboard layout based on dom0 layout. Its syntax is suitable for xkbcomp command (after expanding escape sequences like \n or \t). This is meant only as some default value, VM can ignore this option and choose its own keyboard layout (this is what keyboard setting from Qubes Manager does). This entry is created as part of gui-daemon initialization (so not available when gui-daemon disabled, or not started yet).

- `/keyboard-layout` - keyboard layout based on GuiVM layout. Its syntax can be `layout+variant+options`, `layout+variant`, `layout++options` or simply `layout`. For example, `fr+oss`, `pl++compose:caps` or `fr`. This is meant only as some default value, VM can ignore this option and choose its own keyboard layout (this is what keyboard setting from Qubes Manager does).
- `/qubes-debug-mode` - flag whether VM has debug mode enabled (`qvm-prefs` setting). One of 1, 0
- `/qubes-service/SERVICE_NAME` - subtree for VM services controlled from dom0 (using the `qvm-service` command or Qubes Manager). One of 1, 0. Note that not every service will be listed here, if entry is missing, it means “use VM default”. A list of currently supported services is in the `qvm-service` man page.
- `/qubes-netmask` - network mask (only when VM has `netvm` set); currently hardcoded “255.255.255.0”
- `/qubes-ip` - IP address for this VM (only when VM has `netvm` set)
- `/qubes-gateway` - default gateway IP (only when VM has `netvm` set); VM should add host route to this address directly via `eth0` (or whatever default interface name is)
- `/qubes-primary-dns` - primary DNS address (only when VM has `netvm` set)
- `/qubes-secondary-dns` - secondary DNS address (only when VM has `netvm` set)
- `/qubes-netvm-gateway` - same as `qubes-gateway` in connected VMs (only when VM serves as network backend - ProxyVM and NetVM)
- `/qubes-netvm-netmask` - same as `qubes-netmask` in connected VMs (only when VM serves as network backend - ProxyVM and NetVM)
- `/qubes-netvm-network` - network address (only when VM serves as network backend - ProxyVM and NetVM); can be also calculated from `qubes-netvm-gateway` and `qubes-netvm-netmask`
- `/qubes-netvm-primary-dns` - same as `qubes-primary-dns` in connected VMs (only when VM serves as network backend - ProxyVM and NetVM); traffic sent to this IP on port 53 should be redirected to primary DNS server
- `/qubes-netvm-secondary-dns` - same as `qubes-secondary-dns` in connected VMs (only when VM serves as network backend - ProxyVM and NetVM); traffic sent to this IP on port 53 should be redirected to secondary DNS server
- `/guivm-windows-prefix` - title prefix for any window not originating from another qube. This means windows of applications running in GuiVM itself

## Firewall rules in 3.x

QubesDB is also used to configure firewall in ProxyVMs. Rules are stored in separate key for each target VM. Entries:

- `/qubes-iptables` - control entry - dom0 writing `reload` here signals `qubes-firewall` service to reload rules
- `/qubes-iptables-header` - rules not related to any particular VM, should be applied before domains rules
- `/qubes-iptables-domainrules/NNN` - rules for domain NNN (arbitrary number) in `iptables-save` format. Rules are self-contained - fill `FORWARD` iptables chain and contains all required matches (source IP address etc), as well as final default action (`DROP/ACCEPT`)

VM after applying rules may signal some error, writing a message to `/qubes-iptables-error` key. This does not exclude any other way of communicating problems - like a popup.

## Firewall rules in 4.x

QubesDB is also used to configure firewall in ProxyVMs. Each rule is stored as a separate entry, grouped on target VM:

- `/qubes-firewall/SOURCE_IP` - base tree under which rules are placed. All rules there should be applied to filter traffic coming from `SOURCE_IP`. This can be either IPv4 or IPv6 address. Dom0 will do an empty write to this top level entry after finishing rules update, so VM can setup a watch here to trigger rules reload.
- `/qubes-firewall/SOURCE_IP/policy` - default action if no rule matches: **drop** or **accept**.
- `/qubes-firewall/SOURCE_IP/NNNN` - rule number `NNNN` - decimal number, padded with zeros. See below for rule format. All the rules should be applied in order of rules implied by those numbers. Note that QubesDB itself does not impose any ordering (you need to sort the rules after retrieving them). The first rule has number **0000**.

Each rule is a single QubesDB entry, consisting of pairs `key=value` separated by space. QubesDB enforces limit on a single entry length - 3072 bytes. Possible options for a single rule:

- **action**, values: **accept**, **drop**; this is present in every rule
- **dst4**, value: destination IPv4 address with a mask; for example: `192.168.0.0/24`
- **dst6**, value: destination IPv6 address with a mask; for example: `2000:::/3`
- **dsthost**, value: DNS hostname of destination host
- **proto**, values: **tcp**, **udp**, **icmp**
- **specialtarget**, value: One of predefined target, currently defined values:
  - **dns** - such option should match DNS traffic to default DNS server (but not any DNS server), on both TCP and UDP
- **dstports**, value: destination ports range separated with `-`, valid only together with **proto=tcp** or **proto=udp**; for example `1-1024`, `80-80`
- **icmptype**, value: numeric (decimal) icmp message type, for example 8 for echo request, valid only together with **proto=icmp**
- **dpi**, value: Deep Packet Inspection protocol (like: HTTP, SSL, SMB, SSH, SMTP) or the default 'NO' as no DPI, only packet filtering

Options must appear in the rule in the order listed above. Duplicated options are forbidden.

A rule matches only when all predicates match. Only one of **dst4**, **dst6** or **dsthost** can be used in a single rule.

If tool applying firewall encounters any parse error (unknown option, invalid value, duplicated option, etc), it should drop all the traffic coming from that `SOURCE_IP`, regardless of properly parsed rules.

Example valid rules:

- `action=accept dst4=8.8.8.8 proto=udp dstports=53-53`
- `action=drop dst6=2a00:1450:4000::/37 proto=tcp`
- `action=accept specialtarget=dns`
- `action=drop proto=tcp specialtarget=dns` - drop DNS queries sent using TCP
- `action=drop`

## Keys set by VM for passing info to dom0

- **memory/meminfo (xenstore)** - used memory (updated by qubes-meminfo-writer), input information for qmemman;
  - Qubes 3.x format: 6 lines (EOL encoded as \n), each in format “FIELD: VALUE kB”; fields: MemTotal, MemFree, Buffers, Cached, SwapTotal, SwapFree; meaning the same as in /proc/meminfo in Linux.
  - Qubes 4.0+ format: used memory size in the VM, in kbytes
- **/qubes-block-devices** - list of block devices exposed by this VM, each device (subdirectory) should be named in a way that VM can attach the device based on it. Each should contain these entries:
  - desc - device description (ASCII text)
  - size - device size in bytes
  - mode - default connection mode; r for read-only, w for read-write
- **/qubes-usb-devices** - list of USB devices exposed by this VM, each device (subdirectory) should contain:
  - desc - device description (ASCII text)
  - usb-ver - USB version (1, 2 or 3)

## Qubes RPC

Services called by dom0 to provide some VM configuration:

- **qubes.SetMonitorLayout** - provide list of monitors, one per line. Each line contains four numbers: width height X Y width\_mm height\_mm (physical dimensions - width\_mm and height\_mm - are optional)
- **qubes.WaitForSession** - called to wait for full VM startup
- **qubes.GetAppmenus** - receive appmenus from given VM (template); TODO: describe format here
- **qubes.GetImageRGBA** - receive image/application icon. Protocol:
  1. Caller sends name of requested icon. This can be one of:
    - **xdgicon:NAME** - search for NAME in standard icons theme
    - - - get icon data from stdin (the caller), can be prefixed with format name, for example **png:-**
    - file name
  2. The service responds with image dimensions: width and height as decimal numbers, separated with space and with EOL marker at the and; then image data in RGBA format (32 bits per pixel)
- **qubes.SetDateTime** - set VM time, called periodically by dom0 (can be triggered manually from dom0 by calling **qvm-sync-clock**). The service receives one line at stdin - time in format of **date -u -Iseconds**, for example **2015-07-31T16:10:43+0000**.
- **qubes.SetGuiMode** - called in HVM to switch between fullscreen and seamless GUI mode. The service receives a single word on stdin - either **FULLSCREEN** or **SEAMLESS**
- **qubes.ResizeDisk** - called to inform that underlying disk was resized. Name of disk image is passed on standard input (**root**, **private**, **volatile**, or other). This is used starting with Qubes 4.0.

Other Qrexec services installed by default:

- **qubes.Backup** - store Qubes backup. The service receives location chosen by the user (one line, terminated by \n), the backup archive (*description of backup format*)

- `qubes.DetachPciDevice` - service called in reaction to `qvm-pci -d` call on running VM. The service receives one word - BDF of device to detach. When the service call ends, the device will be detached
- `qubes.Filecopy` - receive some files from other VM. Files sent in *qfile format*
- `qubes.OpenInVM` - open a file in called VM. Service receives a single file on stdin (in *qfile format*). After a file viewer/editor is terminated, if the file was modified, can be sent back (just raw content, without any headers); otherwise service should just terminate without sending anything. This service is used by both `qvm-open-in-vm` and `qvm-open-in-dvm` tools. When called in DispVM, service termination will trigger DispVM cleanup.
- `qubes.Restore` - retrieve Qubes backup. The service receives backup location entered by the user (one line, terminated by `\n`), then should output backup archive in *qfile format* (core-agent-linux component contains `tar2qfile` utility to do the conversion)
- `qubes.SelectDirectory`, `qubes.SelectFile` - services which should show file/directory selection dialog and return (to stdout) a single line containing selected path, or nothing in the case of cancellation
- `qubes.SuspendPre` - service called in every VM with PCI device attached just before system suspend
- `qubes.SuspendPost` - service called in every VM with PCI device attached just after system resume
- `qubes.SyncNtpClock` - service called to trigger network time synchronization. Service should synchronize local VM time and terminate when done.
- `qubes.WindowIconUpdater` - service called by VM to send icons of individual windows. The protocol there is simple one direction stream: VM sends window ID followed by icon in `qubes.GetImageRGBA` format, then next window ID etc. VM can send icon for the same window multiple times to replace previous one (for example for animated icons)
- `qubes.VMShell` - call any command in the VM; the command(s) is passed one per line
  - `qubes.VMShell+WaitForSession` waits for full VM startup first
- `qubes.VMExec` - call any command in the VM, without using shell, the command needs to be passed as argument and encoded as follows:
  - the executable name and arguments are separated by `+`
  - everything except alphanumeric characters, `.` and `_` needs to be escaped
  - bytes are escaped as `-HH` (where HH is hex code, capital letters only)
  - `-` itself can be escaped as `--`
  - example: to run `ls -a /home/user`, use `qubes.VMExec+ls+--a+-2Fhome-2Fuser`
- `qubes.VMExecGUI` - a variant of `qubes.VMExec` that waits for full VM startup first

Services called in GuiVM:

- `policy.Ask`, `policy.Notify` - confirmation prompt and notifications for Qubes RPC calls, see *qrexec-policy implementation* for a detailed description.

Currently Qubes still calls few tools in VM directly, not using service abstraction. This will change in the future. Those tools are:

- `/usr/lib/qubes/qubes-download-dom0-updates.sh` - script to download updates (or new packages to be installed) for dom0 (`qubes-dom0-update` tool)
- `date -u -Iseconds` - called directly to retrieve time after calling `qubes.SyncNtpClock` service (`qvm-sync-clock` tool)
- `nm-online -x` - called before `qubes.SyncNtpClock` service call by `qvm-sync-clock` tool
- `resize2fs` - called to resize filesystem on `/rw` partition by `qvm-grow-private` tool

- `gpk-update-viewer` - called by Qubes Manager to display available updates in a TemplateVM
- `systemctl start qubes-update-check.timer` (and similarly `stop`) - called when enabling/disabling updates checking in given VM (`qubes-update-check` *qvm-service*)

Additionally, automatic tests extensively run various commands directly in VMs. We do not plan to change that.

## GUI protocol

GUI initialization includes passing the whole screen dimensions from dom0 to VM. This will most likely be overwritten by `qubes.SetMonitorLayout` Qubes RPC call.

### 1.13.33 Windows debugging

Debugging Windows code can be tricky in a virtualized environment. The guide below assumes Xen hypervisor and Windows 7 VMs.

User-mode debugging is usually straightforward if it can be done on one machine. Just duplicate your normal debugging environment in the VM.

Things get complicated if you need to perform kernel debugging or troubleshoot problems that only manifest on system boot, user logoff or similar. For that you need two Windows VMs: the *host* and the *target*. The *host* will contain WinDbg installation, your source code and private symbols. The *target* will run the code being debugged. Both will be linked by virtual serial ports.

