Qub3d Development Documentation

Release 0.0.1

Qub3d Engine Group

Sep 15, 2018

Contents

CHAPTER 1

About the Development Docs

This repository is meant to describe our group's standards, policies, technical workflow, and so forth. For more information, see README.md and the Getting Started Guide.

CHAPTER 2

Devolpment Doc Files

2.1 Guides

2.1.1 Getting Started

25 Febuary 2018, 04:47 GMT

Welcome to the Official Qub3d Getting Started Guide!

This file is dedicated to getting started in general with the Qub3d Engine Group.

The simplest way to get started with Qub3d is to join the official Discord Server. We also have an official IRC channel on freenode: #qub3d. If you have any issues that just don't quite fit inside Discord or IRC, don't fret! We have an official email address: yo@qub3d.org where you can send questions that don't contain spam/inappropriate/irrelevant content. This email is *only* used for questions regarding the Qub3d Engine Group.

There are other guides/howtos that go beyond this guide in some fields, such as the Contributors' Guide and the Documentation Howto. This guide is just to help you get ready for getting our hands dirty with the Qub3d repositories.

The aim of this guide is to familiarize new users to the Qub3d Engine Group and its tools. In this guide, we give a brief introduction on each tool used in the Qub3d project, and the Qub3d Engine Group and all of its participating members. If you appreciate this project enough to contribute, then please read the Contributors' Guide.

Qub3d Engine Group

The team is made up of volunteer developers. They are:

TMcSquared (Thomas Monroe/Tre): Lead Code Wrangler

NewbProgrammer101 (Jalus Bilieyich/Jay): Repository Sentry

SonosFuer (apachano/Austin): Docs Librarian

toby109tt (Toby): Lead Artist

TheScarecröwman (Anonymous): The Leader's Leader

THE ABOVE LIST WILL BE FIXED LATER

TMcSquared

To be filled in later.

NewbProgrammer101

To be filled in later.

SonosFuer

To be filled in later.

toby109tt

To be filled in later.

TheScarecröwman

To be filled in later.

Exploring Phabricator

The applications listed below are found at the left of the Phabricator home page in chronological order.

- **Home:** The home page of our Phabricator.
- **Task Finder [P]:** Find a programming task to work on.
- **Getting Started:** Instructions to get started.
- **Dev Docs:** Where the documentation targeted at developers and contributors is located.
- **Feed:** This is where the liveliness of Qub3d is shown.
- **Flags:** Where all your personal bookmarks within Phabricator are found.

Planning

- **Tasks:** Task and bug tracker. Similar to GitHub Issues. Also called Maniphest.
- **Wiki:** Official, collaborative, and online Wiki dedicated to the Qub3d project.
- **Projects:** Browse projects, groups, and other tags. Projects also have Workboards, similar to GitHub Projects.
- **Design Review:** Review design, mocks, etc.

Code

- **Code Review:** Submit and review code (pre-commit review). Similar to GitHub Pull Requests.
- **Repositories:** The source of the Qub3d Engine Group's works.
- **Audit:** Post-commit review. Raise concerns about code that's already been committed.

Tools

- **Calendar:** Planned events billboard.
- **Etherpad:** Collaborative and real-time live editing of files.
- **Paste:** Pastebin with syntax highlighting, history, and comments. Similar to GitHub Gist.
- **Phurl:** URL shortener. Similar to bit.ly and goo.gl.
- **Ponder:** Q&A, StackOverflow style.
- **Slowvote:** Development polls where everyone can vote and comment on.

Setup Instructions

GNU/Linux

Windows

Mac OS X

2.1.2 Contributors' Guide

This file is dedicated to people who want to contribute to the project but don't know what Qub3d Engine Group's guidelines are.

This guide is not a tutorial on how to get the basics with Qub3d. The Getting Started Guide and the numerous Qub3d support channels exist for that. Instead, this guide is a tutorial on how to take advantage of Qub3d's internals and using it for the purpose of contributing to the project.

Getting Started

This section displays the guidelines for making a development environment for the Qub3d project.

Create an account on Qub^{3}d Engine Group's Phabricator using your GitHub account. If you don't have a GitHub account, go ahead and create one. It's free, only takes a couple minutes, and gives you access to the majority of the open source development world! After you're done creating one, on the main page, click on your profile picture somewhere at the top right and click on *Settings* and customize/configure to your heart's content.

Go to your Phabricator profile's home page by clicking on your profile picture at the top right region, click on *Manage*. On the right, you will see *Edit Profile*. Click on it and fill in the blanks to your heart's content.

Note: Before submitting contributions, the Qub3d Engine Group will need verification that you have signed the Terms of Development, otherwise they cannot accept your diff.

Contributor Requirements

The following are the requirements to meet before contributing to the project.

Code

To contribute to the engine/launcher, you must have fair knowledge of at least *one* of the following languages:

C++, Lua, and YAML.

As they are the languages used in the Qub3d code repositories.

Documentation

If you're just contributing to documentation, you should have the following characteristics:

- Professional Working Proficiency (ILR Level 3) or better with English
- Knowledge of ReST and its syntax
- Knowledge of Markdown (Only applicable if you're writing Markdown in the documentation)

Note: Having fair knowledge of English is mandatory if working on documentation.

Beginning Development

Like any complex project, enhancing the Qub3d project requires a clear description and purpose of the problem you're trying to solve. This section will guide you through on how to plan for developing for the Qub3d project. In some cases such as an immediate documentation fix or a typo fix in the code, you don't need to plan for it.

Isolating The Problem

If you *do* come across a bug, please make sure it's expected behavior by looking at the source code to see which file the problem may be related to, or ask the developers at Freenode IRC or the Discord server. If it's not expected behavior, then the bug is confirmed and you may submit it to Maniphest as a bug report. It is strongly advised that you claim the task as the developers already have enough on their plates. By doing so, scroll down to the bottom of your bug report page and click on *Actions…* → *Assign/Claim* to claim the task. If, for any rational reason, you can't claim the task, then leave it open so someone else can do that for you.

Rules

Below are the rules you must abide by when contributing to the project.

Rules For Submitting Code

There are preliminary checks you must do on your branch before launching. All the criteria is in the checklist.

Rules For Submitting Documentation

See the Documentation Howto about this. Also, don't change the scope of the project via docs. In other words, don't be misleading; that is strictly prohibited in the project.

Miscellaneous

If you don't feel like hacking and/or documenting the Qub3d repositories, there's still plenty of other ways for you to help! You can answer questions on the Discord Server and/or Ponder on Phabricator, find bugs, promote Qub^{3}d, contribute to the Qub^{3}d official website, leave behind ideas in the Ideas Board, help review a diff, provide penetration test results for the qub3d.tk server, and/or give end-user feedback. The list of ways you can contribute to the project is inexhaustible.

Post-Launch

You have launched your first diff, congratulations!

If your diff is a work in progress, then please scroll down to the bottom of your diff page and click *Actions…* → *Plan Changes* to let everyone know that you're not ready for review; that is, unless you allow it by letting them know via comment or summary.

Now What?

You wait for the diff to get reviewed. Once it is reviewed, you wait for approval from the maintainers.

When you are certain that you're finished with your diff, please comment on it, saying: "Done."

The Review Process

The number one rule for waiting on the reviewers is to be patient, of course. Every reviewer requires patience and courteousness from you. If your diff is taking a long time to review, this ensures quality as the reviewers are actually taking more time to test, look more closely at the diff, etc. They may request changes and you must address them unless they're wrong (but that's very rare). Please don't push the reviewers because you're just making it frustrating for them and no one likes you in the end.

The process of reviewing a diff is necessary for display to show you how to do so or to help you understand what the reviewers are doing and why.

It is always wise to have more than one reviewer to inspect your diff, and we already have Herald configured to make that happen.

Once your diff is launched, it shows up in Diffusion and sends everyone a *Needs Review* message. That will catch the attention of the reviewers that Herald automatically put in your diff. As the diff can't automatically show up at their faces, it will need to take a while for it to get noticed once it's launched. The reviewer then looks over the diff to see if it checks off all of the items in the *Rules For Submitting Code* checklist. If not, then (s)he will request changes for that. The reviewer will also put the diff into action and testing it thoroughly. We trust that the diff submitter did the same. We also trust that the diff submitter takes the effort to make a good and useful diff because many of the reviewers have day jobs and usually won't have enough time to look closely at the submitted diff.

If a change was requested regarding design (the checklist), then please redesign the diff as publicly as possible as this will save a lot of time in the future.

For documentation diffs, Austin (SonosFuer) and Jay (NewbProgrammer101) are the people you hold accountable to. Two rules: read and apply the Documentation Howto, and don't change the scope of the project. If you break at least one of these rules, you have inflicted frustration on the reviewers because they have to waste time by telling you what to do when you could've just followed two extremely basic rules. The reviewers will comply these two rules while reviewing your documentation change.

There is only one person who has direct access to the repositories, and that is NewbProgrammer101, and he is who you go to ask to land the diff if you aren't a *Trusted Member* yet. Only Trusted Members can perform `arc land` once their diff gets accepted. One of the biggest mistakes a Trusted Member makes is landing the diff without it being accepted by all reviewers listed by the diff, this inevitably causes annoyance among everyone if it's actually broken/crappy.

Troubleshooting

This is where you troubleshoot problems regarding pretty much anything related to contributing to the Qub^{3}d Engine Group. For problems specific to Git and/or Arcanist that is not covered in this section, please visit the *#devop* channel on the Qub^{3}d Engine Group's Discord server.

(Problem 1)

You have launched a diff but it's being prevented by an HTTP error 403. Fear not! There's a solution here.

Conclusion

While this may seem like a lot to abide by, it is beneficial for both you and the Qub3d project. It also gets easier the more you contribute.

2.1.3 Documentation Howto

This file is targeted at people who want to document the project but don't know what guidelines to follow.

The development environment for writing Sphinx docs is covered in the Development Environment file.

The Qub3d documentation is written in Sphinx and uses the ReST file format.

guides/environment

Writing a New File

If creating a new documentation file, it must include the following:

Introduction: Description about what the file is dedicated to.

Description: Long description about the subject; it must be very clear, readable and free of grammatical and spelling errors.

When you successfully create an ReST file, you need to put this file in the Table of Contents section of `source/index.rst` and put the file in its respective category. When inserting a file in the index.rst file, do not insert a file extension at the end of the filename. Another rule when putting the file in the Table of Contents section of the index.rst file, is that it must have the path to the file to be only one line long. An example is shown below as to what it should look like in the index.rst file.

```
guides/gettingstarted
guides/documentation
```

Note that the `source/` directory isn't seen here. That is because, to Sphinx, everything inside the source/ directory is considered; whereas, Sphinx is blind to everything else outside the source/ directory other than make-related files such as Makefile. The order of files in the Table of Contents are sorted by importance. The top, being the most important and bottom being the least important.

Format Standards

The guidelines for formatting documentation is as follows:

- Keep the markup simple and try not to make the most out of it.
- Maintain the format as shown below in Qub3d documentation.

ReST Format

If writing documentation in a ".rst" file, you need to apply the following format:

Title:

```
Title/Section
#############
```

Capitalize the major words of the title/section and the pounds "#" should always be the amount of characters the title has. The title must appear only once in the file, and it must appear at the very top.

Sub-sections:

```
Subsection
==========

Sub-subsection
--------------

Sub-sub-subsection
^^^^^^^^^^^^^^^^^^
```

Description: The description's amount of columns can be unlimited; however, one line *must* never consist of two lines (If you have text wrapping enabled, you can easily notice this situation).

Spacing: Titles/subsections/sub-subsections must be separated from each other by one line.

Links: Links should only be shown as hyperlinks, *never* as raw links. As an example, https://qub3d.org needs to be given the name, "Qub3d." Instead of just leaving the link as it is, do the following:

```
`Qub3d <https://qub3d.org>`_
```

This shows "Qub3d" as a hyperlink for qub3d.org.

Note: Links **must** have `https://` instead of `http://`. The only exception is that if the URL doesn't support `HTTPS`.

Tables

Use grid tables instead of list tables. Here's an example:

```
+----------+----------+----------+
| Column 1 | Column 2 | Column 3 |
+----------+----------+----------+
| `Foo()`  | `Bar()`  | fooBar() |
+----------+----------+----------+
```

Rendered in HTML as this:

Column 1	Column 2	Column 3
Foo()	*Bar()*	fooBar()

For more information on ReST grid tables, see the documentation by Sphinx regarding them.

Markdown Format

It is rare to write Markdown files other than the README and the LICENSE. However, there can be a time where a Markdown file gets written. If that's the case, then the following format is required in order to write a .md file for the Qub3d project:

Title:

```
# Title
```

Where the first letter is capitalized and there is only one pound "#" before the title.

Subsection:

```
## Subsection About Stuff
```

Where the subsection always comes after the Title, and all major words are capitalized. Subsections also must be consistent with two pounds "##" before the subsection title.

Further sub-* sections:

Just add another pound "#" to the section's title. An example is demonstrated below.

```
### Sub-subsection
```

Where there's an extra pound "#" in the title. And so forth.

Description: The amount of columns are limited to 60. If you're starting a new subject within the same section, you must have a space between the two subjects. When doing bullet/list points, you must insert a plain text description between the title and the list/bullet points.

Links: Never insert raw links. Instead, give these links a name. For example, the file shouldn't display https://qub3d.org by itself. Instead it should be given the name, "Qub3d." The incorrect method is demonstrated in the following:

```
https://qub3d.org
```

What should've been done is:

```
[Qub3d](https://qub3d.org)
```

This displays "Qub3d" as a hyperlink to https://qub3d.org.

Note: Links **must** have `https://` instead of `http://`. The only exception is that if the URL doesn't support `HTTPS`.

Miscellaneous

The Qub3d development documentation is already hosted on Read The Docs. However, you can always read and compile it into different file formats locally.

Language

To be consistent, the documentation must be written in American English. Abbreviations are all-uppercase such as HTTP instead of Hyper Text Transfer Protocol.

Localization

The documentation is written with the UTF-8 localization format. Please use unicode for documentation when possible.

For more information about the ReST primer, check it out.

Note: This file is *not* a tutorial on ReST and Markdown, rather, it is a tutorial on how the Qub3d documentation should look like/be written.

2.1.4 Development Environment

This section describes the environment needed for an efficient workflow in order to develop for the Qub^{3}d project.

Supported operating systems

- Windows 7/8/8.1/10
- macOS
- Any Linux distribution
- FreeBSD

Development Tools

IDE (Integrated Development Environment): Any multilingual IDE that floats your boat is recommended.

Supported compilers: GCC 5.x+, and LLVM Clang 2.0+.

- OpenGL
- PawLIB
- CMake
- Sphinx

OpenGL is a rendering API that the Rendering API repository requires to even function.

PawLIB is the library that Qub^{3}d requires in order to work. It has the Goldilocks testing suite required by the Qub^{3}d Engine Group.

CMake is the cross-platform compiling software that most of the Qub^{3}d Engine Group's repositories require.

Sphinx is the software used to make documentation cleaner. It is required by the Qub^{3}d Engine Group if you're working on documentation or want to read the documentation locally in a specified file format.

Installation of CMake

For Windows, download the CMake tarball and extract it. Once done, just run the *.exe*.

For macOS, TO BE FILLED IN LATER

For Linux distributions with the Apt package manager, simply type:

```
$ sudo apt-get install cmake
```

And it will do the magic for you. For Gentoo and its derivatives, type:

```
$ sudo emerge -av dev-util/cmake
```

For Linux distributions with the DNF package manager, type:

```
$ sudo dnf install cmake
```

If you have a Linux installation without a package manager like (B)LFS, you can compile from source and install it. The tarball for CMake is here.

For FreeBSD, type:

```
% sudo pkg install cmake
```

Installing Sphinx

The Qub^{3}d documentation is written in Sphinx and uses the ReST file format.

For Windows, you need to install Python if it isn't on your system already, as most people don't have it installed by default. So in order to do that, you need to install it. Once you have *pip* installed, press the Windows key and type `cmd`. When the Command Prompt is running, type:

```
C:\> pip install -U sphinx
```

On macOS, you need to install Homebrew. Once you have done that, launch iTerm and type:

```
$ brew install sphinx-doc
```

To install Sphinx on Linux, as simple as it is, all you need to do on a Debian(-based) distribution is to access the shell and type:

```
$ sudo apt-get install python-sphinx
```

On RHEL/CentOS distributions:

```
$ sudo yum install python-sphinx
```

On Fedora:

```
$ sudo dnf install python-sphinx
```

On Gentoo(-based) systems:

```
$ sudo emerge -av dev-python/sphinx
```

Once you have Sphinx installed, you can simply go to the root of the Qub^{3}d documentation and type `make html` to build for HTML, `make latexpdf` for PDF files, etc. If you want a different file format, just type `make` for it to list the file formats it currently has.

Arcanist and Git

Arcanist

Git

Check out one of our repositories via Diffusion on Phabricator. (You'll want to set up either a VCS Password or SSH Public Key on your Phabricator Settings.)

Working on the Qub^{3}d engine with Git/Arcanist:

On UNIX-like platforms, type from the command line after installing git:

```
$ git clone https://github.com/qub3d/qub3dengine
$ cd qub3dengine/
```

On your local copy of the repository, create a new branch via git checkout -b thenewbranchname

Make your changes, and then send them up:

```
$ git add .
$ git commit -m "<Insert Commit Summary>"
$ arc diff
```

Your code will appear as a new Revision on Differential. It will need to be reviewed and approved by a Trusted member. If they request changes, do the following after making changes:

```
$ git add .
$ git commit -m "<Insert Problem Address>"
$ arc diff
```

Then, the current diff will get updated to address the change requests.

Git commit messages must be:

- Descriptive. (No "Update file.cpp" or "Fix a problem.") You must tell the maintainers *why* you're making this commit in the first place.
- Concise. The hard limit of characters to be on the subject line is 50.
- Capitalized. All subjects must be capitalized. i.e. "Fix all Bugs with Goldilocks implemented"
- Free of spelling errors.
- In present tense. No "-ed" suffixes.

- Professional. No slang words, no incorporating personal opinions, and no grammatical errors. Professional acronyms such as "AFAIK" are allowed.
- Free of useless punctuation. No periods at the end of the subject line, for space is precious if you're trying to keep below 50 characters.
- Easy to understand. Type the commit messages as if you were talking to average person who knows nothing about your intentions.

Here are some good examples:

```
$ git commit -m "Remove unused function intFoo() and add specific parameters \
                 to intFooBar(), named foo with the type int in order for it to \
                 print out a number assigned to that variable."

$ git commit -m "Turn 3 into 3 superscript for all Qub3d names because staff \
                 agreed to using it that way."
```

Git commit bodies are also useful if you're launching a significant/complex diff. The commit bodies' rules are the same as the commit messages but with two more mandatory rules:

- Limit the amount of characters to 72.
- Tell the maintainer how your patch works in an efficiently descriptive manner.

Global Development Environment (Editors)

If your editor of choice is Emacs, your `.emacs` file may need to contain this:

```
(setq make-backup-files nil)
(global-undo-tree-mode)
```

Since our coding style requires that you use tab instead of spaces, you'll need to do `C-q <TAB>` to do that.

If it's Vim, then the `.vimrc` may need to look like this:

```
syntax on
set nocompatible
filetype plugin indent on
set noswapfile
set hidden
set nobackup
set nowb
set autoindent
set smartindent
set smarttab
set tabstop=4
set linebreak
set expandtab
```

For nano, then `.nanorc` should look like this:

```
set linenumbers
# Disable `set nowrap`
```

Workflow For The Qub^{3}d Engine

You'll need a folder to stack the engine and the library dependencies.

`qub3d-libdeps` will need to be paralell to the engine's repository. On first-time setup, you just need to run these commands:

```
$ mkdir qub3d
$ cd qub3d
$ git clone https://phab.qub3d.tk/source/qub3d-libdeps.git
$ git clone https://phab.qub3d.tk/source/qub3d-engine.git
$ cd qub3d-libdeps
$ make all
$ cd ../qub3d-engine
# Do whatever you're planning to do with the engine...
$ mkdir build && cd build # At the root of the engine repository after configuring␣
↪default.config...
$ cmake ..
$ make -j$(nproc)
# If build passed successfully...
$ cd ..
$ git add .
$ git commit -m "<Summary of what you did>"
$ arc diff
```

Workflow For The Client

You'll need a similar setup as the engine except that the repository is `client`.

First-time setup commands (after setting up the engine):

```
# Assuming you already have cloned the engine and library dependencies...
$ cd qub3d
$ git clone https://phab.qub3d.tk/source/client.git
$ cd sandblox-client
# Do what you planned to do with the client...
$ mkdir build && cd build # At the root of repository after making configurations to␣
↪default.config...
$ cmake ..
$ make -j$(nproc)
# If it built properly...
$ ../bin/client # Check for any runtime failures...
# If runtime passed...
$ cd ..
$ git add .
$ git commit -m "<Summary of what you did>"
$ arc diff
```

2.1.5 Server Configuration

Note: This setup configuration document is soon going to go under "renovation", due to a re-configuration and change in workflow on the servers.

Introduction

We've documented most details relating to how we built our server. This serves (a) as a reference for ongoing maintenance, (b) a training resource for anyone wanting to learn more about LAMP servers, and (c) a way for others to provide feedback on our server architectures.

Note: Of course, these guides are provided without warranty.

If you find errors, omissions, typos, or other problems, or otherwise want to make a suggestion, please file a bug report on Phabricator.

Credits

This server is primarily maintained by Jason C. McDonald (CodeMouse92). This guide is based in part on the Server Configuration documentation by MousePaw Media.

Initial Setup

We are using a Linode 2048 with an Ubuntu 16.04 disk instance deployed from the `Rebuild` pane on the Linode dashboard.

Note: We already added the *qub3d.org* domain name in the Linode DNS Manager, and configured all the subdomains we're using under the *A/AAAA Records* section of that panel.

Using LISH, we will log in as the root user.

Setting Hostname and Timezone

We'll start by defining the name of our host.

```
$ echo "qub3d" > /etc/hostname
$ hostname -F /etc/hostname
$ vim /etc/hosts
```

In that file, add `qub3d` to the end of the second line.

Save and quit.

Now we set the timezone.

```
$ dpkg-reconfigure tzdata
```

Use the arrow and ENTER keys to set your timezone.

Reboot the system.

SOURCE: Getting Started with Linode (Linode)

Defining a Non-Root User

Next, we'll use the root account to set up the regular user account.

```
$ adduser itdude
```

Define the password for the new user, and other information if desired. Then, we add the user to the `sudo` group.

```
$ usermod -aG sudo itdude
$ groups itdude
```

The second command will list all of the groups `itdude` is in. Ensure it includes the `sudo` group.

Repeat this for all the users you want on the system.

Finally, we'll log out of root…

```
$ logout
```

And log in as `itdude`. Once logged in, we can test out the user's sudo abilities by running…

```
$ sudo echo Hi
```

If it works, we have set up sudo properly.

Updates

Before we continue, we need to check for any updates.

```
$ sudo apt update
$ sudo apt full-upgrade
```

Set Up Longview

It will be beneficial for us to use the free plan for Longview, a service provided through Linode. Let's install that now.

On the Linode Manager web interface, go to Longview and add a new client. After a moment, a white box will pop up with a command to run in the server terminal.

It will take a few minutes for Longview to start working, so just close the window on the Linode Manager.

Setup LAMP

Apache2

We'll start by setting up Apache2.

```
$ sudo apt install apache2
```

Next, we'll edit the configuration file to turn off `KeepAlive`, as that uses up extra memory (and we don't have that much to spare).

```
KeepAlive Off
```

Save and close.

Next, we'll change the settings for the `mpm_prefork` module.

```
$ sudo vim /etc/apache2/mods-available/mpm-prefork.conf
```

Set the file to the following…

```
<IfModule mpm_prefork_module>
        StartServers            2
        MinSpareServers         5
        MaxSpareServers         10
        MaxRequestWorkers       39
        MaxConnectionsPerChild  3000
</IfModule>
```

Save and close. Now we'll enable the prefork module.

```
$ sudo a2dismod mpm_event
$ sudo a2enmod mpm_prefork
$ sudo systemctl restart apache2
```

Next, we will add our user to the `wwww-data` group, which will be helpful for permissions, and we'll create the directory we'll be using for all our websites.

```
$ sudo usermod -aG www-data webster
$ cd /opt
$ sudo mkdir html
$ sudo chown webster:www-data html
$ sudo chmod 775 html
```

We need to tell Apache2 to read this directory.

```
$ sudo vim /etc/apache2/apache2.conf
```

Scroll down to the section with all the directories, and add this entry:

```
<Directory /opt/html/>
        Options FollowSymLinks
        AllowOverride All
        Require all granted
</Directory>
```

Save and close, and then restart Apache2.

```
$ sudo systemctl restart apache2
```

Browse to the web server using whatever address is most convenient, and ensure the Apache2 default page is appearing.