- First, you need to prepare separate copies of both *target* and *host* VM configuration files with some changes. Copy the files from `/var/lib/qubes/appvms/vmname/vmname.conf` to some convenient location, let's call them **host.conf** and **target.conf**.
- In both copied files add the following line at the end: `serial = 'pty'`. This will make Xen connect VM's serial ports to dom0's ptys.
- From now on you need to start both VMs like this: `qvm-start --custom-config=/your/edited/host.conf host`
- To connect both VM serial ports together you will either need `socat` or a custom utility described later.
- To determine which dom0 pty corresponds to VM's serial port you need to read xenstore, example script below:

```
#!/bin/sh

id1=$(xl domid "$1-dm")
tty1=$(xenstore-read /local/domain/${id1}/device/console/3/tty)
echo $tty1
```

Pass it a running VM name and it will output the corresponding pty name.

- To connect both ptys you can use `socat` like that:

```
#!/bin/sh

id1=$(xl domid "$1-dm")
id2=$(xl domid "$2-dm")
tty1=$(xenstore-read /local/domain/${id1}/device/console/3/tty)
tty2=$(xenstore-read /local/domain/${id2}/device/console/3/tty)
socat $tty1,raw $tty2,raw
```



...but there is a catch. Xen seems to process the traffic that goes through serial ports and changes all **0x0a** bytes into **0x0d, 0x0a** pairs (newline conversion). I didn't find a way to turn that off (setting ptys to raw mode didn't change anything) and it's not mentioned anywhere on the Internet, so maybe it's something on my system. If the above script works for you then you don't need anything more in dom0.

- On the *target* system, run `bcdedit /set debug on` on the console to turn on kernel debugging. It defaults to the first serial port.
- On the *host* system, install [WinDbg](#) and start the kernel debug (Ctrl-K), choose **com1** as the debug port.
- Reboot the *target* VM.
- Run the above shell script in dom0.
- If everything is fine you should see the proper kernel debugging output in WinDbg. However, if you see something like that:

```
Opened \\.\com1
Waiting to reconnect...
Connected to Windows 7 7601 x64 target at (Wed Mar 19 20:35:43.262 2014 (UTC +1:00)), ptr64 TRUE
Kernel Debugger connection established.
Symbol search path is: srv*c:\symbols*https://msdl.microsoft.com/download/symbols
Executable search path is:
... Retry sending the same data packet for 64 times.
The transport connection between host kernel debugger and target Windows seems lost.
please try resync with target, recycle the host debugger, or reboot the target.
Windows.
Unable to read KTHREAD address fffff8000281ccc0
*****
Unable to read debugger data block header
*****
Unable to read KTHREAD address fffff8000281ccc0
Unable to read PsLoadedModuleList
Unable to read KTHREAD address fffff8000281ccc0
*****
Unable to read debugger data block header
*****
```

...then you're most likely a victim of the CRLF issue mentioned above. To get around it I wrote a small utility that basically does what socat would do and additionally corrects those replaced bytes in the stream. It's not pretty but it works:

```
#include <errno.h>
#include <stdio.h>
#include <fcntl.h>
#include <termios.h>

int fd1, fd2;
char mark = ' ';

void out(unsigned char c)
{
    static int count = 0;
    static unsigned char buf[17] = {0};
```

(continues on next page)

(continued from previous page)

```

// relay to ouputput port
write(fd2, &c, 1);
fprintf(stderr, "%c", mark);

/* dump all data going over the line
if (count == 0)
    fprintf(stderr, "%c", mark);
fprintf(stderr, "%02x ", c);
if (c >= 0x20 && c < 0x80)
    buf[count] = c;
else
    buf[count] = '.';
count++;
if (count == 0x10)
{
    count = 0;
    fprintf(stderr, " %s\n", buf);
}
*/
}

int main(int argc, char* argv[])
{
    unsigned char c = 0;
    struct termios tio;
    ssize_t size;

    if (argc < 3)
    {
        fprintf(stderr, "Usage: %s pty1 pty2 [mark character]\n", argv[0]);
        return EINVAL;
    }

    fd1 = open(argv[1], O_RDONLY | O_NOCTTY);
    if (fd1 <= 0)
    {
        perror("open fd1");
        return errno;
    }
    fd2 = open(argv[2], O_WRONLY | O_NOCTTY);
    if (fd2 <= 0)
    {
        perror("open fd2");
        return errno;
    }

    /*
    // This doesn't make any difference which supports the theory
    // that it's Xen who corrupts the byte stream.
    cfmakeraw(&tio);
    if (tcsetattr(fd1, TCSANOW, &tio) < 0)
    {
        perror("tcsetattr 1");
    }
    */

```

(continues on next page)

(continued from previous page)

```

        return errno;
    }
    if (tcsetattr(fd2, TCSANOW, &tio) < 0)
    {
        perror("tcsetattr 2");
        return errno;
    }
    */
    if (argc == 4)
        mark = argv[3][0];

    while (1)
    {
        size = read(fd1, &c, 1);
        if (size <= 0)
            break;

    parse:
        if (c == 0x0d)
        {
            size = read(fd1, &c, 1);
            if (size <= 0)
            {
                out(0x0d);
                break;
            }
            if (c == 0x0a)
            {
                out(0x0a);
            }
            else
            {
                out(0x0d);
                goto parse;
            }
        }
        else
            out(c);
    }

    close(fd1);
    close(fd2);
    return 0;
}

```

This utility is a unidirectional relay so you need to run two instances to get duplex communication, like:

```

.. code:: bash

    #!/bin/sh

```

(continues on next page)

(continued from previous page)

```
id1=$(xl domid "$1-dm")
id2=$(xl domid "$2-dm")
tty1=$(xenstore-read /local/domain/${id1}/device/console/3/tty)
tty2=$(xenstore-read /local/domain/${id2}/device/console/3/tty)
./ptycrlf ${tty1} ${tty2} - &
./ptycrlf ${tty2} ${tty1} + &
```

With this everything should be good:

```
.. code:: bash
```

```
Opened \\.\com1
Waiting to reconnect...
Connected to Windows 7 7601 x64 target at (Wed Mar 19 20:56:31.371 2014 (UTC +1:00)), ptr64 TRUE
Kernel Debugger connection established.
Symbol search path is: srv*c:\symbols*https://msdl.microsoft.com/download/symbols
Executable search path is:
Windows 7 Kernel Version 7601 MP (1 procs) Free x64
Built by: 7601.18247.amd64fre.win7sp1_gdr.130828-1532
Machine Name:
Kernel base = 0xfffff800`0261a000 PsLoadedModuleList = 0xfffff800`0285d6d0
System Uptime: not available
```

## Debugging HVMs in the Qubes R4.0

There are two main issues to be adopted to get all things to work in the R4.0.

### Add a virtual serial port

Qemu in the stub domain with virtual serial port added in a recommended way (`<serial type="pty"/>`) fails to start (Could not open '/dev/hvc1': No such device). It seems like a lack of multiple xen consoles support/configuration. The only way that I have found is to attach serial port explicitly to the available console.

1. Unpack stub domain in dom0:

```
$ mkdir stubroot
$ cp /usr/lib/xen/boot/stubdom-linux-rootfs stubroot/stubdom-linux-rootfs.gz
$ cd stubroot
$ gunzip stubdom-linux-rootfs.gz
$ cpio -i -d -H newc --no-absolute-filenames < stubdom-linux-rootfs
$ rm stubdom-linux-rootfs
```

2. Edit Init script to remove last loop and to add “-serial /dev/hvc0” to the qemu command line.

3. Apply changes:

```
$ find . -print0 | cpio --null -ov --format=newc | gzip -9 > ../stubdom-linux-rootfs
$ sudo mv ../stubdom-linux-rootfs /usr/lib/xen/boot
```

## Connect two consoles

Run the following script:

```
debugname1=win7new
debugname2=win7dbg
id1=$(xl domid "$debugname1-dm")
id2=$(xl domid "$debugname2-dm")

tty1=$(xenstore-read /local/domain/${id1}/console/tty)
tty2=$(xenstore-read /local/domain/${id1}/console/tty)

socat $tty1,raw $tty2,raw
```

Happy debugging!

### 1.13.34 Safe remote dom0 terminals

If you do not have working graphics in Dom0, then using a terminal can be quite annoying! This was the case for the author while trying to debug PCI-passthrough of a machine's primary (only) GPU.

Your first thought might be to just allow network access to Dom0, enable ssh, and connect in remotely. But, this gravely violates the Qubes security model.

Instead, a better solution is to split the input and output paths of using a terminal. Use your normal keyboard for input, but have the output go to a remote machine in a unidirectional manner.

To do this, we make use of `script(1)`, `qvm-run`, and optionally your network transport of choice.

#### To a different VM

As an example of forwarding terminal output to another VM on the same machine:

```
$ mkfifo /tmp/foo
$ qvm-run -p some-vm 'xterm -e "cat 0<&5" 5<&0' </tmp/foo >/dev/null 2>&1 &
$ script -f /tmp/foo
```

#### To a different machine

In this case over SSH (from a network-connected VM):

```
$ mkfifo /tmp/foo
$ qvm-run -p some-vm \
  'ssh user@host sh -c "DISPLAY=:0 xterm -e \"cat 0<&5\" 5<&0\" \" \
    </tmp/foo >/dev/null 2>&1 &
$ script -f /tmp/foo
```

Note that no data received over SSH is ever treated as terminal input in Dom0. The input path remains only from your trusted local keyboard.

## Multiple terminals

For multiple terminals, you may find it easier to just use tmux than to try to blindly switch to the correct window.

## Terminal size

It is up to you to ensure the sizes of the local and remote terminal are the same, otherwise things may display incorrectly (especially in interactive programs). Depending on your shell, the size of your local (blind) terminal is likely stored in the `$LINES` and `$COLUMNS` variables.

```
$ echo $COLUMNS $LINES
80 24
```

## A note on serial consoles

If your machine has a serial console, you may wish to use that, but note that a similar split-I/O model should be used to ensure Dom0 integrity. If you use the serial console as normal (via e.g. `getty` on `ttyX`, and logging in as normal), then the machine at the end of the serial cable could compromise your machine! Ideally, you would take input from your trusted keyboard, and only send the output over the serial cable via e.g. disabling `getty` and using:

```
script -f /dev/ttyS0
```

You don't even need to connect the TX pin.

### 1.13.35 How to mount LVM images

You want to read your LVM image (e.g., there is a problem where you can't start any VMs except dom0).

1: make the image available for qubesdb. From dom0 terminal:

```
# Example: /dev/qubes_dom0/vm-debian-9-tmp-root
[user@dom0]$ dev=$(basename $(readlink /dev/YOUR_LVM_VG/YOUR_LVM_IMAGE))
[user@dom0]$ qubesdb-write /qubes-block-devices/$dev/desc "YOUR_LVM_IMAGE"
```

2: Create a new disposable VM

```
[user@dom0]$ qvm-run -v --dispvm=YOUR_DVM_TEMPLATE --service qubes.StartApp+xterm &
```

3: Attach the device to your newly created disp VM

From the GUI, or from the command line:

```
[user@dom0]$ qvm-block attach NEWLY_CREATED_DISPVM dom0:$dev
```

4: Mount the partition you want to, and do what you want with it

```
[user@dispXXXX]$ mount /dev/xvdiX /mnt/
```

5: Umount and kill the VM

```
[user@dispXXXX]$ umount /mnt/
```

6: Remove the image from qubesdb

```
[user@dom0]$ qubesdb-rm /qubes-block-devices/$dev/
```

## References

Please consult this issue's [comment](#).

### 1.13.36 Qubes builder

**Note:** See *ISO building instructions* for a streamlined overview on how to use the build system.

We have a fully automated build system for Qubes, that downloads, builds and packages all the Qubes components, and finally should spit out a ready-to-use installation ISO, all in a [secure](#) way.

In order to use it, you should use an rpm-based distro, like Fedora :), and should ensure the following packages are installed:

- sudo
- gnupg
- git
- createrepo
- rpm-build
- dnf-plugins-core
- make
- wget
- rpmdevtools
- python3-sh
- dialog
- rpm-sign
- dpkg-dev
- debootstrap
- python3-pyyaml
- devscripts
- perl-Digest-MD5
- perl-Digest-SHA

Usually you can install those packages by just issuing:

```
sudo dnf install gnupg git createrepo rpm-build make wget rpmdevtools python3-sh dialog_
↪rpm-sign dpkg-dev debootstrap python3-pyyaml devscripts perl-Digest-MD5 perl-Digest-SHA
```

The build system creates build environments in chroots and so no other packages are needed on the host. All files created by the build system are contained within the qubes-builder directory. The full build requires some 25GB of free space, so keep that in mind when deciding where to place this directory.

The build system is configured via builder.conf file. You can use the setup.sh script to create and modify this file. Alternatively, you can copy the provided default builder.conf, and modify it as needed, e.g.:

```
cp example-configs/qubes-os-master.conf builder.conf
# edit the builder.conf file and set the following variables:
NO_SIGN=1
```

One additional useful requirement is that ‘sudo root’ must work without any prompt, which is default on most distros (e.g. ‘sudo bash’ brings you the root shell without asking for any password). This is important as the builder needs to switch to root and then back to user several times during the build process.