Note: This is a good time to go to *qub3d.org* and make sure you see the Apache2 default page!

MySQL

Now we will set up our database software.

```
$ sudo apt install mysql-server
```

When prompted, set the root password.

PHP

We'll be using PHP 7.2, which is necessary to run Phabricator.

```
$ sudo apt install software-properties-common
$ sudo apt install php7.2 php-pear libapache2-mod-php7.2 php7.2-cli php7.2-common␣
↪php7.2-curl php7.2-dev php7.2-gd php-gettext php7.2-json php7.2-mbstring php7.2-
↪opcache php7.2-readline php7.2-xml php7.2-mysql
$ sudo vim /etc/php/7.2/apache2/php.ini
```

Edit the contents of that file so the following lines match the given values:

```
max_input_time = 30
error_reporting = E_COMPILE_ERROR | E_RECOVERABLE_ERROR | E_ERROR | E_CORE_ERROR
error_log = /var/log/php/error.log
```

Create the log directory for PHP, and give ownership to the Apache2 system user. Finally, restart Apache2 to start using the changes.

```
$ sudo mkdir /var/log/php
$ sudo chown www-data /var/log/php
$ sudo a2enmod php7.2
$ sudo systemctl restart apache2
```

SOURCE: Install LAMP on Ubuntu 16.04 (Linode)

Server Hardening

Firewall

We'll enable our firewall, and allow SSH and HTTP(S) through.

```
$ sudo ufw enable

# Open for SSH
$ sudo ufw allow 22

# Open for HTTP
$ sudo ufw allow 80

# Open for HTTPS
$ sudo ufw allow 443
```

SSH Security

We need to lock down SSH for further security.

```
$ sudo vim /etc/ssh/sshd_config
```

Edit the file so the following lines have the given settings:

```
PermitRootLogin no
PasswordAuthentication no
AuthorizedKeysFile      %h/.ssh/authorized_keys
```

Warning: Don't actually disable PasswordAuthentication until you have setup and logged in with a key instead!

Save and close the file, and then run. . .

```
$ sudo systemctl restart sshd
```

Secure Shared Memory

```
$ sudo vim /etc/fstab
```

At the bottom of the file, add the lines:

```
# Secure shared memory
tmpfs /run/shm tmpfs defaults,noexec,nosuid 0 0
```

Save and close the file. The changes will take effect on next reboot.

Lock Down `sudo` Privilege

We'll limit `sudo` privileges to only users in the `admin` group.

```
$ sudo groupadd admin
$ sudo usermod -a -G admin <YOUR ADMIN USERNAME>
$ sudo dpkg-statoverride --update --add root admin 4750 /bin/su
```

Harden Network with `sysctl` Settings

```
$ sudo vi /etc/sysctl.conf
```

Edit the file, uncommenting or adding the following lines.:

```
# IP Spoofing protection
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1

# Ignore ICMP broadcast requests
net.ipv4.icmp_echo_ignore_broadcasts = 1

# Disable source packet routing
net.ipv4.conf.all.accept_source_route = 0
net.ipv6.conf.all.accept_source_route = 0
net.ipv4.conf.default.accept_source_route = 0
net.ipv6.conf.default.accept_source_route = 0

# Ignore send redirects
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0

# Block SYN attacks
net.ipv4.tcp_syncookies = 1
```

(continues on next page)

(continued from previous page)

```
net.ipv4.tcp_max_syn_backlog = 2048
net.ipv4.tcp_synack_retries = 2
net.ipv4.tcp_syn_retries = 5

# Log Martians
net.ipv4.conf.all.log_martians = 1
net.ipv4.icmp_ignore_bogus_error_responses = 1

# Ignore ICMP redirects
net.ipv4.conf.all.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0

# Ignore Directed pings
net.ipv4.icmp_echo_ignore_all = 1
```

Finally, reload `sysctl`. If there are any errors, fix the associated lines.

```
$ sudo sysctl -p
```

Prevent IP Spoofing

To prevent IP spoofing, we edit `/etc/hosts`.

```
$ sudo vim /etc/host.conf
```

Add or edit the following lines.

```
order bind,hosts
nospoof on
```

Harden PHP

```
$ sudo vim /etc/php/7.2/apache2/php.ini
```

Add or edit the following lines and save.:

```
disable_functions = exec,system,shell_exec,passthru
register_globals = Off
expose_php = Off
display_errors = Off
track_errors = Off
html_errors = Off
magic_quotes_gpc = Off
mail.add_x_header = Off
session.name = NEWSESSID
```

Restart the Apache2 server and make sure it still works.

```
$ sudo systemctl restart apache2
```

Harden Apache2

Edit the Apache2 security configuration file…

```
$ sudo vim /etc/apache2/conf-available/security.conf
```

Change or add the following lines:

```
ServerTokens Prod
ServerSignature Off
TraceEnable Off
FileETag None
```

Restart the Apache2 server and make sure it still works.

```
$ sudo systemctl restart apache2
```

Setup ModSecurity

First, install the necessary dependencies. We'll also need to create a symbolic link to work around a bug on 64-bit systems. Finally, we'll install the package itself.

```
$ sudo apt install libxml2 libxml2-dev libxml2-utils libaprutil1 libaprutil1-dev
$ sudo ln -s /usr/lib/x86_64-linux-gnu/libxml2.so.2 /usr/lib/libxml2.so.2
$ sudo apt install libapache2-mod-security2
```

Now we'll copy the default configuration and edit it.

```
$ sudo mv /etc/modsecurity/modsecurity.conf-recommended /etc/modsecurity/modsecurity.
↪conf
$ sudo vim /etc/modsecurity/modsecurity.conf
```

Add and edit the lines:

```
SecRuleEngine On
SecServerSignature FreeOSHTTP
SecRequestBodyLimit 33554432
SecRequestBodyInMemoryLimit 33554432
```

Those last two lines define the maximum upload size in *bytes*. At the moment, we're setting the limit to **32 MB**.

Now we download the latest OWASP security rules.

```
$ cd /etc/modsecurity
$ sudo apt install git
$ sudo git clone https://github.com/SpiderLabs/owasp-modsecurity-crs.git
$ cd owasp-modsecurity-crs
$ sudo cp crs-setup.conf.example crs-setup.conf
$ cd rules
$ sudo cp REQUEST-900-EXCLUSION-RULES-BEFORE-CRS.conf.example REQUEST-900-EXCLUSION-
↪RULES-BEFORE-CRS.conf
$ sudo cp RESPONSE-999-EXCLUSION-RULES-AFTER-CRS.conf.example RESPONSE-999-EXCLUSION-
↪RULES-AFTER-CRS.conf
```

Edit the configuration for the ModSecurity Apache module…

```
$ sudo vim /etc/apache2/mods-available/security2.conf
```

Add the following lines just below the other `IncludeOptional` directive.

```
IncludeOptional /etc/modsecurity/owasp-modsecurity-crs/crs-setup.conf
IncludeOptional /etc/modsecurity/owasp-modsecurity-crs/rules/*.conf
```

Enable the modules and restart Apache2, ensuring that it still works.

```
$ sudo a2enmod headers
$ sudo a2enmod security2
$ sudo systemctl restart apache2
```

Finally, to make sure it works, go to `http://qub3d.org/?param="><script>alert(1);</script>`. Check `/var/log/apache2/error.log` for an error report from `mod_security`. If one is there, the configuration worked!

SOURCE: owasp-modsecurity-crs INSTALL

Setup ModEvasive

To harden against DDoS attacks, we'll install ModEvasive.

```
$ sudo apt install libapache2-mod-evasive
```

For the `Postfix Configuration`, select `Local Only` and use the default FQDN (`ubuntu.members.linode.com`). We **will** be changing this later.

Now we'll create the log directory for ModEvasive and set its permissions accordingly.

```
$ sudo mkdir /var/log/mod_evasive
$ sudo chown www-data:www-data /var/log/mod_evasive/
```

Edit the ModEvasive configuration file...

```
$ sudo vim /etc/apache2/mods-available/evasive.conf
```

Modify the file to match the following.

```
<ifmodule mod_evasive20.c>
   DOSHashTableSize 3097
   DOSPageCount  15
   DOSSiteCount  50
   DOSPageInterval 1
   DOSSiteInterval  1
   DOSBlockingPeriod  10

   DOSLogDir   /var/log/mod_evasive
   DOSEmailNotify  itdude@ubuntu.members.linode.com
   DOSWhitelist   127.0.0.1
</ifmodule>
```

There is also a bug reported for Ubuntu 12.04 regarding email. I don't know if it's fixed, but the workaround doesn't hurt anything anyway.

```
$ sudo ln -s /etc/alternatives/mail /bin/mail/
```

Enable the modules and restart Apache2, ensuring that it still works.

```
$ sudo a2enmod evasive
$ sudo systemctl restart apache2
```

Read the Docs

Setup DenyHosts

DenyHosts blocks SSH attacks and tracks suspicious IPs.

```
$ sudo apt install denyhosts
$ sudo vim /etc/denyhosts.conf
```

Edit the following lines.:

```
ADMIN_EMAIL = itdude@localhost
SMTP_FROM = DenyHosts
SYSLOG_REPORT = YES
```

Setup Fail2Ban

Fail2Ban does much the same things as DenyHosts, but its coverage includes Apache, FTP, and other things.

```
$ sudo apt install fail2ban
$ sudo vim /etc/fail2ban/jail.conf
```

To turn on various "jails", scroll down to the `# JAILS` section. Place `enabled = true` under each jail name you want turned on. This is the list of jails we enabled:

- sshd
- sshd-ddos
- apache-auth
- apache-badbots
- apache-noscript
- apache-overflows
- apache-nohome
- apache-botsearch
- apache-fakegooglebot
- apache-modsecurity
- apache-shellshock

We also need to modify a file for `apache-fakegooglebot` to work around a bug. If you run `python -V` and it reports a version of Python2 (which it almost certainly will), run. . .

```
$ sudo vim /etc/fail2ban/filter.d/ignorecommands/apache-fakegooglebot
```

Change the first line to `#!/usr/bin/python3`, and then save and close.

SOURCE: Fail2Ban fakegooglebot Jail Bug (Shell and Co)

Finally, restart the fail2ban process.

```
$ sudo systemctl restart fail2ban
```

Setup PSAD

```
$ sudo apt install psad
$ sudo vim /etc/psad/psad.conf
```

Change “EMAIL_ADDRESS” to `itdude@localhost` and “HOSTNAME” to `qub3d`.

```
$ sudo iptables -A INPUT -j LOG
$ sudo iptables -A FORWARD -j LOG
$ sudo ip6tables -A INPUT -j LOG
$ sudo ip6tables -A FORWARD -j LOG
$ sudo psad -R
$ sudo psad --sig-update
$ sudo psad -H
$ sudo psad --Status
```

When you run that last command, it may whine about not finding a pidfile. It appears we can ignore that error.

We also need to tweak Fail2Ban so that it doesn’t start up before `psad` does. Otherwise, `psad` won’t be able to log correctly.

```
$ sudo vim /lib/systemd/system/fail2ban.service
```

In that file, add `ufw.service` and `psad.service` to the `After=` directive, so it looks something like this:

```
After=network.target iptables.service firewalld.service ufw.service psad.service
```

Save and close, and then reload the daemons for systemctl and restart fail2ban.

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart fail2ban
```

Now we need to adjust the UFW settings.

```
$ sudo ufw logging high
$ sudo vim /etc/ufw/before.rules
```

Add the following lines before the final commit message.:

```
-A INPUT -j LOG
-A FORWARD -j LOG
```

Save and close. Repeat this with `before6.rules`. Then, restart ufw and reload PSAD.

SOURCE: PSAD Is Giving a Firewall Setup Warning (Ubuntu Forums)

```
$ sudo systemctl restart ufw
$ sudo psad --fw-analyze
```

Restart the computer, and ensure PSAD isn't sending any system emails complaining about the firewall configuration. (Check system email by running `$ mail`).

Rootkit Checks

We use two different rootkit checkers.

```
$ sudo apt install rkhunter chkrootkit
```

We have a script set up on the system that runs the following...

```
#!/bin/bash
sudo ckrootkit
sudo rkhunter --update
sudo rkhunter --propupd
sudo rkhunter --check --cronjob -l
echo "Rootkit Check Done!"
```

Miscellaneous

These are a few other useful programs.