Additionally, if building with signing enabled (NO\_SIGN is not set), you must adjust `~/ .rpmmacros` file so that it points to the GPG key used for package signing, e.g.:

```
%_signature gpg
%_gpg_path /home/user/.gnupg
%_gpg_name AC1BF9B3 # <-- Key ID used for signing
```

It is also recommended that you use an empty passphrase for the private key used for signing. Contrary to a popular belief, this doesn’t affect your key or sources security – if somebody compromised your system, then the game is over anyway, whether you have used an additional passphrase for the key or not.

So, to build Qubes you would do:

```
# Import the Qubes master key
gpg --recv-keys 0xDDFA1A3E36879494

# Verify its fingerprint, set as 'trusted'.
# This is described here:
# https://www.qubes-os.org/doc/VerifyingSignatures

wget https://keys.qubes-os.org/keys/qubes-developers-keys.asc
gpg --import qubes-developers-keys.asc

git clone https://github.com/QubesOS/qubes-builder.git qubes-builder
cd qubes-builder

# Verify its integrity:
git tag -v `git describe`

cp example-configs/qubes-os-master.conf builder.conf
# edit the builder.conf file and set the following variables:
# NO_SIGN="1"

# Download all components:

make get-sources

# And now to build all Qubes RPMs (this will take a few hours):

make qubes

# ... and then to build the ISO

make iso
```

And this should produce a shiny new ISO.



You can also build selected component separately. Eg. to compile only gui virtualization agent/daemon:

```
make gui-daemon
```

You can get a full list from make help.

## Making customized build

### Manual source modification

You can also modify sources somehow if you wish. Here are some basic steps:

1. Download qubes-builder as described above (if you want to use marmarek's branches, you should also download qubes-builder from his repo - replace 'QubesOS' with 'marmarek' in above git clone command)
2. Edit builder.conf (still the same as above), some useful additions:
  - You can also set GIT\_PREFIX="marmarek/qubes-" to use marmarek's repo instead of "mainstream" - it contains newer (but less tested) versions
3. Download unmodified sources

```
make get-sources
```

4. **Make your modifications here**
5. Build the Qubes `make qubes` actually is just meta target which builds all required components in correct order. The list of components is configured in `builder.conf`. You can also check the current value at the end of `make help`, or using `make build-info`.
6. `get-sources` is already done, so continue with the next one. You can skip `sign-all` if you've disabled signing

```
make vmm-xen core-admin linux-kernel gui-daemon template desktop-linux-kde_
↪ installer-qubes-os manager linux-dom0-updates
```

7. build iso installation image

```
make iso
```

## Use pre-built Qubes packages

For building just a few selected packages, it's very useful to download pre-built qubes-specific dependencies from `{yum,deb}.qubes-os.org`. This is especially true for `gcc`, which takes several hours to build.

Before creating the `chroot`, add this to your `builder.conf`:

```
USE_QUBES_REPO_VERSION = $(RELEASE)
```

It will add the 'current' Qubes repository to your `chroot` environment. Next, specify which components (`gcc`, for example) you want to download instead of compiling:

```
COMPONENTS := $(filter-out gcc,$(COMPONENTS))
```

Alternatively, edit the actual `COMPONENTS` list which is defined in the included version-dependent config from `example-configs` (see series of `include` directives near the beginning of `builder.conf`). This way, you can build only the packages in which you are interested.

If you also want to use the ‘current-testing’ repository, add this to your configuration:

```
USE_QUBES_REPO_TESTING = 1
```

In the case of an existing *chroot*, for mock-enabled builds, this works immediately because *chroot* is constructed each time separately. For legacy builds, it will not add the necessary configuration into the build environment unless a specific builder change or configuration would force rebuilding *chroot*.

Also, once enabled, disabling this setting will not disable repositories in relevant *chroots*. And even if it did, there could be some leftover packages installed from those repos (which may or may not be desirable).

**Note** If you are building Ubuntu templates, you cannot use this option. This is because Qubes does not provide official packages for Ubuntu templates.

## Code verification keys management

*QubesBuilder* by default verifies signed tags on every downloaded code. Public keys used for that are stored in *keyrings/git*. By default Qubes developers’ keys are imported automatically, but if you need some additional keys (for example your own), you can add them using:

```
GNUPGHOME=$PWD/keyrings/git gpg --import /path/to/key.asc
GNUPGHOME=$PWD/keyrings/git gpg --edit-key ID_OF_JUST_IMPORTED_KEY
# here use "trust" command to set key fully or ultimately trusted - only those keys are
↪accepted by QubesBuilder
```

All Qubes developers’ keys are signed by the Qubes Master Signing Key (which is set as ultimately trusted key), so are trusted automatically.

If you are the owner of Master key and want to revoke such signature, use the *revsig gpg key edit* command and update the key in *qubes-developers-keys.asc* - now the key will be no longer trusted (unless manually set as such).

## Further information

For advanced *QubesBuilder* use, and preparing sources, take a look at *QubesBuilderDetails* page, or *QubesBuilder’s* *doc* directory.

## 1.13.37 Qubes builder details

### Components *Makefile.builder* file

*QubesBuilder* expects that each component have *Makefile.builder* file in its root directory. This file specifies what should be done to build the package. As name suggests, this is normal makefile, which is included by builder as its configuration. Its main purpose is to set some variables. Generally all available variables/settings are described as comments at the beginning of *Makefile.\** in *QubesBuilder*.

Variables for Linux build:

- **RPM\_SPEC\_FILES** List (space separated) of spec files for RPM package build. Path should be relative to component root directory. *QubesBuilder* will install all *BuildRequires* (in *chroot* environment) before the build. In most Qubes components all spec files are kept in *rpm\_spec* directory. This is mainly used for Fedora packages build.
- **ARCH\_BUILD\_DIRS** List (space separated) of directories with *PKGBUILD* files for Archlinux package build. Similar to RPM build, *QubesBuilder* will install all *makedepends*, then build the package.

Most components uses *archlinux* directory for this purpose, so its good to keep this style.

Variables for Windows build:

- **WIN\_COMPILER** Choose which compiler should be used for this component, thus which build scripts. Currently two options available:
  - **WDK** - Windows Driver Kit (default). Command used to build: *build -cZg*.
  - **mingw** - MinGW (Windows gcc port). Command used to build: *make all*
- **WIN\_SOURCE\_SUBDIRS** List of directories in which above command should be run. In most cases it will be only one entry: current directory (.).
- **WIN\_PREBUILD\_CMD** Command to run before build, mostly useful for WDK build (in mingw case, you can use makefile for this purpose). Can be used to set some variables, preprocess some files etc.
- **WIN\_SIGN\_CMD** Command used to sign resulting binaries. Note that default value is *sign.bat*. If you don't want to sign binaries, specify some placeholder here (eg. *true*). Check existing components (eg. *vmm-xen-windows-pvdrivers*) for example scripts. This command will be run with certain environment variables:
  - **CERT\_FILENAME** Path to key file (pfx format)
  - **CERT\_PASSWORD** Key password
  - **CERT\_PUBLIC\_FILENAME** Certificate path in the case of self-signed cert
  - **CERT\_CROSS\_CERT\_FILENAME** Certificate path in the case of correct authenticode cert
  - **SIGNTOOL** Path to signtool
- **WIN\_PACKAGE\_CMD** Command used to produce installation package (msi or msm). Default value is *wix.bat*, similar to above - use *true* if you don't want this command.
- **WIN\_OUTPUT\_HEADERS** Directory (relative to **WIN\_SOURCE\_SUBDIRS** element) with public headers of the package - for use in other components.
- **WIN\_OUTPUT\_LIBS** Directory (relative to **WIN\_SOURCE\_SUBDIRS** element) with libraries (both DLL and implib) of the package - for use in other components. Note that *QubesBuilder* will copy files specified as *\$(WIN\_OUTPUT\_LIBS)/!\*\** to match WDK directory layout (*<specified directory>/<arch directory>/<actual libraries>*), so you in mingw build you need to place libraries in some additional subdirectory.
- **WIN\_BUILD\_DEPS** List of components required to build this one. *QubesBuilder* will copy files specified with **WIN\_OUTPUT\_HEADERS** and **WIN\_OUTPUT\_LIBS** of those components to some directory and provide its path with **QUBES\_INCLUDES** and **QUBES\_LIBS** variables. Use those variables in your build scripts (*sources* or *Makefile* - depending on selected compiler). You can assume that the variables are always set and directories always exists, even if empty.

## builder.conf settings

Most settings are documented in *builder.conf.default* file, which can be used as template the actual configuration.

**TODO**

## Notes

- For a list of custom TemplateVMs available in QubesBuilder look at [Supported Versions page](#).

### 1.13.38 Development workflow

A workflow for developing Qubes OS+

First things first, setup [QubesBuilder](#). This guide assumes you're using qubes-builder to build Qubes.

#### Repositories and committing Code

Qubes is split into a bunch of git repos. These are all contained in the `qubes-src` directory under `qubes-builder`. Subdirectories there are separate components, stored in separate git repositories.

The best way to write and contribute code is to create a git repo somewhere (e.g., github) for the repo you are interested in editing (e.g., `qubes-manager`, `core-agent-linux`, etc). To integrate your repo with the rest of Qubes, `cd` to the repo directory and add your repository as a remote in git

#### Example:

```
$ cd qubes-builder/qubes-src/qubes-manager
$ git remote add abel git@github.com:abeluck/qubes-manager.git
```

You can then proceed to easily develop in your own branches, pull in new commits from the dev branches, merge them, and eventually push to your own repo on github.

When you are ready to submit your changes to Qubes to be merged, push your changes, then create a signed git tag (using `git tag -s`). Finally, send a letter to the Qubes listserv describing the changes and including the link to your repository. You can also create pull request on github. Don't forget to include your public PGP key you use to sign your tags.

#### Kernel-specific notes

##### Prepare fresh version of kernel sources, with Qubes-specific patches applied

In `qubes-builder/qubes-src/linux-kernel`:

```
make prep
```

The resulting tree will be in `kernel-<VERSION>/linux-<VERSION>`:

```
ls -ltrd kernel*/linux*
```

```
drwxr-xr-x 23 user user 4096 Nov  5 09:50 kernel-3.4.18/linux-3.4.18
drwxr-xr-x  6 user user 4096 Nov 21 20:48 kernel-3.4.18/linux-obj
```

## Go to the kernel tree and update the version

In qubes-builder/qubes-src/linux-kernel:

```
cd kernel-3.4.18/linux-3.4.18
```

## Changing the config

In kernel-3.4.18/linux-3.4.18:

```
cp ../../config .config
make oldconfig
```

Now change the configuration. For example, in kernel-3.4.18/linux-3.4.18:

```
make menuconfig
```

Copy the modified config back into the kernel tree:

```
cp .config ../../../../config
```

## Patching the code

TODO: describe the workflow for patching the code, below are some random notes, not working well

```
ln -s ../../patches.xen
export QUILT_PATCHES=patches.xen
export QUILT_REFRESH_ARGS="-p ab --no-timestamps --no-index"
export QUILT_SERIES=../../series-pvops.conf

quilt new patches.xen/pvops-3.4-0101-usb-xen-pvusb-driver-bugfix.patch
quilt add drivers/usb/host/Kconfig drivers/usb/host/Makefile \
        drivers/usb/host/xen-usbback/* drivers/usb/host/xen-usbfront.c \
        include/xen/interface/io/usbif.h

*edit something*

quilt refresh
cd ../../
vi series.conf
```

## Building RPMs

TODO: Is this step generic for all subsystems?

Now it is a good moment to make sure you have changed kernel release name in rel file. For example, if you change it to '1debug20121116c' the resulting RPMs will be named 'kernel-3.4.18-1debug20121116c.pvops.qubes.x86\_64.rpm'. This will help distinguish between different versions of the same package.

You might want to take a moment here to review (git diff, git status), commit your changes locally.

To actually build RPMs, in qubes-builder:

```
make linux-kernel
```

RPMs will appear in `qubes-src/linux-kernel/pkgs/fc20/x86_64`:

```
-rw-rw-r-- 1 user user 42996126 Nov 17 04:08 kernel-3.4.18-1debug20121116c.pvops.qubes.  
↳x86_64.rpm  
-rw-rw-r-- 1 user user 43001450 Nov 17 05:36 kernel-3.4.18-1debug20121117a.pvops.qubes.  
↳x86_64.rpm  
-rw-rw-r-- 1 user user 8940138 Nov 17 04:08 kernel-devel-3.4.18-1debug20121116c.pvops.  
↳qubes.x86_64.rpm  
-rw-rw-r-- 1 user user 8937818 Nov 17 05:36 kernel-devel-3.4.18-1debug20121117a.pvops.  
↳qubes.x86_64.rpm  
-rw-rw-r-- 1 user user 54490741 Nov 17 04:08 kernel-qubes-vm-3.4.18-1debug20121116c.  
↳pvops.qubes.x86_64.rpm  
-rw-rw-r-- 1 user user 54502117 Nov 17 05:37 kernel-qubes-vm-3.4.18-1debug20121117a.  
↳pvops.qubes.x86_64.rpm
```

### Useful QubesBuilder commands

1. `make check` - will check if all the code was committed into repository and if all repository are tagged with signed tag.
2. `make show-vtags` - show version of each component (based on git tags) - mostly useful just before building ISO. **Note:** this will not show version for components containing changes since last version tag.
3. `make push` - push change from **all** repositories to git server. You must set proper remotes (see above) for all repositories first.
4. `make prepare-merge` - fetch changes from remote repositories (can be specified on commandline via `GIT_SUBDIR` or `GIT_REMOTE` vars), (optionally) verify tags and show the changes. This do not merge the changes - there are left for review as `FETCH_HEAD` ref. You can merge them using `git merge FETCH_HEAD` (in each repo directory). Or `make do-merge` to merge all of them.

### Copying Code to dom0

When developing it is convenient to be able to rapidly test changes. Assuming you're developing Qubes on Qubes, you should be working in a special VM for Qubes and occasionally you will want to transfer code or RPMs back to dom0 for testing.

Here are some handy scripts Marek has shared to facilitate this.

You may also like to run your *test environment on separate machine*.

### Syncing dom0 files

TODO: edit this script to be more generic

```
#!/bin/sh  
  
set -x  
set -e
```

(continues on next page)

(continued from previous page)

```

QUBES_PY_DIR=/usr/lib64/python2.6/site-packages/qubes
QUBES_PY=$QUBES_PY_DIR/qubes.py
QUBESUTILS_PY=$QUBES_PY_DIR/qubesutils.py

qvm-run -p qubes-devel 'cd qubes-builder/qubes-src/core/dom0; tar c qmemman/qmemman*.py \
↳ qvm-core/*.py qvm-tools/* misc/vm-template-hvm.conf misc/qubes-start.desktop ../misc/ \
↳ block-snapshot aux-tools ../qrexec' |tar xv
cp $QUBES_PY qubes.py.bak$$
cp $QUBESUTILS_PY qubesutils.py.bak$$
cp /etc/xen/scripts/block-snapshot block-snapshot.bak$$
sudo cp qvm-core/qubes.py $QUBES_PY
sudo cp qvm-core/qubesutils.py $QUBESUTILS_PY
sudo cp qvm-core/guihelpers.py $QUBES_PY_DIR/
sudo cp qmemman/qmemman*.py $QUBES_PY_DIR/
sudo cp misc/vm-template-hvm.conf /usr/share/qubes/
sudo cp misc/qubes-start.desktop /usr/share/qubes/
sudo cp misc/block-snapshot /etc/xen/scripts/
sudo cp aux-tools/qubes-dom0-updates.cron /etc/cron.daily/
# FIXME(Abel Luck): I hope to

```

## Apply qvm-tools

TODO: make it more generic

```

#!/bin/sh

BAK=qvm-tools.bak$$
mkdir -p $BAK
cp -a /usr/bin/qvm-* /usr/bin/qubes-* $BAK/
sudo cp qvm-tools/qvm-* qvm-tools/qubes-* /usr/bin/

```

## Copy from dom0 to an appvm

```

#!/bin/sh
#
# usage ./cp-domain <vm_name> <file_to_copy>
#
domain=$1
file=$2
fname=`basename $file`

qvm-run $domain 'mkdir /home/user/incoming/dom0 -p'
cat $file| qvm-run --pass-io $domain "cat > /home/user/incoming/dom0/$fname"

```

## Git connection between VMs

Sometimes it's useful to transfer git commits between VMs. You can use `git format-patch` for that and simply copy the files. But you can also setup custom qrexec service for it.

Below example assumes that you use `builder-RX` directory in target VM to store sources in qubes-builder layout (where X is some number). Make sure that all the scripts are executable.

Service file (save in `/usr/local/etc/qubes-rpc/local.Git` in target VM):

```
#!/bin/sh

exec 2>/tmp/log2

read service rel repo
echo "Params: $service $rel $repo" >&2
# Adjust regexps if needed
echo "$repo" | grep -q '^[A-Za-z0-9-]\+$' || exit 1
echo "$rel" | grep -q '^[0-9-]\+$' || exit 1
path="/home/user/builder-R$rel/qubes-src/$repo"
if [ "$repo" = "builder" ]; then
    path="/home/user/builder-R$rel"
fi
case $service in
    git-receive-pack|git-upload-pack)
        echo "starting $service $path" >&2
        exec $service $path
        ;;
    *)
        echo "Unsupported service: $service" >&2
        ;;
esac
```

Client script (save in `~/bin/git-qrexec` in source VM):

```
#!/bin/sh

VMNAME=$1

(echo $GIT_EXT_SERVICE $2 $3; exec cat) | qrexec-client-vm $VMNAME local.Git
```

You will also need to setup qrexec policy in dom0 (`/etc/qubes-rpc/policy/local.Git`).

Usage:

```
[user@source core-agent-linux]$ git remote add testbuilder "ext::git-qrexec testbuilder.
↪3 core-agent-linux"
[user@source core-agent-linux]$ git push testbuilder master
```

You can create `~/bin/add-remote` script to ease adding remotes:

```
#!/bin/sh

[ -n "$1" ] || exit 1
```

(continues on next page)



(continued from previous page)

```

if [ "$1" = "tb" ]; then
    git remote add $1 "ext::git-qrexec testbuilder 3 `basename $PWD`"
    exit $?
fi

git remote add $1 git@github.com:$1/qubes-`basename $PWD`

```

It should be executed from component top level directory. This script takes one argument - remote name. If it is `tb`, then it creates qrexec-based git remote to testbuilder VM. Otherwise it creates remote pointing at github account of the same name. In any case it points at repository matching current directory name.

### Sending packages to different VM

Other useful script(s) can be used to setup local package repository hosted in some VM. This way you can keep your development VM behind firewall, while having an option to expose some yum/apt repository to the local network (to have them installed on test machine).

To achieve this goal, a dummy repository can be created, which instead of populating metadata locally, will upload the packages to some other VM and trigger repository update there (using qrexec). You can use unstable repository flavor, because there is no release managing rules bundled (unlike current and current-testing).

### RPM packages - yum repo

In source VM, grab `linux-yum` repository (below is assumed you've made it in `~/repo-yum-upload` directory) and replace `update_repo.sh` script with:

```

#!/bin/sh

VMNAME=repo-vm

set -e
qvm-copy-to-vm $VMNAME $1
# remove only files, leave directory structure
find -type f -name '*.rpm' -delete
# trigger repo update
qrexec-client-vm $VMNAME local.UpdateYum

```

In target VM, setup actual yum repository (also based on `linux-yum`, this time without modifications). You will also need to setup some gpg key for signing packages (it is possible to force yum to install unsigned packages, but it isn't possible for `qubes-dom0-update` tool). Fill `~/rpmmacros` with key description:

```
%_gpg_name Test packages signing key
```

Then setup `local.UpdateYum` qrexec service (`/usr/local/etc/qubes-rpc/local.UpdateYum`):

```

#!/bin/sh

if [ -z "$QREXEC_REMOTE_DOMAIN" ]; then
    exit 1
fi

```

(continues on next page)

(continued from previous page)

```

real_repository=/home/user/linux-yum
incoming=/home/user/QubesIncoming/$QREXEC_REMOTE_DOMAIN

find $incoming -name '*.rpm' |xargs rpm -K |grep -iv pgp |cut -f1 -d: |xargs -r setsid -
↳w rpm --addsign 2>&1

rsync -lr --remove-source-files $incoming/ $real_repository
cd $real_repository
export SKIP_REPO_CHECK=1
if [ -d $incoming/r3.1 ]; then
    ./update_repo-unstable.sh r3.1
fi

if [ -d $incoming/r3.0 ]; then
    ./update_repo-unstable.sh r3.0
fi

if [ -d $incoming/r2 ]; then
    ./update_repo-unstable.sh r2
fi
find $incoming -type d -empty -delete
exit 0

```

Of course you will also need to setup qrexec policy in dom0 `/etc/qubes-rpc/policy/local.UpdateYum`.

If you want to access the repository from network, you need to setup HTTP server serving it, and configure the system to let other machines actually reach this HTTP server. You can use for example using [port forwarding](#) or setting up Tor hidden service. Configuration details of those services are outside of the scope of this page.

Usage: setup `builder.conf` in source VM to use your dummy-uploader repository:

```
LINUX_REPO_BASEDIR = ../../repo-yum-upload/r3.1
```

Then use `make update-repo-unstable` to upload the packages. You can also specify selected components on command line, then build them and upload to the repository:

```
make COMPONENTS="core-agent-linux gui-agent-linux linux-utils" qubes update-repo-unstable
```

On the test machine, add yum repository (`/etc/yum.repos.d`) pointing at just configured HTTP server. For example:

```

[local-test]
name=Test
baseurl=http://local-test.lan/linux-yum/r$releasever/unstable/dom0/fc20

```

Remember to also import gpg public key using `rpm --import`.

## Deb packages - Apt repo

Steps are mostly the same as in the case of yum repo. The only details that differ:

- use `linux-deb` instead of `linux-yum` as a base - both in source and target VM
- use different `update_repo.sh` script in source VM (below)
- use `local.UpdateApt` `qrexec` service in target VM (code below)
- in target VM additionally place `update-local-repo.sh` script in repository dir (code below)

`update_repo.sh` script:

```
#!/bin/sh

set -e

current_release=$1
VMNAME=repo-vm

qvm-copy-to-vm $VMNAME $1
find $current_release -type f -name '*.deb' -delete
rm -f $current_release/vm/db/*
qrexec-client-vm $VMNAME local.UpdateApt
```

`local.UpdateApt` service code (`/usr/local/etc/qubes-rpc/local.UpdateApt` in repo-serving VM):

```
#!/bin/sh

if [ -z "$QREXEC_REMOTE_DOMAIN" ]; then
    exit 1
fi

incoming=/home/user/QubesIncoming/$QREXEC_REMOTE_DOMAIN

rsync -lr --remove-source-files $incoming/ /home/user/linux-deb/
cd /home/user/linux-deb
export SKIP_REPO_CHECK=1
if [ -d $incoming/r3.1 ]; then
    for dist in `ls r3.1/vm/dists`; do
        ./update-local-repo.sh r3.1/vm $dist
    done
fi

if [ -d $incoming/r3.0 ]; then
    for dist in `ls r3.0/vm/dists`; do
        ./update-local-repo.sh r3.0/vm $dist
    done
fi

if [ -d $incoming/r2 ]; then
    for dist in `ls r2/vm/dists`; do
        ./update-local-repo.sh r2/vm $dist
    done
fi
```

(continues on next page)

(continued from previous page)

```
find $incoming -type d -empty -delete
exit 0
```

update-local-repo.sh:

```
#!/bin/sh

set -e

# Set this to your local repository signing key
SIGN_KEY=01ABCDEF

[ -z "$1" ] && { echo "Usage: $0 <repo> <dist>"; exit 1; }

REPO_DIR=$1
DIST=$2

if [ "$DIST" = "wheezy-unstable" ]; then
    DIST_TAG=deb7
elif [ "$DIST" = "jessie-unstable" ]; then
    DIST_TAG=deb8
elif [ "$DIST" = "stretch-unstable" ]; then
    DIST_TAG=deb9
fi

pushd $REPO_DIR
mkdir -p dists/$DIST/main/binary-amd64
dpkg-scanpackages --multiversion --arch "$DIST_TAG*" . > dists/$DIST/main/binary-amd64/
↳ Packages
gzip -9c dists/$DIST/main/binary-amd64/Packages > dists/$DIST/main/binary-amd64/Packages.
↳ gz
cat > dists/$DIST/Release <<EOF
Label: Test repo
Suite: $DIST
Codename: $DIST
Date: `date -R`
Architectures: amd64
Components: main
SHA1:
EOF
function calc_sha1() {
    f=dists/$DIST/$1
    echo -n " "
    echo -n `shasum $f|cut -d' ' -f 1` ""
    echo -n `stat -c %s $f` ""
    echo $1
}
calc_sha1 main/binary-amd64/Packages >> dists/$DIST/Release

rm -f $DIST/Release.gpg
rm -f $DIST/InRelease
gpg -abs -u "$SIGN_KEY" \
```

(continues on next page)

(continued from previous page)

```

    < dists/$DIST/Release > dists/$DIST/Release.gpg
gpg -a -s --clearsign -u "$SIGN_KEY" \
    < dists/$DIST/Release > dists/$DIST/InRelease
popd

if [ `id -u` -eq 0 ]; then
    chown -R --reference=$REPO_DIR $REPO_DIR
fi

```

Usage: add this line to `/etc/apt/sources.list` on test machine (adjust host and path):

```
deb http://local-test.lan/linux-deb/r3.1 jessie-unstable main
```

### 1.13.39 Qubes ISO building

#### Build Environment

Fedora 36 (and 37) has been successfully used to build Qubes R4.1 with the below steps. Other rpm-based operating systems may also work. Travis-CI uses Ubuntu 18.04 to perform test builds, except it can not test the `./setup` script.