```
$ sudo apt install nmap logwatch libdate-manip-perl apparmor apparmor-profiles tiger␣
↪clamav

# Ensure apparmor is working.
$ sudo apparmor_status
```

To use logwatch, run...

```
$ sudo logwatch | less
```

To scan for vulnerabilites with Tiger, run...

```
$ sudo tiger
$ sudo less /var/log/tiger/security.report.*
```

Let's Encrypt Certificates

We'll install the Let's Encrypt Certbox, and then create our server certificates. While we can *technically* install the `letsencrypt` package, it's out of date compared to `certbot-auto`.

```
$ cd /opt
$ sudo mkdir certbot
$ cd certbot
$ sudo wget https://dl.eff.org/certbot-auto
$ sudo chmod a+x certbot-auto
```

When we get our certificates, we want all the domains to point to the same webroot. Thus, we need to turn on our default site, and turn off the others. Of course, if we're following this guide from bare-metal (nothing installed), then there are no other sites to disable.

Now we'll get our certificates.

```
$ sudo /opt/certbot/certbot-auto certonly -a webroot --webroot-path /var/www/html -d␣
↪qub3d.org -d www.qub3d.org -d blog.qub3d.org -d discourse.qub3d.org -d docs.qub3d.
↪org -d files.qub3d.org -d mail.qub3d.org -d pad.qub3d.org -d phab.qub3d.org -d␣
↪webmail.qub3d.org
```

Note: If you're needing to add a domain or subdomain to an existing certificate, use the command above, and include the `--expand` flag as the first argument after `certonly`.

Follow the instructions on the screen to complete the process of getting the certificates. If successful, they can be found (visible only as root) in `/etc/letsencrypt/live/qub3d.org`.

We also need to add a special configuration file that Apache2 will use with the certificates.

```
$ sudo vim /etc/letsencrypt/options-ssl-apache.conf
```

Set the contents of that file to the following.

```
# Baseline setting to Include for SSL sites
SSLEngine on

# Intermediate configuration, tweak to your needs
SSLProtocol all -SSLv2 -SSLv3
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-
↪AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-DSS-
↪AES128-GCM-SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-
↪SHA256:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-
↪ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-
↪SHA256:DHE-RSA-AES128-SHA:DHE-DSS-AES128-SHA256:DHE-RSA-AES256-SHA256:DHE-DSS-
↪AES256-SHA:DHE-RSA-AES256-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-
↪SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:AES:CAMELLIA:DES-CBC3-SHA:!aNULL:!eNULL:!
↪EXPORT:!DES:!RC4:!MD5:!PSK:!aECDH:!EDH-DSS-DES-CBC3-SHA:!EDH-RSA-DES-CBC3-SHA:!KRB5-
↪DES-CBC3-SHA
SSLHonorCipherOrder on
SSLCompression off

SSLOptions +StrictRequire

# Add vhost name to log entries:
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" vhost_
↪combined
LogFormat "%v %h %l %u %t \"%r\" %>s %b" vhost_common

#CustomLog /var/log/apache2/access.log vhost_combined
#LogLevel warn
#ErrorLog /var/log/apache2/error.log

# Always ensure Cookies have "Secure" set (JAH 2012/1)
#Header edit Set-Cookie (?i)^(.*)(;\s*secure)??((\s*;)?(.*)) "$1; Secure$3$4"
```

Save and close.

Renewal Scripts

There are a few things we'll need to do every time the certificate is renewed. Perhaps most important, we need to copy the certs over to a new folder and change their permissions, so they can be used by various parts of our server setup.

We'll start by creating a special group for accessing certificates.

```
$ sudo groupadd certs
```

Now we'll create a directory for the copied certs, and make the script file.

```
$ cd /etc/apache2
$ sudo mkdir certs
$ cd certs
$ sudo vim renewcert_pre
```

Put the following contents into that file. Comment out the lines regarding the sites you do not have. Be sure to uncomment them later!

```
#!/bin/bash

a2dissite grav pad phab webmail
a2ensite 000-default
systemctl reload apache2
```

Save and close. Now, let's create the post script.

```
$ sudo vim renewcert_post
```

Put the following contents into that file. Comment out the lines regarding the sites you do not have. Be sure to uncomment them later!

```
#!/bin/bash

# Work out of the certificate's working directory.
cd /etc/apache2/certs

# Copy the certificates over and update their permissions.
cp /etc/letsencrypt/live/qub3d.org/*.pem ./
chgrp certs ./*.pem
chmod u=rw,g=r,o= ./*.pem

# Restore the sites and restart critical services which use this.
a2dissite 000-default
a2ensite grav phab pad webmail
systemctl restart apache2
```

Save and close. Change the script permissions so it can only be read, accessed, and run by its owner and group (both root).

```
$ sudo chmod 770 renewcert_pre
$ sudo chmod 770 renewcert_post
```

Finally, we'll test the configuration.

```
$ sudo /opt/certbot/certbot-auto renew --dry-run --pre-hook "/etc/apache2/certs/
↪renewcert_pre" --post-hook "/etc/apache2/certs/renewcert_post"
```

Note: The expansion script is below.

```
$ sudo /opt/certbot/certbot-auto certonly --expand -a webroot --webroot-path /var/www/
↪html -d qub3d.org -d www.qub3d.org -d blog.qub3d.org -d discourse.qub3d.org -d docs.
↪qub3d.org -d files.qub3d.org -d mail.qub3d.org -d pad.qub3d.org -d phab.qub3d.org -
↪d webmail.qub3d.org --pre-hook "/etc/apache2/certs/renewcert_pre" --post-hook "/etc/
↪apache2/certs/renewcert_post"
```

Scheduling Auto-Renewal

Now we need to schedule the autorenewal task.

```
$ sudo mkdir -p /opt/scripts/root_scripts
$ cd /opt/scripts
$ sudo chown root:root root_scripts
$ sudo chmod 770 root_scripts
$ sudo su
$ cd /opt/scripts/root_scripts
$ vim renewcert
```

Set the contents of that file to the following…

```
#!/bin/bash
/opt/certbot/certbot-auto renew --pre-hook "/etc/apache2/certs/renewcert_pre" --post-
↪hook "/etc/apache2/certs/renewcert_post"
```

Save and close. Then run…

```
$ exit
$ sudo crontab -e
```

Add the following line to the end:

```
32 3 * * * /opt/scripts/root_scripts/renewcerts
```

This will run the renewal script once a day at 3:32am. (Let's Encrypt asks that a random time be used by each user, to spread out server load.)

Server Controls

PHPMyAdmin

```
$ sudo apt-get update
$ sudo apt-get install phpmyadmin
```

On the configuration dialog, select `apache2` by selecting it and tapping `Space`. Enter an application password (different from the MySQL root password) and confirm it.

Edit the configuration for PHP, to force HTTPS.

```
$ sudo vim /etc/phpmyadmin/config.inc.php
```

Add the following line to the bottom of that file.

```
$cfg['ForceSSL'] = true;
```

Save and close.

Now enable two necessary PHP modules and restart Apache2.

```
$ sudo phpenmod mcrypt
$ sudo phpenmod mbstring
$ sudo systemctl restart apache2
```

Validate that you can `https://<serveraddress>/phpmyadmin`.

> **Warning:** You may need to disable the Apache2 module `security2` before you can access PHPMyAdmin. Otherwise, it throws an internal 404. We're not sure why. To fix the problem, run `sudo a2dismod security2` and restart the Apache2 service.

Control Access Switch

For security reasons, we want to be able to turn on and off controls like PHPMyAdmin using a script.

```
$ sudo vim /opt/scripts/sys_scripts/controls
```

The contents of that file are as follows.

```
#!/bin/bash

set -e

case $1 in
'on')
    sudo a2dismod security2
    sudo a2enconf phpmyadmin
    sudo systemctl restart apache2
    echo "Admin control panels are turned ON."
    ;;
'off')
    sudo a2enmod security2
    sudo a2disconf phpmyadmin
    sudo systemctl restart apache2
    echo "Admin control panels are turned OFF."
    ;;
*)
    echo "You must specify 'on' or 'off'."
    exit 1
    ;;
esac
```

Email Server

Postfix Setup

These instructions assume you've already configured your DNS correctly, according to Linode's instructions. (See the SOURCE at the bottom of this section.)

We'll start by installing the necessary software.

```
$ sudo apt install postfix postfix-mysql dovecot-core dovecot-imapd dovecot-pop3d␣
↪dovecot-lmtpd dovecot-mysql
$ sudo dpkg-reconfigure postfix
```

This time, select the following options:

- Internet Site
- `qub3d.org`
- `webster`
- `$myhostname, localhost, ubuntu.members.linode.com, qub3d, qub3d.org`
- Force Synchronous Updates? **No**
- (Default)
- `0`
- `+`
- `all`

Postfix is now configured.

MySQL Setup

Now we can set up the MySQL database. Replace `password` with a unique password.

```
$ mysqladmin -u root -p create mailserver
$ mysql -u root -p mailserver
```

```
GRANT SELECT ON mailserver.* TO 'postmaster'@'localhost' IDENTIFIED BY 'password';
FLUSH PRIVILEGES;
exit
```

We'll be setting up the tables in the next step.

Postfix Admin

Let's install a control panel for managing Postfix. We want the latest version from Github, instead of the outdated one in the repos. This will also define our tables.

```
$ cd /opt
$ sudo wget https://github.com/postfixadmin/postfixadmin/archive/postfixadmin-3.0.2.
↪zip
$ sudo unzip postfixadmin-3.0.2.zip
$ sudo mv postfixadmin-postfixadmin-3.0.2 postfixadmin
$ sudo mkdir -p /opt/postfixadmin/templates_c
$ sudo chown -R jason:www-data postfixadmin
```

Now we need to either add or edit a configuration file for postfixadmin in Apache2.

```
$ sudo vim /etc/apache2/conf-available/postfixadmin.conf
```

Set the contents of that file to…

```
Alias /postfixadmin /opt/postfixadmin
```

Save and close, and then edit Apache2's main configuration…

```
$ sudo vim /etc/apache2/apache2.conf
```

Adding the following.

```
<Directory /opt/postfixadmin>
    AllowOverride None
    Require all granted
</Directory>
```

Save and close, and then load our configuration and reload Apache2.

```
$ sudo a2enconf postfixadmin
$ sudo systemctl reload apache2
```

When prompted, create a unique password for the application to access MySQL.

Now open the database configuration…

```
$ sudo vim /opt/postfixadmin/config.inc.php
```

In that file, change the following lines to match the given values. Of course, be sure to replace `PASSWORD` with the actual database password…

```
$CONF['database_user'] = 'postfixadmin';
$CONF['database_password'] = 'PASSWORD';
$CONF['database_name'] = 'mailserver';
```

You may also want to change the welcome message sent to new users. The following is the setting for our setup.

```
$CONF['welcome_text'] = <<<EOM
Weclome to Qub3d Webmail!

Your new account is ready and raring to go. It is compatible with your favorite email␣
↪client,
or you can just use our handy-dandy webmail. Of course, if you're reading this now,␣
↪you know
one of those two things.

Any bugs or issues you encounter should be reported on https://phab.qub3d.org/

Enjoy!

- Your Friendly IT Dude
EOM;
```

Save and close.

Then, using PHPMyAdmin, grant the `postfixadmin` user full permissions over the `mailserver` database.

Finally, navigate to `https://qub3d.org/postfixadmin/setup.php` and follow the instructions to set the setup password and proceed with setup.

Note: At this point, you should use Postfixadmin to set up your domain(s) and at least one mailbox, `webmaster@qub3d.org`.

You will also want to add `postfixadmin` to your script for toggling admin controls.

```
$ sudo vim /opt/scripts/sys_scripts/controls
```

The edited script is below.

```
#!/bin/bash

set -e

case $1 in
'on')
    sudo a2dismod security2
    sudo a2enconf phpmyadmin
    sudo a2enconf postfixadmin
    sudo systemctl restart apache2
    echo "Admin control panels are turned ON."
    ;;
'off')
    sudo a2enmod security2
    sudo a2disconf phpmyadmin
    sudo a2disconf postfixadmin
    sudo systemctl restart apache2
    echo "Admin control panels are turned OFF."
    ;;
*)
    echo "You must specify 'on' on 'off'."
    exit 1
    ;;
```

Save and close.

Postfix Configuration, Round Two

We need to further adjust Postfix. We'll make a backup of the configuration file, and then start making our edits.