**Notes:** On modern Fedora system (like Fedora 37) SeLinux is enforced by default and is blocking the build system. You would get error like “can’t create transaction lock on `./../rpm/rpm.lock` (Permission denied)”. You can set SeLinux to permissive mode with

```
sudo setenforce 0
```

In `dom0`, install the Fedora 36 (or 37) template if you don’t already have it.

```
sudo qubes-dom0-update qubes-template-fedora-36
```

Create a standalone AppVM from the Fedora template. Set private storage to at least 60 GB if you will be building only the default templates; 100 GB or more if you plan on additional. It’s not required, but if you allocate additional CPU cores, the build process can utilize them at some steps such as the kernel build. Likewise, more memory (up to 16 GB) can help. Last, you may want to disable memory balancing, but keep in mind the impact on your other qubes.

Once you’ve built the development AppVM, open a Terminal window to it and install the necessary dependencies (see *QubesBuilder* for more info):

```
$ sudo dnf install git createrepo rpm-build rpm-sign make python3-sh rpmdevtools rpm-
↪sign dialog perl-open python3-pyyaml perl-Digest-MD5 perl-Digest-SHA
```

Get the necessary keys to verify the sources (run these and other commands below as a regular user, not root):

```

wget https://keys.qubes-os.org/keys/qubes-master-signing-key.asc
gpg --import qubes-master-signing-key.asc
gpg --edit-key 36879494
fpr
# Verify fingerprint! See Note below!
# Once verified, set trust to *ultimate*
# (Typical sequence is trust, 5, Y, q)
wget https://keys.qubes-os.org/keys/qubes-developers-keys.asc
gpg --import qubes-developers-keys.asc

```

**Note** In the above process, we do *not* rely on the security of our server (keys.qubes-os.org) nor the connection (ssl, cert) – we only rely on you getting the Qubes Master Signing Key fingerprint *somehow* and ensuring they match! See [verifying signatures](#) for verification sources.

Now let's bootstrap the builder. Unfortunately, the builder cannot verify itself (the classic Chicken and Egg problem), so we need to verify the signature manually:

```
git clone https://github.com/QubesOS/qubes-builder.git
cd qubes-builder
git tag -v `git describe`
```

**Note** It's very important to check if the verification message contains “Good signature from ...” and does not contain “WARNING: This key is not certified with a trusted signature!”.

Assuming the verification went fine, we're good to go with all the rest without ever thinking more about verifying digital signatures on all the rest of the components, apart from an additional step if doing a non-scripted build. The builder will do that for us for each component, every time we build, even for all auxiliary files (e.g. Xen or Linux kernel sources).

### Build using setup script

Let's configure the builder first (see [procedure](#) at bottom if you would prefer to manually configure):

```
cd ~/qubes-builder
./setup
# Select Yes to add Qubes Master Signing Key
# Select Yes to add Qubes OS Signing Key
# Select 4.1 for version
# Stable
# Select Yes for fast Git cloning
# Select Current (if you want the option to use pre-built packages)
# Select No (we want a full build)
# Select fc36 and bullseye (for the currently shipping templates)
# Select builder-rpm, builder-debian, template-whonix, mgmt-salt
# Select Yes to add adrelanos's third party key
# Select Yes (to download)
```

Once it completes downloading, re-run `setup` to add the Whonix templates:

```
./setup
# Choose the same options as above, except at templates select:
# fc36, bullseye, whonix-gateway-16, whonix-workstation-16
```

Continue the build process with:

```
make install-deps
make get-sources
```

When building the Whonix templates, you will often need to add/update the `WHONIX_TBB_VERSION` variable in `builder.conf` at this stage to specify the currently shipping Tor Browser version. See the related note under [Extra Whonix Build Options](#).

You may also want to add `COMPONENTS := $(filter-out gcc,$(COMPONENTS))` to bypass a multiple hour compile step. See [QubesBuilder](#) for more detail.

Finally, if you are making a test build, use:

```
make qubes
make iso
```

Or for a fully signed build (this requires setting `SIGN_KEY` in `builder.conf`):

```
make qubes
make sign-all
make iso
```

Enjoy your new ISO!

## Build using manual steps

Instead of using `./setup`, you can manually configure the build. The script takes care of a lot of the keyring preparation for us, so we first need to set that up.

If you will be building Whonix templates:

```
cd ~
gpg --keyserver pgp.mit.edu --recv-keys 916B8D99C38EAF5E8ADC7A2A8D66066A2EEACCD
gpg --fingerprint 916B8D99C38EAF5E8ADC7A2A8D66066A2EEACCD
```

**Note:** It's very important to check the fingerprint displayed against multiple sources such as the [Whonix web site](#), etc. It should look something like this:

```
pub  rsa4096 2014-01-16 [SC] [expires: 2026-01-23]
      Key fingerprint = 916B 8D99 C38E AF5E 8ADC 7A2A 8D66 066A 2EEA CCDA
uid          [ unknown] Patrick Schleizer <adrelanos@kicksecure.com>
uid          [ unknown] Patrick Schleizer <adrelanos@riseup.net>
uid          [ unknown] Patrick Schleizer <adrelanos@whonix.org>
sub  rsa4096 2014-01-16 [E] [expires: 2026-01-23]
sub  rsa4096 2014-01-16 [A] [expires: 2026-01-23]
sub  rsa4096 2014-01-16 [S] [expires: 2026-01-23]
```

Next, prepare the Git keyring directory and copy them in:

```
export GNUPGHOME=~/.qubes-builder/keyrings/git
mkdir --parents "$GNUPGHOME"
cp ~/.gnupg/pubring.gpg "$GNUPGHOME"
cp ~/.gnupg/trustdb.gpg "$GNUPGHOME"
chmod --recursive 700 "$GNUPGHOME"
```

Copy one of the example configurations:

```
cd ~/qubes-builder
cp example-configs/qubes-os-master.conf builder.conf
```

Edit `builder.conf`, referring to `doc/Configuration.md` for a description of all available options.

Continue the build process with:

```
make install-deps
make get-sources
unset GNUPGHOME
```

When building the Whonix templates, you will often need to add/update the `WHONIX_TBB_VERSION` variable at this stage to specify the currently shipping Tor Browser version. See the related note under [Extra Whonix Build Options](#).

Finally, if you are making a test build, use:

```
make qubes
make iso
```

Or for a fully signed build (this requires setting `SIGN_KEY` in `builder.conf`):

```
make qubes
make sign-all
make iso
```

Enjoy your new ISO!

### 1.13.40 Release notes

#### Qubes R1.0 release notes

Detailed release notes in [this blog post](#).

#### Known issues

- Installer might not support some USB keyboards (#230). This seems to include all the Mac Book keyboards (most PC laptops have PS2 keyboards and are not affected).
- If you don't enable Composition (System Setting -> Desktop -> Enable desktop effects), which you really should do, then the KDE task bar might get ugly (e.g. half of it might be black). This is some KDE bug that we don't plan to fix.
- Some keyboard layout set by KDE System Settings can cause [keyboard not working at all](#). If you hit this issue, you can switch to console (by console login option) and manually edit `/etc/X11/xorg.conf.d/00-system-setup-keyboard.conf` (and `/etc/sysconfig/keyboard`) and place correct keyboard layout settings (details in linked thread). You can check if specific keyboard layout settings are proper using `setxkbmap` tool.
- On systems with more than 8GB of RAM there is problem with DisposableVM. To fix it, limit maximum memory allocation for DispVM to 3GB

```
qvm-prefs -s fedora-17-x64-dvm maxmem 3072
qvm-create-default-dvm --default-template --default-script
```

- On some systems the KDE Window Manager might freeze upon resuming from S3 sleep when compositing is enabled (and the only method to log in to the system if this happens is to switch to a text console, enter your user's password, kill the `kwin` process, go back to the Xorg console, log in, and start a new instance of `kwin` using Konsole application :) If you experience such problems, make sure to disable compositing before putting the system into sleep by pressing `Alt-Ctrl-F12` (and then enabling it back once you log in after resume) – this way you should never see this problem again.



## Downloads

See Qubes Downloads.

## Installation instructions

See *Installation Guide*.

## Upgrading

### From Qubes 1.0-rc1

If you're already running Qubes 1.0-rc1, you don't need to reinstall, it's just enough to update the packages in your Dom0 and the template VM(s). The easiest way for doing this is to click on the Update Button in the Qubes Manger – one click when you selected Dom0, and one click for each of your template VM (by default there is just one template).

### From Qubes 1.0 Beta 3

If you have Qubes Beta 3 currently installed on your system, you must reinstall from scratch, as we offer no direct upgrade option in the installer (sorry). However, we do offer tools for smooth migration of your AppVMs. In order to do that, please backup your AppVMs using the `qvm-backup` tool *as usual*. Then, after you install Qubes 1.0 rc1, you can restore them using `qvm-backup-restore` tool. However, because we have changed the default template in RC1, you should tell `qvm-back-restore` about that by passing `--replace-template` option:

```
qvm-backup-restore <backup_dir> --replace-template=fedora-15-x64:fedora-17-x64
```

## Qubes R2.0 release notes

Detailed release notes in [this blog post](#)

## New features since 1.0

- Support for generic fully virtualized VMs (without qemu in the TCB!)
- Support for Windows-based AppVMs integration (clipboard, file exchange, qrexec, pv drivers)
- Secure audio input to select AppVMs (Hello Skype users!)
- Clipboard is now also controlled by central policies, unified with other qrexec policies.
- Out of the box TorVM support
- Experimental support for PVUSB
- Updated Xorg packages in Dom0 to support new GPUs
- DisposableVM customization support
- Introduced Xfce 4.10 environment for Dom0 as an alternative to KDE
- Advanced infrastructure for system backups
- Ability to autostart selected VM at system startup

- Support for dynamic screen resolution change
- Dom0 distribution upgraded to Fedora 20

### Known issues

- On some graphics cards the Xfce4 Window Manager (one of the two supported Dom0 Windows Managers in Qubes R2, the other being KDE) might behave “strangely”, e.g. decorations might not be drawn sometimes. Also the accompanying lightdm login manager might incorrectly display the wallpaper. If you’re facing those problems, it’s advisable to use the KDE Window Manager and kdm instead of Xfce4 and lightdm (this is default if one chooses the KDE only installation option in the installer).
- Some icons in the Qubes Manager application might not be drawn correctly when using the Xfce4 environment in Dom0. If this bothers you, please use the KDE environment instead.
- If your GPU is not correctly supported by the Dom0 kernel (e.g. the 3D desktop effects do not run smoothly) then you might experience “heaviness” with Windows 7-based AppVMs. In that case, please solve the problem with your GPU support in Dom0 in the first place (by using a different kernel), or install Qubes OS on a different system.
- Under some circumstances, Qubes backup can create broken backup, without any visible message (#902). It is advisable to verify a backup to spot the problem. If you encounter this problem, backup VM directory manually.
- System shutdown sometimes is very slow (#903). To mitigate the problem, shutdown all the VMs first.
- For other known issues take a look at [our trac tickets](#)

It is advised to install updates just after system installation to apply bug fixes for (some of) the above problems.

### Downloads

See [Qubes Downloads](#).

### Installation instructions

See *[Installation Guide](#)*.

### Upgrading

#### From Qubes R2 rc1

Upgrading from Qubes R2 rc1 should be a simple matter of installing updates for *dom0* and *VMs*.

## From Qubes R2 beta 3 and older

The easiest and safest way to upgrade to Qubes R2 (especially from older releases) is to install it from scratch and use *qubes backup and restore tools* for migrating of all of the user VMs.

Users of R2 beta 3 can upgrade using procedure that has been described [here](#).

Note: if the user has custom Template VMs (i.e. other than the default template, e.g. created from it by cloning), or Standalone VMs, then the user should perform manual upgrade from R2B3 to R2rc1, as described under the link given above.

## Migrating between beta releases

### From Qubes R1 to R2 beta1

If you're already running Qubes Release 1, you don't need to reinstall, it's just enough to update the packages in your Dom0 and the template VM(s). This procedure is described [here](#)?

### From Qubes R1 or R2 Beta 1 to R2 beta2

Because of the distribution change in R2B2 (from fc13 to fc18) it's preferred that users reinstall Qubes R2B2 from scratch, and use *qubes backup and restore tools* for migrating of all of the user VMs.

Advanced users (and advanced users only) can also try a manual upgrade procedure that has been described [here](#). It's advisable to backup your VMs before proceeding anyway!