```
$ sudo cp /etc/postfix/main.cf /etc/postfix/main.cf.orig
$ sudo vim /etc/postfix/main.cf
```

Edit the file to match the following:

```
# See /usr/share/postfix/main.cf.dist for a commented, more complete version

# Debian specific:  Specifying a file name will cause the first
# line of that file to be used as the name.  The Debian default
# is /etc/mailname.
#myorigin = /etc/mailname

smtpd_banner = $myhostname ESMTP $mail_name (Ubuntu)
biff = no

# appending .domain is the MUA's job.
append_dot_mydomain = no

```

(continues on next page)

(continued from previous page)

```
# Uncomment the next line to generate "delayed mail" warnings
#delay_warning_time = 4h

readme_directory = no

# TLS parameters
smtpd_tls_cert_file=/etc/apache2/certs/cert.pem
smtpd_tls_CAfile=/etc/apache2/certs/chain.pem
smtpd_tls_CApath=/etc/apache2/certs
smtpd_tls_key_file=/etc/apache2/certs/privkey.pem
smtpd_use_tls=yes
#smtpd_tls_session_cache_database = btree:${data_directory}/smtpd_scache
#smtp_tls_session_cache_database = btree:${data_directory}/smtp_scache

# Enabling SMTP for authenticated users, and handing off authentication to Dovecot
smtpd_sasl_type = dovecot
smtpd_sasl_path = private/auth
smtpd_sasl_auth_enable = yes

smtpd_recipient_restrictions =
        permit_sasl_authenticated,
        permit_mynetworks,
        reject_unauth_destination

# See /usr/share/doc/postfix/TLS_README.gz in the postfix-doc package for
# information on enabling SSL in the smtp client.

#smtpd_relay_restrictions = permit_mynetworks permit_sasl_authenticated defer_unauth_
↪destination
myhostname = qub3d.qub3d.org
mydomain = mail.qub3d.org
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
myorigin = /etc/mailname
mydestination = localhost
relayhost =
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128
mailbox_size_limit = 0
recipient_delimiter = +
inet_interfaces = all

# Handing off local delivery to Dovecot's LMTP, and telling it where to store mail
virtual_transport = lmtp:unix:private/dovecot-lmtp

#Virtual domains, users, and aliases
virtual_mailbox_domains = mysql:/etc/postfix/mysql-virtual-mailbox-domains.cf
virtual_mailbox_maps = mysql:/etc/postfix/mysql-virtual-mailbox-maps.cf
virtual_alias_maps = mysql:/etc/postfix/mysql-virtual-alias-maps.cf,
        mysql:/etc/postfix/mysql-virtual-email2email.cf

#default_transport = smtp
#relay_transport = smtp
#inet_protocols = all
```

Save and close. Now we need to edit four files. In each, be sure to replace `mailuserpass` with the password for the `postmaster` MySQL user we set earlier.

```
$ sudo vim /etc/postfix/mysql-virtual-mailbox-domains.cf
```

Set the contents to:

```
user = postmaster
password = mailuserpass
hosts = 127.0.0.1
dbname = mailserver
query = SELECT 1 FROM domain WHERE domain='%s'
```

Save and close, then run…

```
$ sudo vim /etc/postfix/mysql-virtual-mailbox-maps.cf
```

Set the contents to:

```
user = postmaster
password = mailuserpass
hosts = 127.0.0.1
dbname = mailserver
query = SELECT 1 FROM mailbox WHERE username='%s'
```

Save and close, then run…

```
$ sudo vim /etc/postfix/mysql-virtual-alias-maps.cf
```

Set the contents to:

```
user = postmaster
password = mailuserpass
hosts = 127.0.0.1
dbname = mailserver
query = SELECT goto FROM alias WHERE address='%s'
```

Save and close, then run…

```
$ sudo vim /etc/postfix/mysql-virtual-email2email.cf
```

Set the contents to:

```
user = postmaster
password = mailuserpass
hosts = 127.0.0.1
dbname = mailserver
query = SELECT username FROM mailbox WHERE username='%s'
```

Save and close. Then we'll restart postfix and test things. Note the comments below, displaying the expected output.

```
$ sudo systemctl restart postfix
$ sudo postmap -q qub3d.org mysql:/etc/postfix/mysql-virtual-mailbox-domains.cf
# EXPECTED OUTPUT: 1
$ sudo postmap -q webmaster@qub3d.org mysql:/etc/postfix/mysql-virtual-mailbox-maps.cf
# EXPECTED OUTPUT: 1
$ sudo postmap -q webmaster@qub3d.org mysql:/etc/postfix/mysql-virtual-alias-maps.cf
# EXPECTED OUTPUT: webmaster@qub3d.org
```

If we got the expected outputs, we're doing great! Now we need to edit another configuration file.

```
$ sudo cp /etc/postfix/master.cf /etc/postfix/master.cf.orig
$ sudo vim /etc/postfix/master.cf
```

Uncomment the two lines starting with `submission` and `smtps`, as well as the block of lines starting with `-o` after each. Thus, the first part of that file should look like this:

```
#
# Postfix master process configuration file.  For details on the format
# of the file, see the master(5) manual page (command: "man 5 master").
#
# Do not forget to execute "postfix reload" after editing this file.
#
# ==========================================================================
# service type  private unpriv  chroot  wakeup  maxproc command + args
#               (yes)   (yes)   (yes)   (never) (100)
# ==========================================================================
smtp      inet  n       -       -       -       -       smtpd
#smtp      inet  n       -       -       -       1       postscreen
#smtpd     pass  -       -       -       -       -       smtpd
#dnsblog   unix  -       -       -       -       0       dnsblog
#tlsproxy  unix  -       -       -       -       0       tlsproxy
submission inet n       -       -       -       -       smtpd
  -o syslog_name=postfix/submission
  -o smtpd_tls_security_level=encrypt
  -o smtpd_sasl_auth_enable=yes
  -o smtpd_client_restrictions=permit_sasl_authenticated,reject
  -o milter_macro_daemon_name=ORIGINATING
smtps     inet  n       -       -       -       -       smtpd
  -o syslog_name=postfix/smtps
  -o smtpd_tls_wrappermode=yes
  -o smtpd_sasl_auth_enable=yes
  -o smtpd_client_restrictions=permit_sasl_authenticated,reject
  -o milter_macro_daemon_name=ORIGINATING
```

Save and close. Then we'll fix some permissions, restart postfix, and move on to the next piece of the email server system.

```
$ sudo chmod -R o-rwx /etc/postfix
$ sudo systemctl restart postfix
```

Dovecot

Let's setup the other half of the mail system - Dovecot. First, we want to make copies of our configuration files before we start changing stuff, just in case.

```
$ sudo cp /etc/dovecot/dovecot.conf /etc/dovecot/dovecot.conf.orig
$ sudo cp /etc/dovecot/conf.d/10-mail.conf /etc/dovecot/conf.d/10-mail.conf.orig
$ sudo cp /etc/dovecot/conf.d/10-auth.conf /etc/dovecot/conf.d/10-auth.conf.orig
$ sudo cp /etc/dovecot/dovecot-sql.conf.ext /etc/dovecot/dovecot-sql.conf.ext.orig
$ sudo cp /etc/dovecot/conf.d/10-master.conf /etc/dovecot/conf.d/10-master.conf.orig
$ sudo cp /etc/dovecot/conf.d/10-ssl.conf /etc/dovecot/conf.d/10-ssl.conf.orig
$ sudo vim /etc/dovecot/dovecot.conf
```

Edit that file, adding the last line in the sample below in the indicated position:

```
## Dovecot configuration file

# If you're in a hurry, see http://wiki2.dovecot.org/QuickConfiguration

# "doveconf -n" command gives a clean output of the changed settings. Use it
# instead of copy&pasting files when posting to the Dovecot mailing list.

# '#' character and everything after it is treated as comments. Extra spaces
# and tabs are ignored. If you want to use either of these explicitly, put the
# value inside quotes, eg.: key = "# char and trailing whitespace  "

# Default values are shown for each setting, it's not required to uncomment
# those. These are exceptions to this though: No sections (e.g. namespace {})
# or plugin settings are added by default, they're listed only as examples.
# Paths are also just examples with the real defaults being based on configure
# options. The paths listed here are for configure --prefix=/usr
# --sysconfdir=/etc --localstatedir=/var

# Enable installed protocols
!include_try /usr/share/dovecot/protocols.d/*.protocol
protocols = imap pop3 lmtp
```

Save and close, and then open the next config file…

```
$ sudo vim /etc/dovecot/conf.d/10-mail.conf
```

Search for and modify the following lines (they’re not together in the file):

```
mail_location = maildir:/var/mail/vhosts/%d/%n
mail_privileged_group = mail
```

Save and close. Next, we need to verify some permissions, so run…

```
$ ls -ld /var/mail
```

Ensure the output is at follows (neverminding the date/time):

```
drwxrwsr-x 2 root mail 4096 Feb 21 10:07 /var/mail
```

Then, we’ll add subdirectories for each domain we’ll be receiving email on.

```
$ sudo mkdir -p /var/mail/vhosts/qub3d.org
```

Now we need to set up a `vmail` user.

```
$ sudo groupadd -g 5000 vmail
$ sudo useradd -g vmail -u 5000 vmail -d /var/mail
```

We’ll transfer ownership of the mail directory and all its contents to the `vmail` user.

```
$ sudo chown -R vmail:vmail /var/mail
```

Now we edit another configuration.

```
$ sudo vim /etc/dovecot/conf.d/10-auth.conf
```

Change or add the following lines. Notice that the last two are just being commented or uncommented:

```
auth_mechanisms = plain login
disable_plaintext_auth = yes

#!include auth-system.conf.ext
!include auth-sql.conf.ext
```

Save and close. Then, run…

```
$ sudo vim /etc/dovecot/conf.d/auth-sql.conf.ext
```

Ensure the following lines match and are uncommented:

```
passdb {
  driver = sql
  args = /etc/dovecot/dovecot-sql.conf.ext
}
userdb {
  driver = static
  args = uid=vmail gid=vmail home=/var/mail/vhosts/%u
   }
```

Save and close. Then, run…

```
$ sudo vim /etc/dovecot/dovecot-sql.conf.ext
```

Find, uncomment, and edit the following lines so they match, replacing `userpassword` with the actual password for the `postmaster` MySQL account:

```
driver = mysql
connect = host=127.0.0.1 dbname=mailserver user=postmaster password=userpassword
default_pass_scheme = SHA512-CRYPT
password_query = \
    SELECT username as username, password \
    FROM mailbox WHERE username = '%u'
```

Save and close. Then, we'll adjust a few more permissions and edit the sockets configuration file.

```
$ sudo chown -R vmail:dovecot /etc/dovecot
$ sudo chmod -R o-rwx /etc/dovecot
$ sudo vim /etc/dovecot/conf.d/10-master.conf
```

We're going to disable IMAP and POP3 (the unencrypted forms) and instead use the secure versions (IMAPS and POP3S). Edit the following lines of code. Be careful of the nested code and brackets:

```
service imap-login {
  inet_listener imap {
    port = 0
  }
  inet_listener imaps {
    port = 993
    ssl = yes
  }

service pop3-login {
  inet_listener pop3 {
    port = 0
  }
```

(continues on next page)

(continued from previous page)

```
  inet_listener pop3s {
    port = 995
    ssl = yes
  }
}

service lmtp {
  unix_listener /var/spool/postfix/private/dovecot-lmtp {
    mode = 0666
    user = postfix
    group = postfix
  }

service auth {
  unix_listener auth-userdb {
    mode = 0666
    user = vmail
    #group =
  }

  # Postfix smtp-auth
  unix_listener /var/spool/postfix/private/auth {
    mode = 0666
    user = postfix
    group = postfix
  }

  # Auth process is run as this user.
  #user = $default_internal_user
  user = dovecot
}

service auth-worker {
  # Auth worker process is run as root by default, so that it can access
  # /etc/shadow. If this isn't necessary, the user should be changed to
  # $default_internal_user.
  user = vmail
}
```

Save and close. Now we'll configure Dovecot to use our Let's Encrypt certificate.

```
$ sudo vim /etc/dovecot/conf.d/10-ssl.conf
```

Uncomment and change the following lines:

```
# SSL/TLS support: yes, no, required. <doc/wiki/SSL.txt>
ssl = required

# PEM encoded X.509 SSL/TLS certificate and private key. They're opened before
# dropping root privileges, so keep the key file unreadable by anyone but
# root. Included doc/mkcert.sh can be used to easily generate self-signed
# certificate, just make sure to update the domains in dovecot-openssl.cnf
ssl_cert = </etc/apache2/certs/cert.pem
ssl_key = </etc/apache2/certs/privkey.pem
ssl_ca = </etc/apache2/certs/chain.pem
```

Save and close, and then restart both Postfix and Dovecot.