### Upgrading from Qubes R1 or R2 Beta 2 to R2 beta 3

The easiest and safest way to upgrade to Qubes R2B3 is to install it from scratch and use *qubes backup and restore tools* for migrating of all of the user VMs.

Users can also try a manual upgrade procedure that has been described [here](#).

Note: if the user has custom Template VMs (i.e. other than the default template, e.g. created from it by cloning), or Standalone VMs, then the user should perform manual upgrade from R2B2 to R2B3, as described under the link given above.

## Qubes R3.0 release notes

This Qubes OS release is dedicated to the memory of Caspar Bowden.

### New features since 2.0

- HAL (Hypervisor Abstraction Layer) - based on libvirt, opens a whole new possibilities of using different hypervisors. Currently Qubes OS uses Xen.
- Xen 4.4 - many new features, but for us the most important is much more mature libxl toolstack.
- Qrexec 3 - greatly improved performance by using direct VM-VM connections and bigger buffers.
- Debian templates gets official support.
- Whonix templates

- Build system improvements - especially support for distribution-specific plugins (makes supporting multiple distributions much easier) and building templates using DispVM.
- Automated tests - makes much easier to find bugs, before its even shipped to users

### Known issues

- Windows Tools: `qvm-block` does not work
- UEFI is not supported, you need to enable “legacy boot” in BIOS before installing Qubes OS
- Some icons in the Qubes Manager application might not be drawn correctly when using the Xfce4 environment in Dom0. If this bothers you, please use the KDE environment instead.
- If your GPU is not correctly supported by the Dom0 kernel (e.g. the 3D desktop effects do not run smoothly) then you might experience “heaviness” with Windows 7-based AppVMs. In that case, please solve the problem with your GPU support in Dom0 in the first place (by using a different kernel), or install Qubes OS on a different system.
- For other known issues take a look at [our tickets](#)

It is advised to install updates just after system installation to apply bug fixes for (some of) the above problems.

### Downloads

See Qubes Downloads.

### Installation instructions

See *Installation Guide*.

### Upgrading

#### From R3.0 release candidate

If you are using Qubes R3.0rc1, R3.0rc2 or R3.0rc3, just install system updates, there is no special steps required.

#### From R2.0 or earlier

The easiest and safest way to upgrade to Qubes R3.0 is to install it from scratch and use *qubes backup and restore tools* for migrating of all of the user VMs.

Users of Qubes R2 can upgrade using *experimental procedure*.

## Qubes R3.1 release notes

### New features since 3.0

- Management Stack based of Salt Stack in dom0 - [documentation](#)
- Out of the box Whonix setup
- UEFI support
- LIVE edition (still alpha, not part of R3.1-rc1)
- Updated GPU drivers in dom0
- Colorful window application icons (instead of just colorful lock icon)
- PV Grub support ([documentation](#))
- Out of the box USB VM setup, including [handling USB mouse](#)
- Xen upgraded to 4.6, for better hardware support (especially Skylake platform)
- Improve updates proxy flexibility - especially repositories served over HTTPS

You can get detailed description in [completed github issues](#)

### Known issues

- Installation image does not fit on DVD, requires either DVD DL, or USB stick (5GB or more)
- Windows Tools: `qvm-block` does not work
- Some icons in the Qubes Manager application might not be drawn correctly when using the Xfce4 environment in Dom0. If this bothers you, please use the KDE environment instead.
- USB mouse (in the case of USB VM) does not work at first system startup (just after completing firstboot). Workaround: restart the system.
- For other known issues take a look at [our tickets](#)

It is advised to install updates just after system installation to apply bug fixes for (some of) the above problems.

### Downloads

See Qubes Downloads.

### Installation instructions

See [Installation Guide](#).

### Upgrading

#### From R3.0

The easiest and safest way to upgrade to Qubes R3.1 is to install it from scratch and use *qubes backup and restore tools* for migrating of all of the user VMs.

Users of Qubes R3.0 can upgrade using *experimental procedure*.

#### From R2 or earlier

When upgrading from earlier versions the easiest and safest way is to install it from scratch and use *qubes backup and restore tools* for migrating of all of the user VMs.

Alternatively you can *upgrade to R3.0 using* first, then follow the instructions above. This will be time consuming process.

### Qubes R3.2 release notes

#### New features since 3.1

- Management Stack extended to support in-VM configuration - *documentation*
- PV USB - *documentation*
- Dom0 update to Fedora 23 for better hardware support
- Kernel 4.4.x
- Default desktop environment switched to Xfce4
- KDE 5 support (but it is no longer the default one)
- Tiling window managers support: awesome, *i3*
- More flexible Qubes RPC services - *related ticket*, *documentation*

You can get detailed description in *completed github issues*

#### Known issues

- *Fedora 23 reached EOL in December 2016*. There is a *manual procedure to upgrade your VMs*.
- Windows Tools: *qvm-block* does not work
- Some icons in the Qubes Manager application might not be drawn correctly when using the Xfce4 environment in Dom0. If this bothers you, please use the KDE environment instead.
- For other known issues take a look at *our tickets*

It is advised to install updates just after system installation to apply bug fixes for (some of) the above problems.

## Downloads

See Qubes Downloads.

## Installation instructions

See *Installation Guide*. After installation, manually upgrade to Fedora 26.

## Upgrading

### From R3.1

The easiest and safest way to upgrade to Qubes R3.2 is to install it from scratch and use *qubes backup and restore tools* for migrating of all of the user VMs.

Users of Qubes R3.1 can also upgrade using *this procedure*.

### From R3.0 or earlier

When upgrading from earlier versions the easiest and safest way is to install it from scratch and use *qubes backup and restore tools* for migrating of all of the user VMs.

Alternatively you can *upgrade to R3.1 using* first, then follow the instructions above. This will be time consuming process.

## Qubes R4.0 release notes

### New features since 3.2

- Core management scripts rewrite with better structure and extensibility, [API documentation](#)
- [Admin API](#) allowing strictly controlled managing from non-dom0
- All `qvm-*` command-line tools rewritten, some options have changed
- Renaming VM directly is prohibited, there is GUI to clone under new name and remove old VM
- Use [PVH](#) and [HVM](#) by default to [mitigate Meltdown & Spectre](#) and lower the [attack surface on Xen](#)
- Create USB VM by default
- [Multiple DisposableVMs templates support](#)
- New *backup format* using `scrypt` key-derivation function
- Non-encrypted backups no longer supported
- [split VM packages](#), for better support minimal, specialized templates
- [Qubes Manager decomposition](#) - domains and devices widgets instead of full Qubes Manager; devices widget support also USB
- *More flexible firewall interface* for ease unikernel integration
- Template VMs do not have network interface by default, [qrexec-based updates proxy](#) is used instead
- More flexible IP addressing for VMs - [custom IP](#), [hidden from the IP](#)

- More flexible Qubes RPC policy - [related ticket](#), [documentation](#)
- [New Qubes RPC confirmation window](#), including option to specify destination VM
- [New storage subsystem design](#)
- Dom0 update to Fedora 25 for better hardware support
- Kernel 4.9.x

You can get detailed description in [completed github issues](#)

### Security Notes

- PV VMs migrated from 3.2 to 4.0-rc4 or later are automatically set to PVH mode in order to protect against Meltdown (see [QSB #37](#)). However, PV VMs migrated from any earlier 4.0 release candidate (RC1, RC2, or RC3) are not automatically set to PVH mode. These must be set manually.
- The following steps may need to be applied in dom0 and Fedora 26 TemplateVMs in order to receive updates (see [#3737](#)). Steps for dom0 updates:

1. Open the Qubes Menu by clicking on the “Q” icon in the top-left corner of the screen.
2. Select **Terminal Emulator**.
3. In the window that opens, enter this command:

```
sudo nano /etc/yum.repos.d/qubes-dom0.repo
```

4. This opens the nano text editor. Change all four instances of `http` to `https`.
5. Press CTRL+X, then Y, then ENTER to save changes and exit.
6. Check for updates normally.

Steps for Fedora 26 TemplateVM updates:

1. Open the Qubes Menu by clicking on the “Q” icon in the top-left corner of the screen.
2. Select **Template: fedora-26**, then **fedora-26: Terminal**.
3. In the window that opens, enter the command for your version:

```
[Qubes 3.2] sudo gedit /etc/yum.repos.d/qubes-r3.repo  
[Qubes 4.0] sudo gedit /etc/yum.repos.d/qubes-r4.repo
```

4. This opens the gedit text editor in a window. Change all four instances of `http` to `https`.
5. Click the “Save” button in the top-right corner of the window.
6. Close the window.
7. Check for updates normally.
8. Shut down the TemplateVM.



## Known issues

- Locale using coma as decimal separator [crashes qubesd](#). Either install with different locale (English (United States) for example), or manually apply fix explained in that issue.
- In the middle of installation, [keyboard layout reset to US](#). Be careful what is the current layout while setting default user password (see upper right screen corner).
- On some laptops (for example Librem 15v2), touchpad do not work directly after installation. Reboot the system to fix the issue.
- List of USB devices may contain device identifiers instead of name
- With R4.0.1, which ships kernel-4.19, you may never reach the anaconda startup and be block on an idle black screen with blinking cursor. You can try to add `plymouth.ignore-serial- consoles` in the grub installer boot menu right after `quiet rhgb`. With legacy mode, you can do it directly when booting the DVD or USB key. In UEFI mode, follow the same procedure described for [disabling nouveau module](#) (related [solved issue](#) in further version of Qubes).
- For other known issues take a look at [our tickets](#)

It is advised to install updates just after system installation to apply bug fixes for (some of) the above problems.

## Downloads

See Qubes Downloads.

## Installation instructions

See *Installation Guide*.

## Upgrading

There is no in-place upgrade path from earlier Qubes versions. The only supported option to upgrade to Qubes R4.0 is to install it from scratch and use *qubes backup and restore tools* for migrating of all of the user VMs. We also provide *detailed instruction* for this procedure.

## Qubes OS 4.1 release notes

### New features and improvements since Qubes 4.0

- Optional qubes-remote-support package now available from repositories (strictly opt-in, no package installed by default; no new ports or network connections open by default; requires explicit connection initiation by the user, then requires sharing a code word with the remote party before a connection can be established; see [#6364](#) for more information)
- Qubes firewall reworked to be more defensive (see [#5540](#) for details)
- Xen upgraded to version 4.14
- Dom0 operating system upgraded to Fedora 32
- Default desktop environment upgraded to Xfce 4.14
- Upgraded default template releases

- Experimental support for GUI running outside of dom0 (hybrid mode GUI domain without real GPU passthrough; see [#5662](#) for details)
- Experimental support for audio server running outside of dom0 (“Audio domain”)
- sys-firewall and sys-usb are now disposables by default
- UEFI boot now loads GRUB, which in turn loads Xen, making the boot path similar to legacy boot and allowing the user to modify boot parameters or choose an alternate boot menu entry
- New qrexec policy format (see [#4370](#) for details)
- qrexec protocol improvements (see [#4909](#) for details)
- New qrexec-policy daemon
- Simplified using in-qube kernels
- Windows USB and audio support courtesy of [tabit-pro](#) (see [#5802](#) and [#2624](#))
- Clarified disposable-related terminology and properties
- Default kernelopts can now be specified by a kernel package
- Improved support for high-resolution displays
- Improved notifications when a system drive runs out of free space
- Support for different cursor shapes
- “Paranoid mode” backup restore option now properly supported using disposables
- Users can now choose between Debian and Fedora in the installer
- Certain files and applications are now opened in disposables, e.g., Thunderbird email attachments
- New graphical interface for managing testing repository updates
- New “Cute Qube” icon family (replaces padlock icons)
- Disposable qube types now use the disposable icon
- New Template Manager tool for installing, removing, and updating templates (meanwhile, the tool previously known as the “Template Manager,” which was for mass template switching, has been integrated into the Qube Manager)
- The “file” storage driver has been deprecated in Qubes 4.1 and will be removed in Qubes 4.2
- `property-del` event renamed to `property-reset` to avoid confusion
- qrexec no longer supports non-executable files in `/etc/qubes-rpc`
- qrexec components have been reorganized into the core-qrexec repository
- The `qvm-pool` argument parser has been rewritten and improved
- Removed the need for the out-of-tree `u2mfn` kernel module
- Qrexec services can now run as a socket server
- Improved template distribution mechanism
- Now possible to restart qrexec-agent
- The term “VM” has largely been replaced by “qube”
- GUI daemon is now configured using `qvm-features` tool, `/etc/qubes/guid.conf` file is no longer used
- `qvm-run` tool got `--no-shell` option to run a single command without using a shell inside the qube

- MAC Randomization for iwlwifi (see [#938](#))

For a full list, including more detailed descriptions, please see [here](#).

## Known issues

For a full list of known 4.1 issues with open bug reports, please see [here](#). We strongly recommend *updating Qubes OS* immediately after installation in order to apply any and all available bug fixes.