```
$ sudo systemctl restart postfix dovecot
```

Firewall Settings

Now we need to open the firewall to allow email to pass through.

```
$ sudo ufw allow 25
$ sudo ufw allow 465
$ sudo ufw allow 587
$ sudo ufw allow 993
$ sudo ufw allow 995
```

Note: By this point, the email system should be 100% functional, sending and receiving email and serving it to clients over POPS3, IMAPS, and SMTPS. Test this out before continuing! Note the debugging instructions below.

Debugging

Errors can usually be found by running `sudo systemctl status postfix` and `sudo systemctl status postfix`, with the full logs visible at `/var/log/mail.log`.

You can check postfix's delivery queue with `postqueue -p`, and attempt to clear it (deliver everything) with `postqueue -f`.

MXToolbox SuperTool can be used to check for DNS and MX issues. Enter the domain and select `Test Email Server` from the options.

SOURCE: Email with Postfix, Dovecot, and MySQL (Linode)

Mail Server Security (DKIM, SPF, and Postfix)

Setup

First, we'll install the packages we need.

```
$ sudo apt install opendkim opendkim-tools postfix-policyd-spf-python
```

We also need the `postfix` user to be a member of the group for opendkim.

```
$ sudo adduser postfix opendkim
```

SPF

The purpose of SPF is to prevent spoofing and other email fraud. There are a couple of approaches for this, but we want to link all hosts listed in our MX records to our server.

Go to the Linode DNS manager for the domain in question, and add a new `TXT Records`. Leave the `Name` blank, and set the `Value` to the following:

```
v=spf1 mx -all
```

Now we need to edit our mail server to work with SPF.

```
$ sudo vim /etc/postfix-policyd-spf-python/policyd-spf.conf
```

Change the following lines:

```
HELO_reject = False
Mail_From_reject = False
```

Save and close, then run. . .

```
$ sudo vim /etc/postfix/master.cf
```

Add the following to the bottom of that file:

```
policyd-spf  unix  -       n       n       -       0       spawn
    user=policyd-spf argv=/usr/bin/policyd-spf
```

Save and close, then run. . .

```
$ sudo vim /etc/postfix/main.cf
```

Edit the following section to include the last line in the example below, also ensuring that a comma is at the end of every line except the last one:

```
smtpd_recipient_restrictions =
    permit_sasl_authenticated,
    permit_mynetworks,
    reject_unauth_destination,
    check_policy_service unix:private/policyd-spf
```

Also add the following line to the bottom:

```
policyd-spf_time_limit = 3600
```

Save and close, and then restart Postfix.

> **Warning:** Be *very* careful with your configurations! Continue to test your email send/receive periodically, to make sure nothing breaks. Some email can actually get lost or be eaten if your settings are wrong, and you don't want important messages going into a black hole.

To test, send a message TO an account on your mail server, and check the headers. You should see a line something like this:

```
Received-SPF: Pass (sender SPF authorized) identity=mailfrom; client-
↪ip=2607:f8b0:400e:c05::22a; helo=mail-pg0-x22a.google.com; envelope-
↪from=someemail@example.com; receiver=webster@qub3d.org
```

You should also see something like the following in `/var/log/mail.log`:

```
ubuntu policyd-spf[18663]: None; identity=helo; client-ip=2607:f8b0:400e:c00::22e;␣
↪helo=mail-pf0-x22e.google.com; envelope-from=someemail@example.com;␣
↪receiver=test@mousepawgames.net
ubuntu policyd-spf[18663]: Pass; identity=mailfrom; client-ip=2607:f8b0:400e:c00::22e;
↪ helo=mail-pf0-x22e.google.com; envelope-from=someemail@example.com;␣
↪receiver=test@mousepawgames.net
```

OpenDKIM

DKIM allows us to verify that messages sent from our server really *did* come from us, which further helps ensure the safety and validity of our messages, and avoid winding up in spam.

We'll start by modifying the configuration for OpenDKIM…

```
$ sudo vim /etc/opendkim.conf
```

Edit to make it match the following:

```
# Log to syslog
Syslog                  yes
# Required to use local socket with MTAs that access the socket as a non-
# privileged user (e.g. Postfix)
UMask                   002
# OpenDKIM user
# Remember to add user postfix to group opendkim
UserID                  opendkim

# Map domains in From addresses to keys used to sign messages
KeyTable                refile:/etc/opendkim/key.table
SigningTable            refile:/etc/opendkim/signing.table

# Hosts to ignore when verifying signatures
ExternalIgnoreList      /etc/opendkim/trusted.hosts
InternalHosts           /etc/opendkim/trusted.hosts

# Sign for example.com with key in /etc/dkimkeys/dkim.key using
# selector '2007' (e.g. 2007._domainkey.example.com)
#Domain                 example.com
#KeyFile                /etc/dkimkeys/dkim.key
#Selector               2007

# Commonly-used options; the commented-out versions show the defaults.
#Canonicalization       simple
Canonicalization        relaxed/simple
Mode                    sv
SubDomains              no
#ADSPAction             continue
AutoRestart             yes
AutoRestartRate         10/1M
Background              yes
DNSTimeout              5
SignatureAlgorithm      rsa-sha256

# Always oversign From (sign using actual From and a null From to prevent
# malicious signatures header fields (From and/or others) between the signer
# and the verifier.  From is oversigned by default in the Debian pacakge
# because it is often the identity key used by reputation systems and thus
# somewhat security sensitive.
OversignHeaders         From

LogWhy                  Yes
TemporaryDirectory      /var/tmp
```

Save and close, and then update the permissions for that file. We'll also be setting up directories for OpenDKIM.

```
$ sudo chmod u=rw,go=r /etc/opendkim.conf
$ sudo mkdir /etc/opendkim
$ sudo mkdir /etc/opendkim/keys
$ sudo chown -R opendkim:opendkim /etc/opendkim
$ sudo chmod go-rw /etc/opendkim/keys
$ sudo vim /etc/opendkim/signing.table
```

That last command will open up a file for editing, where we'll define the domains we're signing for. Set the contents to something like the following, replacing with the domains you're setting up for:

```
*@qub3d.org qub3dorg._domainkey.qub3d.org
```

Save and close, and then open the key table file.

```
$ sudo vim /etc/opendkim/key.table
```

Set the contents to something like the following, with the short keys from the earlier file on the left. Also, be sure to change the date to the current four-digit year and two-digit month:

```
qub3dorg._domainkey.qub3d.org qub3d.org:201705:/etc/opendkim/keys/qub3dorg.private
```

Save and close, and then open the trusted hosts configuration file.

```
$ sudo vim /etc/opendkim/trusted.hosts
```

Set the contents of that file to:

```
127.0.0.1
::1
localhost
qub3d
qub3d.qub3d.org
qub3d.org
```

Save and close, and then double-check the permissions on OpenDKIM's directories.

```
$ sudo chown -R opendkim:opendkim /etc/opendkim
$ sudo chmod -R go-rwx /etc/opendkim/keys
```

Now we can generate our keys and set their permissions. Since we need to regenerate this regularly, and there is a LOT involved, I wrote up a handy script.

```
$ sudo vim /opt/scripts/root_scripts/dkim_manage
```

Get the code from the dkim_manage Github.

Save and close, and set the proper permissions.

```
$ sudo chmod +x /opt/scripts/root_scripts/dkim_manage
```

Before we can run the script, we also need to set up the configuration file it uses. Note that I'm creating this at the path specified at the top of script above (near the `# CHANGE THIS` comment.)

```
$ sudo vim /opt/scripts/root_scripts/domains.txt
```

Set the contents of that file to:

```
qub3d.org
```

Save and close. Now we can use the script to generate the keys.

```
$ sudo /opt/scripts/root_scripts/dkim_manage -g
```

Now we need to configure our DNS to use these keys. It can be a little tricky to get the entries correct, so I set up the script from earlier to also display what we need.

```
$ sudo /opt/scripts/root_scripts/dkim_manage -d
```

Copy and paste each of the keys (starting with `v=DKIM1` to the end of the string) into a new TXT record for each DNS. The record name should be `YYYYMM._domainkey` (i.e. `201705._domainkey`).

Once you have these in place, wait a bit for them to propegate, and then test the keys.

```
$ sudo /opt/scripts/root_scripts/dkim_manage -t
```

If ALL your keys have passed, we are ready to move them into place.

```
$ sudo /opt/scripts/root_scripts/dkim_manage -m
```

That last command will also restart OpenDKIM and Postfix automatically.

OpenDKIM and Postfix

Once OpenDKIM is set up, we need to configure Postfix to use it.

```
$ sudo mkdir -p /var/run/opendkim
$ sudo chown opendkim:postfix /var/run/opendkim
$ sudo vim /etc/default/opendkim
```

Make sure the uncommented line matches:

```
SOCKET="inet:8891@localhost"
```

Save and close the file, and then edit the configuration for Postfix.

```
sudo vim /etc/postfix/main.cf
```

Add the following just below the `smtpd_recipient_restrictions` section:

```
# Milter configuration
# OpenDKIM
milter_default_action = accept
# Postfix  2.6 milter_protocol = 6, Postfix  2.5 milter_protocol = 2
milter_protocol = 6
smtpd_milters = inet:localhost:8891
non_smtpd_milters = inet:localhost:8891

# FOLLOWING https://www.linode.com/docs/email/postfix/configure-spf-and-dkim-in-
↪postfix-on-debian-8
```

Save and close, and then restart OpenDKIM and Postfix. These need to be restarted separately, so OpenDKIM sets up the proper socket for Postfix.

```
$ sudo systemctl restart opendkim
$ sudo systemctl restart postfix
```

To test it out, send an email to `check-auth@verifier.port25.com`. The report should return `DKIM check: pass`.

Note: Special thanks for SCHAPiE for fixing this for us!

DMARC and ADSP

After all that, this one is nice and simple. Just add a new TXT record to each domain's DNS. Set the Name to `_dmarc`, and Value to `v=DMARC1;p=quarantine;sp=quarantine;adkim=r;aspf=r`.

ADSP is more-or-less a moot point - it's actually deprecated. Still, setting it is super easy, and it doesn't hurt anything, so it's not bad to offer ultra-legacy support. Once again, add a new TXT record to each domain's DNS. The Name is `_adsp._domainkey`, and the Value is `dkim=all`.

SOURCE: Configure SPF and DKIM in Postfix on Debian 8

Mail Filtering

Setup

We'll start by installing the packages we need.

```
$ sudo apt install amavisd-new spamassassin clamav-daemon postfix-policyd-spf-python␣
↪pyzor razor arj cabextract cpio nomarch pax rar unrar unzip zip
```

Amavis (`amavisd-new`) is the program responsible for the actual filtering.

Now we need to edit SpamAssassin's configuration.

```
$ sudo vim  /etc/default/spamassassin
```

Change the following lines:

```
ENABLED=1
CRON=1
```

Save and close, and then we'll update SpamAssassin's definitions and start the daemon.

```
$ sudo sa-update
$ sudo systemctl start spamassassin
```

Save and close.

Next, we'll make a copy of the default configuration file and edit that.

```
$ sudo cp /etc/spamassassin/local.cf /etc/spamassassin/local.cf.orig
$ sudo vim /etc/spamassassin/local.cf
```

Find and uncomment the following lines:

```
required_score 5.0
use_bayes 1
bayes_auto_learn 1
bayes_ignore_header X-Bogosity
bayes_ignore_header X-Spam-Flag
bayes_ignore_header X-Spam-Status
```

Note: This is enabling the Bayes system for SpamAssassin. You *will* need to regularly train this system.

Save and close.

Now we connect Amavis to ClamAV and SpamAssassin.

```
$ sudo vim /etc/amavis/conf.d/15-content_filter_mode
```

Change the file so it matches the following, simply by uncommenting the lines for the anti-virus and spam checking modes:

```
use strict;

# You can modify this file to re-enable SPAM checking through spamassassin
# and to re-enable antivirus checking.

#
# Default antivirus checking mode
# Please note, that anti-virus checking is DISABLED by
# default.
# If You wish to enable it, please uncomment the following lines:


# @bypass_virus_checks_maps = (
#    \%bypass_virus_checks, \@bypass_virus_checks_acl, \$bypass_virus_checks_re);


#
# Default SPAM checking mode
# Please note, that anti-spam checking is DISABLED by
# default.
# If You wish to enable it, please uncomment the following lines:


@bypass_spam_checks_maps = (
   \%bypass_spam_checks, \@bypass_spam_checks_acl, \$bypass_spam_checks_re);

1;  # ensure a defined return
```

Save and close, and then run…

```
$ sudo vim /etc/amavis/conf.d/20-debian_defaults
```

We want spam messages to be discarded instead of bounced, so we need to edit that setting here:

```
$final_spam_destiny       = D_DISCARD;
```

Also edit the following lines so all mail is given info headers, and to control when the spam filters kick it to varying degrees.

> $sa_spam_subject_tag = ‘’; # shut off header rewriting, we’ll Junk-bin it instead. $sa_tag_level_deflt = -999; # add spam info headers if at, or above that level $sa_tag2_level_deflt = 5.0; # add ‘spam detected’ headers at that level $sa_kill_level_deflt = 12; # triggers spam evasive actions $sa_dsn_cutoff_level = 10; # spam level beyond which a DSN is not sent

Save and close.

SOURCE: Amavis Spam FAQ

We also need to modify the hostname and domains Amavis works with.

```
$ sudo vim /etc/amavis/conf.d/50-user
```

Change or add the following lines:

```
$myhostname = 'qub3d.qub3d.org';
@local_domains_acl = ( "qub3d.org" );
```

Save and close, and then restart Amavis.

```
$ sudo systemctl restart amavis
```

Postfix Integration

Run the following. . .

```
$ sudo postconf -e 'content_filter = smtp-amavis:[127.0.0.1]:10024'
$ sudo vim /etc/postfix/master.cf
```

Add the following to the end of the file:

```
smtp-amavis     unix    -       -       -       -       2       smtp
        -o smtp_data_done_timeout=1200
        -o smtp_send_xforward_command=yes
        -o disable_dns_lookups=yes
        -o max_use=20

127.0.0.1:10025 inet    n       -       -       -       -       smtpd
        -o content_filter=
        -o local_recipient_maps=
        -o relay_recipient_maps=
        -o smtpd_restriction_classes=
        -o smtpd_delay_reject=no
        -o smtpd_client_restrictions=permit_mynetworks,reject
        -o smtpd_helo_restrictions=
        -o smtpd_sender_restrictions=
        -o smtpd_recipient_restrictions=permit_mynetworks,reject
        -o smtpd_data_restrictions=reject_unauth_pipelining
        -o smtpd_end_of_data_restrictions=
        -o mynetworks=127.0.0.0/8
        -o smtpd_error_sleep_time=0
        -o smtpd_soft_error_limit=1001
        -o smtpd_hard_error_limit=1000
        -o smtpd_client_connection_count_limit=0
        -o smtpd_client_connection_rate_limit=0
        -o receive_override_options=no_header_body_checks,no_unknown_recipient_checks,
↪no_milters
```

Also add the following just below the `pickup` line:

```
-o content_filter=
-o receive_override_options=no_header_body_checks
```

Save and close, and then restart Postfix.

```
$ sudo systemctl restart postfix
```

Ensure Amavis is running correctly with…

```
$ telnet localhost 10024
```

The output should be:

```
Trying ::1...
Connected to localhost.
Escape character is '^]'.
220 [::1] ESMTP amavisd-new service ready
```

Press `Ctrl-]` and `Enter` to exit the telnet session, and then type 'quit' and press `Enter`.

To test everything out, send a message to your email server, and check it for the spam and virus scan headers.

Note: mailbox folders must be recursively set as owner `vmail:vmail`, with recursively-applied permissions `700` and `g+s`.

SOURCE: Mail Filtering (Ubuntu)

Dovecot Sieve

To automatically place spam messages into Junk, we need to configure a Dovecot sieve.

```
$ sudo apt install dovecot-sieve
$ sudo vim /etc/dovecot/conf.d/90-sieve.conf
```

Edit that file to comment out the line:

```
#sieve = file:~/sieve;active=~/.dovecot.sieve
```

Save and close, and then run…

```
$ sudo vim /etc/dovecot/conf.d/90-plugin.conf
```

Add the following to that file:

```
plugin {
    sieve = /etc/dovecot/sieve/default.sieve
}
```

Save and close, and run…

```
$ sudo vim /etc/dovecot/conf.d/15-lda.conf
```

Edit the following section so it incorporates the following:

```
protocol lda {
  mail_plugins = $mail_plugins sieve
}
```

Save and close, and then…

```
$ sudo vim /etc/dovecot/conf.d/20-lmtp.conf
```

As before, edit the following section so it incorporates the following:

```
protocol lmtp {
  mail_plugins = $mail_plugins sieve
}
```

Now we can set up the sieve itself.

```
$ sudo mkdir /etc/dovecot/sieve/
$ sudo vim /etc/dovecot/sieve/default.sieve
```

Set the contents of that new file to:

```
require "fileinto";
if header :contains "X-Spam-Flag" "YES" {
    fileinto "Junk";
}
```

Save and close, and run…

```
$ sudo chown vmail:vmail /etc/dovecot/sieve/ -R
$ sudo systemctl restart postfix
$ sudo systemctl restart dovecot
$ sudo systemctl restart spamassassin
$ sudo systemctl restart amavis
```

As always, test to make sure normal mail still gets through.

SOURCE: How to move spam to spam folder? (StackOverflow)

Training SpamAssassin

We want SpamAssassin to automatically train itself based on the Inbox, Junk, and Archive folders.

Note: This requires all users to monitor their own mailboxes, to ensure that Junk is filled only with spam, and none slips past in the Inbox.

Edit the root crontab…

```
$ sudo crontab -e
```

Add the following lines.

```
00 8 * * * /usr/bin/sa-learn --spam /var/mail/vhosts/qub3d.org/*/.Junk/cur
15 8 * * * /usr/bin/sa-learn --ham /var/mail/vhosts/qub3d.org/*/cur
30 8 * * * /usr/bin/sa-learn --ham /var/mail/vhosts/qub3d.org/*/.Archive*/cur
```

Blacklists

Sometimes a spam filter just isn't enough. When we're regularly getting spam from a particular email address or domain name, we can blacklist it entirely.

```
$ sudo vim /etc/postfix/main.cf
```

Change the following section to match what is shown here:

```
smtpd_recipient_restrictions =
    check_sender_access hash:/etc/postfix/sender_checks,
    permit_sasl_authenticated,
    permit_mynetworks,
    reject_unauth_destination,
    check_policy_service unix:private/policyd-spf
```

Save and close. Now we create the blacklist file.

```
$ sudo vim /etc/postfix/sender_checks
```

We format the blacklist like this:

```
# Restricts sender addresses this system accepts in MAIL FROM commands.

guerrillamail.com REJECT Just a test
.guerrillamail.com REJECT

periscopedata.com REJECT
.periscopedata.com REJECT

nimblechapps.co.uk REJECT
.nimblechapps.co.uk REJECT
nimblechaps@gmail.com REJECT
```

Note that we are blocking email from the domain on the first of each pair, and blocking email from all subdomains on that domain on the second of each pair.

Sometimes we also need to block a specific email address, such as on the last line of our example. (We obviously don't want to block all of Gmail!)

After each domain or email address, we include the argument `REJECT`, optionally followed by a comment or description thereof.

Note: The first pair is only on this list for *testing* purposes. Be sure to unblock guerrillamail.com when we're done!

Save and close the file, and run…

```
$ sudo postmap /etc/postfix/sender_checks
$ sudo systemctl reload postfix
```

This will load the blacklist in. Now, to test it, we can go to `guerrillamail.com` and use their free service to send an email to our mail server. If it arrives, we made a mistake somewhere.

Otherwise, if the email doesn't arrive, our blacklist works. We can unblock that test domain…

```
$ sudo vim /etc/postfix/sender_checks
```

Change that file to:

```
# Restricts sender addresses this system accepts in MAIL FROM commands.

#guerrillamail.com REJECT Just a test
#.guerrillamail.com REJECT

periscopedata.com REJECT
.periscopedata.com REJECT

nimblechapps.co.uk REJECT
.nimblechapps.co.uk REJECT
nimblechaps@gmail.com REJECT
```

Note that we *commented out* the guerrillamail.com lines, in case we need them for testing again. However, you can absolutely remove those lines instead.

Save and close, and then run…

```
$ sudo postmap /etc/postfix/sender_checks
$ sudo systemctl reload postfix
```

Send one more test email from `guerrillamail.com` to make sure non-blacklisted email addresses are not blocked.

If that works, all is humming along as it should!

SOURCE: Blacklist and Whitelist with Postfix (linuxlasse.net)

Mail Clients

We'll be installing the web client Rainloop.

```
$ mkdir /opt/rainloop
$ cd /opt/rainloop
$ sudo wget https://www.rainloop.net/repository/webmail/rainloop-community-latest.zip
$ sudo unzip rainloop-community-latest.zip
$ sudo chown -R www-data:www-data /opt/rainloop
$ sudo find . -type d -exec chmod 755 {} \;
$ sudo find . -type f -exec chmod 644 {} \;
$ sudo vim /etc/apache2/sites-available/webmail.conf
```

Set the contents of that file to…

```
<IfModule mod_ssl.c>
    <VirtualHost *:443>
        ServerName webmail.qub3d.org

        ServerAdmin webmaster@qub3d.org
        DocumentRoot /opt/rainloop

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        <Directory /opt/rainloop>
            Options FollowSymLinks
            AllowOverride All
            Require all granted
        </Directory>
```

(continues on next page)

(continued from previous page)

```
        # prevent iframing this site
        Header always append X-Frame-Options SAMEORIGIN

        # SSL
        SSLEngine on
        SSLCertificateFile      /etc/apache2/certs/fullchain.pem
        SSLCertificateKeyFile   /etc/apache2/certs/privkey.pem

        Include /etc/letsencrypt/options-ssl-apache.conf

        <FilesMatch "\.(cgi|shtml|phtml|php)$">
                SSLOptions +StdEnvVars
        </FilesMatch>
        <Directory /usr/lib/cgi-bin>
                SSLOptions +StdEnvVars
        </Directory>
    </VirtualHost>
</IfModule>
<VirtualHost *:80>
    ServerName webmail.qub3d.org

    RewriteEngine On
    RewriteCond %{HTTPS} off
    RewriteRule ^ https://%{HTTP_HOST}%{REQUEST_URI}
</VirtualHost>
```

Save and close. Now edit Apache2's main configuration.

```
$ sudo vim /etc/apache2/apache2.conf
```

Add the following to the *<Directory>* directives section.

```
<Directory /opt/rainloop>
    Options FollowSymLinks
    AllowOverride All
    Require all granted
</Directory>
```

Enable the site and restart Apache2…

```
$ sudo a2ensite webmail
$ sudo systemctl restart apache2
```

Now go to `https://webmail.qub3d.org/?admin`.

Grav

Grav has relatively few system requirements, but it does need some PHP and Apache modules. Let's install and enable those…

```
$ sudo apt install php7.2-curl php7.2-ctype php7.2-dom php7.2-gd php7.2-json php7.2-
↪mbstring php7.2-xml php7.2-yaml php7.2-zip
$ sudo apt install composer
$ sudo a2enmod rewrite
$ sudo a2enmod ssl
```

Now we create the directory for Grav to live in.

```
$ sudo mkdir /opt/grav
$ sudo chown itdude:www-data grav
```

And now we install!

```
$ cd /opt/grav
$ composer create-project getgrav/grav /opt/grav
$ sudo chown -R itdude:www-data /opt/grav
$ cd /opt/grav
$ find . -type f | xargs chmod 664
$ find ./bin -type f | xargs chmod 775
$ find . -type d | xargs chmod 775
$ find . -type d | xargs chmod +s
```

Next, we'll create a VirtualHost for Grav.

```
$ sudo vim /etc/apache2/sites-available/grav.conf
```

Set the contents of that file to. . .