## Download

See downloads.

## Installation instructions

See the *installation guide*.

## Upgrading

Please see *how to upgrade to Qubes 4.1*.

## Qubes OS 4.2.0 release notes

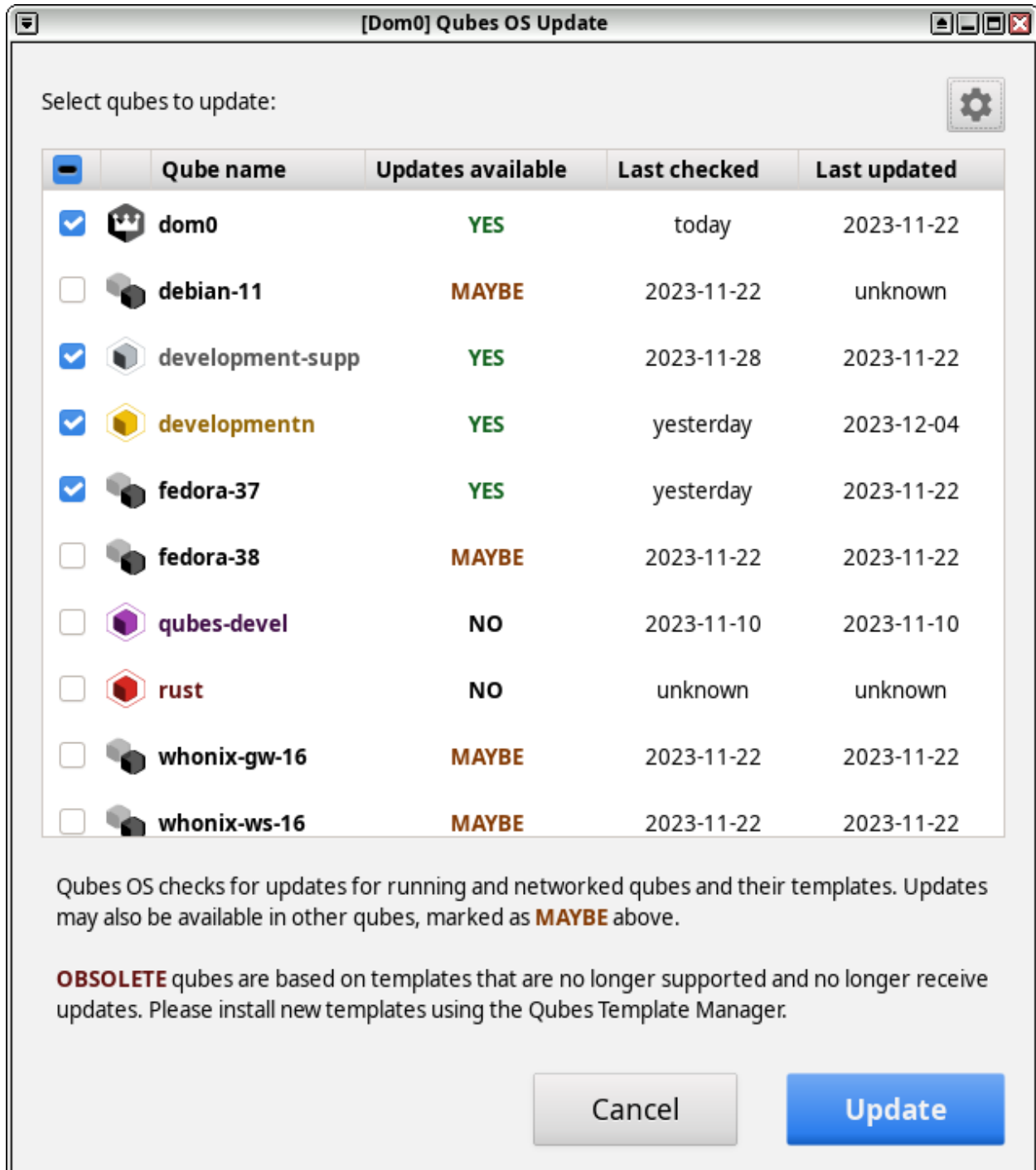
### New features and improvements since Qubes 4.1

- Dom0 upgraded to Fedora 37 ([#6982](#))
- Xen upgraded to version 4.17
- Default Debian template upgraded to Debian 12
- Default Fedora and Debian templates use Xfce instead of GNOME ([#7784](#))
- SELinux support in Fedora templates ([#4239](#))
- Several GUI applications rewritten (screenshots below), including:
  - Applications Menu (also available as preview in R4.1) ([#6665](#)), ([#5677](#))
  - Qubes Global Settings ([#6898](#))
  - Create New Qube
  - Qubes Update ([#7443](#))
- Unified `grub.cfg` location for both UEFI and legacy boot ([#7985](#))
- PipeWire support ([#6358](#))
- fwupd integration for firmware updates ([#4855](#))
- Optional automatic clipboard clearing ([#3415](#))
- Official packages built using Qubes Builder v2 ([#6486](#))
- Split GPG management in Qubes Global Settings

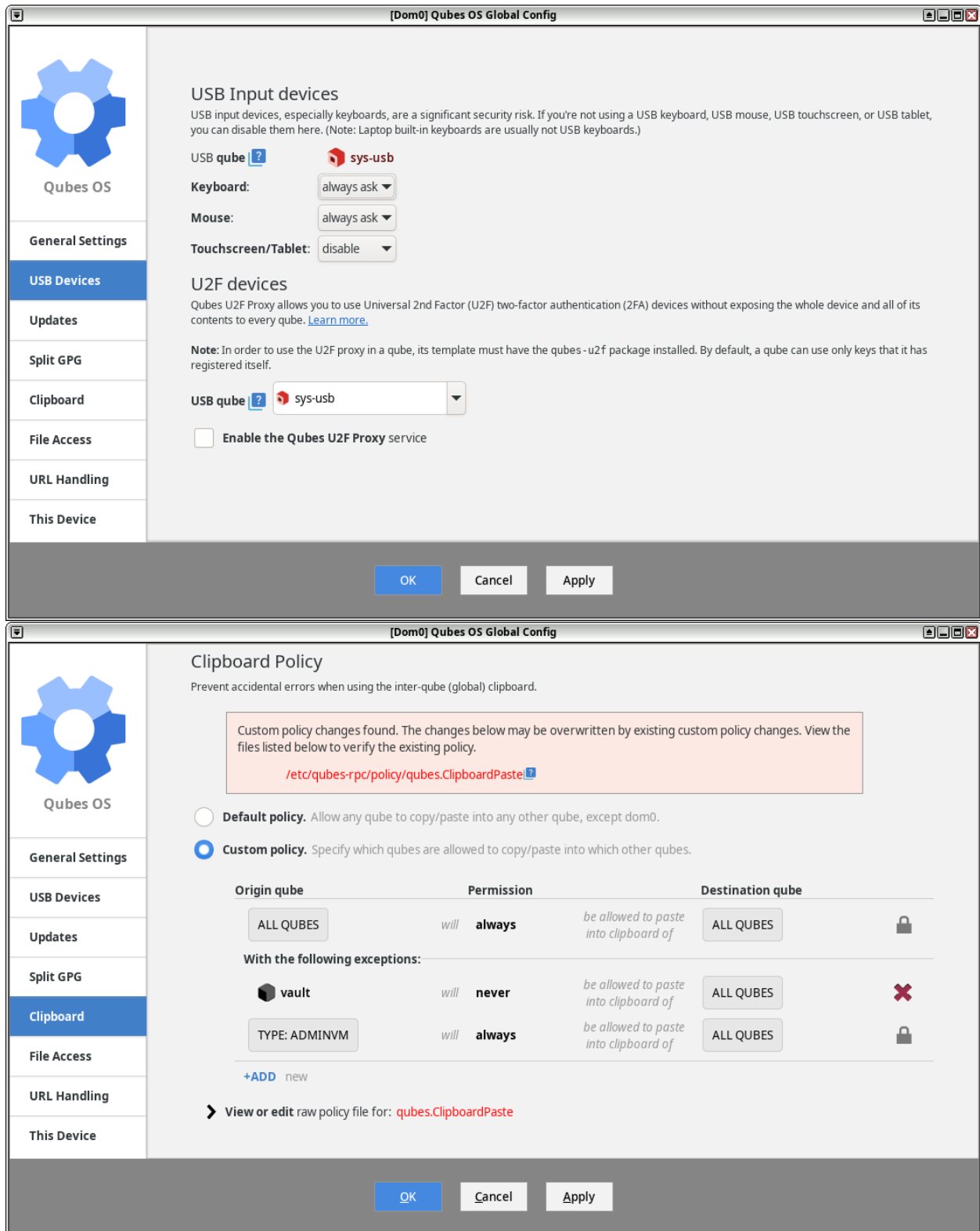
- Qrexec services use new qrexec policy format by default (but old format is still supported) (#8000)
- Improved keyboard layout switching

For a full list, including more detailed descriptions, please see [here](#). Below are some screenshots of the new and improved Qubes GUI tools.

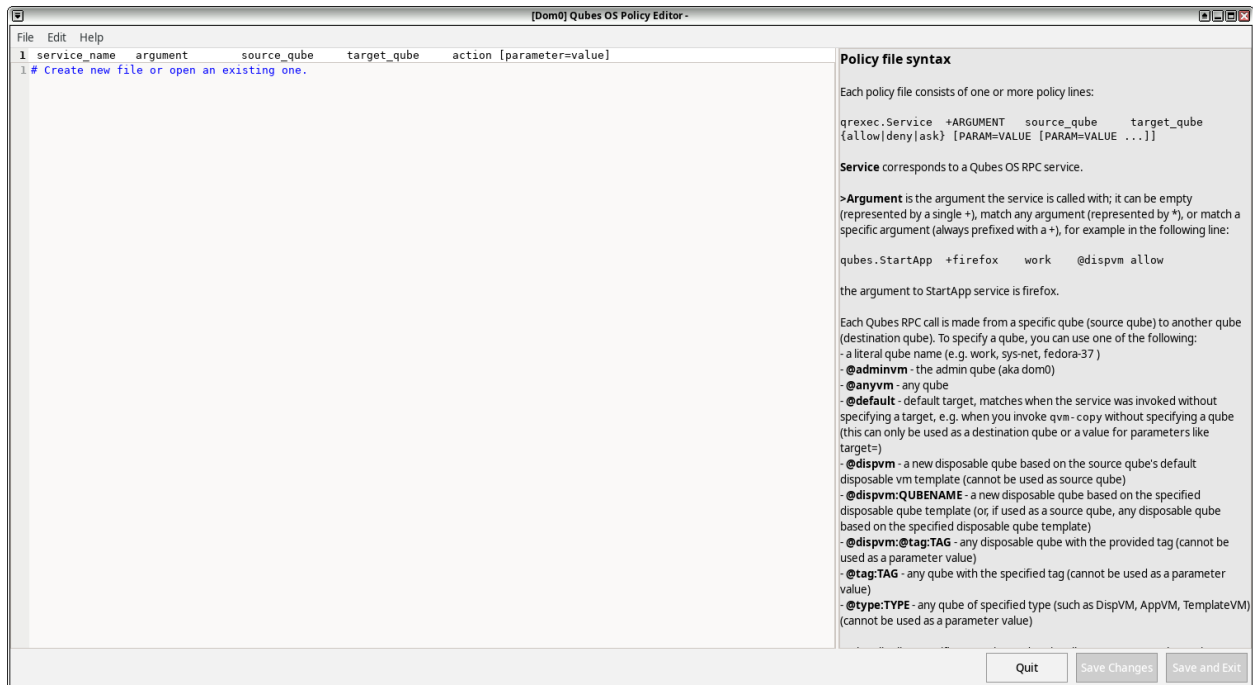
The new Qubes OS Update tool:



The new Qubes OS Global Config tool:



The new Qubes OS Policy Editor tool:



### Known issues

- DomU firewalls have completely switched to nftables. Users should add their custom rules to the `custom-input` and `custom-forward` chains. ([#5031](#), [#6062](#))

For a full list of open bug reports affecting 4.2, please see [here](#). We strongly recommend *updating Qubes OS* immediately after installation in order to apply any and all available bug fixes.

### Download

All Qubes ISOs and associated *verification files* are available on the [downloads](#) page.

### Installation instructions

See the *installation guide*.

### Upgrading

Please see *how to upgrade to Qubes 4.2*.

### 1.13.41 Release schedules

#### Qubes R3.0 release schedule

Date	Stage
5 Sep 2015	current-testing freeze before 3.0-rc3
15 Sep 2015	3.0-rc3 release
1 Oct 2015	3.0 release

#### Qubes R3.1 release schedule

This schedule is based on *Version Scheme*.

Date	Stage
8 Dec 2015	3.1-rc1 release
5 Jan 2016	current-testing freeze before 3.1-rc2
12 Jan 2016	3.1-rc2 release
26 Jan 2016	decide whether 3.1-rc2 is the final 3.1
9 Feb 2016	current-testing freeze before 3.1-rc3
16 Feb 2016 23 Feb 2016	3.1-rc3 release

#### Qubes R3.2 release schedule

This schedule is based on *Version Scheme*.

Date	Stage
18 Jun 2016	3.2-rc1 release
2 Jul 2016	decide whether 3.2-rc1 is the final 3.2
16 Jul 2016 20 Jul 2016	current-testing freeze before 3.2-rc2
23 Jul 2016 27 Jul 2016	3.2-rc2 release
5 Aug 2016 9 Aug 2016	decide whether 3.2-rc2 is the final 3.2
24 Aug 2016	current-testing freeze before 3.2-rc3
31 Aug 2016	3.2-rc3 release
29 Sep 2016	3.2 release

#### Qubes R4.0 release schedule

This schedule is based on *Version Scheme*.

Date	Stage
31 Jul 2017	4.0-rc1 release
28 Aug 2017 11 Sep 2017 9 Oct 2017 18 Oct 2017	current-testing freeze before 4.0-rc2
4 Sep 2017 18 Sep 2017 16 Oct 2017 23 Oct 2017	4.0-rc2 release
6 Nov 2017	decide whether 4.0-rc2 is the final 4
20 Nov 2017	current-testing freeze before 4.0-rc3
27 Nov 2017	4.0-rc3 release
11 Dec 2017	decide whether 4.0-rc3 is the final 4.0
1 Jan 2018	current-testing freeze before 4.0-rc4
8 Jan 2018 31 Jan 2018	4.0-rc4 release
22 Jan 2018 14 Feb 2018	decide whether 4.0-rc4 is the final 4.0
27 Feb 2018	current-testing freeze before 4.0-rc5
6 Mar 2018	4.0-rc5 release
20 Mar 2018	decide whether 4.0-rc5 is the final 4.0
28 Mar 2018	final 4.0 release

### Qubes R4.1 release schedule

The table below is based on our *release schedule policy*.

Date	Stage
2021-10-11	4.1.0-rc1 release
2021-11-08	current-testing freeze before 4.1.0-rc2
2021-11-15	4.1.0-rc2 release
2021-11-29	decide whether 4.1.0-rc2 is the final 4.1
2021-12-13	current-testing freeze before 4.1.0-rc3
2021-12-20	4.1.0-rc3 release
2022-01-03	decide whether 4.1.0-rc3 is the final 4.1
2022-01-11	current-testing freeze before 4.1.0-rc4
2022-01-18	4.1.0-rc4 release
2022-01-31	decide whether 4.1.0-rc4 is the final 4.1
2022-02-04	final 4.1.0 release

### Qubes R4.2 release schedule

**Please note:** *This page is still an unfinished draft in progress. It is being updated as Qubes 4.2 development and testing continues.*

The table below is based on our *release schedule policy*.

Date	Stage
2023-06-02	4.2.0-rc1 release
2023-08-28	4.2.0-rc2 release
2023-09-03	4.2.0-rc3 release
2023-10-13	4.2.0-rc4 release



### 1.13.42 Release checklist

*the checklist is probably unfinished*

#### On -rc1

- write schedule
- create package repositories (linux-yum, linux-deb)
- update repository definition (core-agent-linux, installer-qubes-os/qubes-release)
- push all packages to `current-testing`
- draft release notes, one note per feature
- create upgrade package in previous release branch (r2->r3.0, r3.0->r3.1, etc) - core-agent-linux
- make sure that keys for the current release are included in previous release's qubes-release package (for upgrade)
- build ISO and push to mirrors

#### On subsequent -rc

- push packages to `current`
- update release notes
- build ISO and push to mirrors
- notify @Rudd-O about the new ISO for new torrent hosting

#### On final release

- push packages to `current`
- finish release notes
- update InstallationInstructions
- build ISO and push to mirrors
- notify @Rudd-O about the new ISO for new torrent hosting
- write blog post
- announce on Twitter

### 1.13.43 Version scheme

The Qubes OS Project uses the [semantic versioning](#) standard. Version numbers are written as `<major>.<minor>.<patch>`. When `<patch>` is omitted (e.g., 4.1), it is usually either because `<patch>` is zero (as in 4.1.0) or because we are referring to a specific minor release irrespective of any particular patch release within it. Similarly, the major release number alone (e.g., R4) is sometimes used to refer to an entire release series inclusive of all minor and patch releases within it.

In general, patch releases are for backward-compatible bug fixes, minor releases are for backward-compatible enhancements and new features, and major release are for any backward-incompatible changes. This means that, in general, one should *not* try to introduce features or enhancements in patch releases or any backward-incompatible changes in

patch or minor releases. (Templates are a notable exception, as upstream OSes almost always have their own release schedules.) Bug fixes are allowed in all releases, and backward-compatible changes are allowed in all major and minor releases.

Qubes OS minor releases generally include new features, new templates, and occasionally new defaults, but they are still backward-compatible in the sense that qubes and features that worked in the previous release still function, though the UI may be different in some cases. In general, deprecated features are removed only in major releases, and in-place upgrades between major versions are not guaranteed.

Following standard practice, **version** refers to any build that has been assigned a version name or number, e.g., 3.2-rc2, 4.0.4, 4.1-beta1. By contrast, **release** refers to any version that is intended for consumption by the general userbase. For example, 4.0.4 was both a **version** and a **release**, since it was stable and intended for general public use, while 4.1-beta1 was a **version** but *not* a **release**, since it was not stable and was intended only for *testing*. All releases are versions, but not all versions are releases.

The letter **R**, as in R4.1, stands for **release**. The abbreviation **RC**, as in 3.2-rc2, stands for **release candidate**.

### Qubes distributions and products

We intend to make it easy to make a remix of Qubes, targeting another hypervisor or isolation provider. We may also create commercial products intended for specific circumstances. There is one distinguished distribution called **Qubes OS**. All source code for it is available for download under a *free and open-source license* and is openly developed on [GitHub](#) and our [mailing lists](#). The rest of this document discusses Qubes OS. Another remix may have its own version series.

### Release versioning

Qubes OS as a whole is released from time to time. When preparing a new release, we decide on the <major>.<minor> numbers (e.g., 3.0, which is short for 3.0.0). We then publish the first release candidate, e.g., 3.0.0-rc1. When we feel that enough progress has been made, we'll release 3.0.0-rc2 and so on. All these versions (which are not yet releases) are considered unstable and are not intended for production use. You are welcome to *help us test* these versions.

When enough progress has been made, we announce the first stable release, e.g. 3.0.0. This is not only a version but an actual release. It is considered stable, and we commit to supporting it according to our *support schedule*. Core components are branched at this moment, and bug fixes are backported from the master branch. Please see *help, support, mailing lists, and forum* for places to ask questions about stable releases. No major features or interface incompatibilities are to be included in this release. We release bug fixes as patch releases (3.0.1, 3.0.2, and so on), while backward-compatible enhancements and new features are introduced in the next minor release (e.g., 3.1). Any backward-incompatible changes are introduced in the next major release (e.g., 4.0).

Please see *issue tracking* for information about how releases are handled in the issue tracker.

### Release schedule

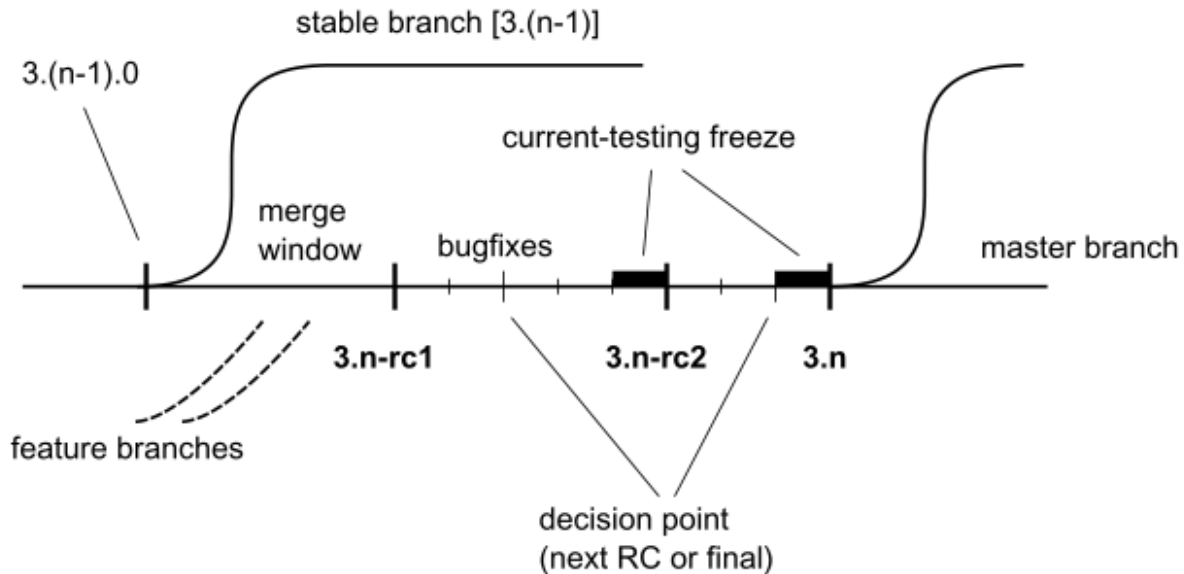
There is no specific schedule for releases other than a general roadmap. When the time comes, we declare a feature freeze, tag -rc1, and release an ISO. From this point on, no new features are accepted, and our schedule begins.

Each release candidate period is as follows: For the first two weeks, we accept and assign bug reports to be fixed before the next release candidate. For the next two weeks, we generally focus on fixing assigned bug reports, so issues discovered during this period may be postponed until a later RC. Finally, there is a one week current-testing freeze, during which time no new packages are released, in the hope that they will be installed and tested by wider user base.

The next RC is released five weeks after the former. All packages are published in the current repository, and the cycle starts over. There should always be at least one release candidate before the final release.

Stage	Duration
initial testing	two weeks
bug fixing	two weeks
current-testing freeze	one week

Starting with the second cycle (that is, after `-rc1`), two weeks into the cycle (after the primary bug-reporting period), we decide whether there should be another RC. If, based on the bugs that have been reported, we decide that the latest RC will be designated as the stable release, then we decide on its release date, which should be no more than one week later.



## Bug priorities

When deciding whether the current release candidate is the final one, the Committee takes bug *priorities* into consideration. The meaning of them is as follows:

- **blocker** — when any such bug is present in the current release candidate, it can't be considered final release. Bugs with this priority must be fixed before the next release candidate, even if that means delaying its release (which should be considered only last resort option).
- **critical** — when any such bug is present in the current release candidate, it can't be considered final release. But such bugs are not qualified to delay next release candidate release.
- **major** — existence of such bugs do not strictly prevent the current release candidate be considered final (but of course we should try hard to not have them there). Fixing bugs of this priority can be delayed and qualified as updates to the final stable release.
- **default** and **minor** — existence of such bugs do not prevent the current release candidate be considered final. Fixing such bugs can be delayed to the next Qubes OS release. Eventually such fixes might be backported as an update to the stable release(s). (**default** should really be assigned a more specific priority, but in practice there are too many issues and not enough time, so **default** ends up staying on many issues.)

All above is about bugs, no features should be assigned to the current release after first `-rc`. Supreme Committee is free to adjust priorities appropriately.

### Component version

Qubes release is defined as specific versions of components, which are developed more or less separately. Their versions are composed of major and minor version of target Qubes OS release followed by third component which is just incremented. There is no apparent indication that given version is stable or not.

There are some non-essential components like `qubes-apps-*` that are shared between releases. Their versions indicate oldest qubes-release that is supported. We try hard to support multiple releases by one branch to ease code maintenance.

Different Qubes releases remixes may comprise of different components and version are not guaranteed to be monotonic between releases. We may decide that for newer release some component should be downgraded. There is no guarantee that arbitrary combination of different versions of random components will yield usable (or even install-able) compilation.

### Git tags and branches

We mark each component version in the repository by tag containing `v<version>`. Likewise, each Qubes OS release is marked by `R<release>` tag.

At the release of some release we create branches named like `release2`. Only bug fixes and compatible improvements are backported to these branches. These branches should compile. All new development is done in `master` branch. This branch is totally unsupported and may not even compile depending on maintainer of repository.

All version and release tags should be made and signed by someone from ITL staff. Public keys are included in `qubes-builder` and available at <https://keys.qubes-os.org/keys/>.

### Check installed version

If you want to know which version you are running, for example to report an issue, you can either check in the Qubes Manager menu under `About > Qubes OS` or in the file `/etc/qubes-release` in `dom0`. For the latter you can use a command like `cat /etc/qubes-release` in a `dom0` terminal.

## 1.14 External Documentation

Unofficial, third-party documentation from the Qubes community and others.