```
<IfModule mod_ssl.c>
    <VirtualHost *:443>
        ServerName qub3d.org

        ServerAdmin webmaster@qub3d.org
        DocumentRoot /opt/qub3d

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        <Directory /opt/qub3d>
                Options -MultiViews -Indexes
                AllowOverride All
        </Directory>

        # SSL
        SSLEngine on
        SSLCertificateFile      /etc/apache2/certs/fullchain.pem
        SSLCertificateKeyFile   /etc/apache2/certs/privkey.pem

        Include /etc/letsencrypt/options-ssl-apache.conf

        <FilesMatch "\.(cgi|shtml|phtml|php)$">
                SSLOptions +StdEnvVars
        </FilesMatch>
        <Directory /usr/lib/cgi-bin>
                SSLOptions +StdEnvVars
        </Directory>

    </VirtualHost>
</IfModule>
<VirtualHost *:80>
    ServerName qub3d.org

    RewriteEngine On
    RewriteCond %{HTTPS} off
```

(continues on next page)

(continued from previous page)

```
    RewriteRule ^ https://%{HTTP_HOST}%{REQUEST_URI}
</VirtualHost>
```

Save and close. Now, open up the Apache2 configuration…

```
$ sudo vim /etc/apache2/apache2.conf
```

…and add the following to the section with all the <*Directory*> statments.

```
<Directory /opt/grav>
    Options FollowSymLinks
    AllowOverride All
    Require all granted
</Directory>
```

Finally, we restart Apache2, and go to `https://qub3d.org` to see our newly installed Grav!

Now let's add the admin panel.

```
$ /opt/grav/bin/gpm version -f
$ /opt/grav/bin/gpm selfupgrade
$ /opt/grav/bin/gpm install admin
```

Go to `https://qub3d.org` again, and set up a new admin account.

After accessing the admin panel, you can create additional user profiles from the command line with a simple command…

```
$ /opt/grav/bin/plugin login newuser
```

Follow the instructions, and you're done!

You can now use the Admin Panel to edit the Grav website. Login and get started at `https://qub3d.org/admin`.

SOURCE: Grav Docs

Phabricator

Setting Up System Groups and Users

We'll add a group to control who can access Phabricator's stuff. For ease of use, we'll add our login users to this group. We will also create a new user called `phabdaemon` for Phabricator-based daemons.

```
$ sudo groupadd phab
$ sudo useradd -G phab phabdaemon
$ sudo usermod -a -G phab jason
$ sudo usermod -a -G phab www-data
```

Now we need to modify the `phabdaemon` user.

```
$ sudo vim /etc/passwd
```

Look for the `phabdaemon` entry and set the last field to `/usr/sbin/nologin`. Save and close. Then…

```
$ sudo vim /etc/shadow
```

Look for the `phabdaemon` entry again, and set the second field to `*`. Save and close.

Installing Phabricator

Let's start by pulling down Phabricator…

```
$ sudo mkdir /opt/phab
$ sudo chown itdude:www-data
$ cd /opt/phab
$ git clone https://github.com/phacility/libphutil.git
$ git clone https://github.com/phacility/arcanist.git
$ git clone https://github.com/phacility/phabricator.git
$ ./phabricator/scripts/install/install_ubuntu.sh
```

Once that's done, we need to configure Apache2 to know about Phabricator.

```
$ sudo vim /etc/apache2/sites-available/phab.conf
```

Set the contents of that file to…

```
<IfModule mod_ssl.c>
    <VirtualHost *:443>
        ServerName phab.qub3d.org

        ServerAdmin webmaster@qub3d.org
        DocumentRoot /opt/phab/phabricator/webroot

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        RewriteEngine on
        RewriteRule ^(.*)$          /index.php?__path__=$1  [B,L,QSA]

        <Directory /opt/phab>
                Options -MultiViews -Indexes
                AllowOverride All
        </Directory>

        # SSL
        SSLEngine on
        SSLCertificateFile      /etc/apache2/certs/fullchain.pem
        SSLCertificateKeyFile   /etc/apache2/certs/privkey.pem

        Include /etc/letsencrypt/options-ssl-apache.conf

        <FilesMatch "\.(cgi|shtml|phtml|php)$">
                SSLOptions +StdEnvVars
        </FilesMatch>
        <Directory /usr/lib/cgi-bin>
                SSLOptions +StdEnvVars
        </Directory>

    </VirtualHost>
</IfModule>
<VirtualHost *:80>
    ServerName phab.qub3d.org

    RewriteEngine On
    RewriteCond %{HTTPS} off
```

(continues on next page)

(continued from previous page)

```
    RewriteRule ^ https://%{HTTP_HOST}%{REQUEST_URI}
</VirtualHost>
```

Save and close. Now, open up the Apache2 configuration…

```
$ sudo vim /etc/apache2/apache2.conf
```

…and add the following to the section with all the <*Directory*> statments.

```
<Directory /opt/phab/phabricator/webroot>
    Options FollowSymLinks
    AllowOverride All
    Require all granted
</Directory>
```

Save and close. Now enable the new site and restart Apache2.

```
$ sudo a2ensite phab
$ sudo systemctl restart apache2
```

Right off the bat, you'll probably get a complaint about our `disable_functions` setting in `php.conf`. They just don't want us turning off stuff Phabricator needs, but our disabled commands do *not* conflict. Therefore, we'll just temporarily disable…

```
$ sudo vim /etc/php/7.2/apache2/php.ini
```

Find the line starting with `disable_functions =`, and put a semicolon (`;`) in front of it. Save and exit, and then restart Apache2…

```
$ sudo systemctl restart apache2
```

And then refresh Phabricator's web page. Now it will probably complain about MySQL.

Go to `https://qub3d.org/phpmyadmin` and login using the MySQL root credentials. Go to *User Accounts*, and select *Add user account*…

- User name: `phab`
- Host name: `localhost`
- Password: (Set a password)

Scroll down and grant the user account *Global privileges*. Then, scroll to the bottom and click *Go*.

Then, run the following commands…

```
$ sudo /opt/phab/phabricator/bin/config set mysql.user phab
$ sudo /opt/phab/phabricator/bin/config set mysql.pass thepassword
```

…replacing `thepassword` with the password you specified a moment ago. Then run…

```
$ /opt/phab/phabricator/bin/storage upgrade
```

Give it a few minutes to set up its tables.

If all went well, you will arrive on the home page of your new Phabricator installation! Create the base administrative account, and then dive in.

Set Logging Locations

```
$ sudo mkdir -p /opt/log/phab
$ sudo chown -R itdude:phab /opt/log/phab
$ sudo chmod -R u=rwx,g=rwx,o=rx /opt/log/phab
$ sudo /opt/phab/phabricator/bin/config set log.access.path /opt/log/phab/access.log
$ sudo /opt/phab/phabricator/bin/config set log.ssh.path /opt/log/phab/ssh.log
$ sudo /opt/phab/phabricator/bin/config set phd.log-directory /opt/log/phab/phd.log
```

Fixing Installation Issues

Now we're going to resolve most of the installation issues being reported. Run the following commands…

```
$ sudo /opt/phab/phabricator/bin/config set phabricator.base-uri 'https://phab.qub3d.
↪org/'
$ sudo apt install php7.2-apcu python-pygments python3-pygments imagemagick subversion
$ sudo /opt/phab/phabricator/bin/config set pygments.enabled true
$ sudo ln -s /usr/lib/git-core/git-http-backend /opt/phab/phabricator/support/bin/git-
↪http-backend
```

Now we need to edit several parameters in `php.ini`.

```
$ sudo vim /etc/php/7.2/apache2/php.ini
```

Change the following value:

```
date.timezone = America/Chicago
opcache.validate_timestamps = 0
post_max_size = 32M
```

We also need to edit our MySQL configuration…

```
$ sudo vim /etc/mysql/mysql.conf.d/mysqld.cnf
```

Add or edit the following lines under the `[mysqld]` section:

```
max_allowed_packet = 32M
upload_max_filesize = 32M
sql_mode=STRICT_ALL_TABLES
innodb_buffer_pool_size=1600M
```

Restart MySQL…

```
$ sudo systemctl restart mysql
```

And then refresh Phabricator. Over half of our installation issues are now resolved!

Large File Storage

Instead of storing files in blobs on the MySQL database, we'll store them in a folder.

```
$ sudo mkdir /opt/phabfiles
$ sudo chown itdude:phab /opt/phabfiles
```

(continues on next page)

(continued from previous page)

```
$ sudo chmod 775 /opt/phabfiles
$ sudo /opt/phab/phabricator/bin/config set storage.local-disk.path "/opt/phabfiles"
```

Repository Path

Next, we need to set up the folder for hosting our repositories.

```
$ sudo mkdir /opt/phabrepos
$ sudo chown itdude:phab /opt/phabrepos
$ sudo chmod 775 /opt/phabrepos
$ sudo /opt/phab/phabricator/bin/config set repository.default-local-path "/opt/
↪phabrepos"
```

Setting Up Alternative File Domain

You'll notice that earlier, we included `files.qub3d.org` in our certificate. That's so we can set up an alternative file domain.

We need to make a new VirtualHost for our second domain. Since this will be practically identical to the one for Phabricator, we can just run…

```
$ sudo cp /etc/apache2/sites-available/phab.conf /etc/apache2/sites-available/
↪phabfile.conf
$ sudo vim /etc/apache2/sites-available/phabfile.conf
```

Edit *both instances* of the following line to match the following…

```
ServerName files.qub3d.org
```

Save and close, and then enable the new site and restart Apache2.

```
$ sudo a2ensite phabfile
$ sudo systemctl restart apache2
```

Finally, we tell Phabricator to use the new domain.

```
$ sudo /opt/phab/phabricator/bin/config set security.alternate-file-domain 'https://
↪files.qub3d.org/'
```

Set Up Phabricator Daemons

We need to autostart the Phabricator daemons. I wrote a special script that handles that.

```
$ sudo mkdir /opt/scripts/phab
$ sudo chown jason:phab /opt/scripts/phab
$ sudo chmod u=rwx,g=rwx,o=rx /opt/scripts/phab
$ sudo vim /opt/scripts/phab/phd_start
```

Put the following in that file.

```
#!/bin/bash
#Start Phabricator daemons

echo "STARTING PHD" > /opt/log/phab/phd_start.log
sudo -u phabdaemon /opt/phab/phabricator/bin/phd start > /opt/log/phab/phd_start.log
sudo -u phabdaemon /opt/phab/phabricator/bin/phd launch phabricatorbot /opt/phab/
↪phabricator/resources/chatbot/botconfig.json > /opt/log/phab/phd_start.log
```

Save and close. Then, change its permissions.

```
$ sudo chmod u=rwx,g=rwx,o=rx /opt/scripts/phab/phd_start
```

Now, add this script to the crontab.

```
$ sudo crontab -e
```

At the bottom, add the line:

```
@reboot sleep 60; /opt/scripts/phab/phd_start
```

Save and close.

Note: It is vital that we sleep for 60 seconds before running, as the script fails out of the gate otherwise. (Not sure why.)

Finally, update Phabricator's configuration to expect this user to run the daemons.

```
$ sudo /opt/phab/phabricator/bin/config set phd.user phabdaemon
```

Of course, we can run this to start the Phabricator daemons right now...

```
$ sudo /opt/scripts/phab/phd_start
```

Note: If it complains about not being able to modify a path starting with `/var/tmp/phd`, just CAREFULLY run `sudo rm -r /var/tmp/phd`.

Phabricator Aphlict Notification Server

Aphlict is our notification server, which we now need to install and configure.

```
$ sudo apt install nodejs npm
$ cd /opt/phab/phabricator/support/aphlict/server/
$ npm install ws
```

You can safely ignore the warning messages from `npm`.

Next, we'll add the `phabdaemon` user to the group that can view the SSL certificates.

```
$ sudo usermod -a -G certs phabdaemon
```

Now we need to adjust the Aphlict configuration, or it won't start.

```
$ cd /opt/phab/phabricator/conf/aphlict
$ cp aphlict.default.json aphlict.custom.json
$ vim aphlict.custom.json
```

The file should look like this:

```
{
  "servers": [
    {
      "type": "client",
      "port": 22280,
      "listen": "0.0.0.0",
      "ssl.key": "/etc/apache2/certs/privkey.pem",
      "ssl.cert": "/etc/apache2/certs/fullchain.pem",
      "ssl.chain": null
    },
    {
      "type": "admin",
      "port": 22281,
      "listen": "127.0.0.1",
      "ssl.key": null,
      "ssl.cert": null,
      "ssl.chain": null
    }
  ],
  "logs": [
    {
      "path": "/opt/log/phab/aphlict.log"
    }
  ],
  "pidfile": "/var/tmp/aphlict/pid/aphlict.pid"
}
```

Finally, open the necessary port and start Aphlict via…

```
$ sudo ufw allow 22280
$ cd /opt/phab/phabricator
$ sudo -u phabdaemon ./bin/aphlict start
```

It should start up without any issues. If there are some, check the previous steps.

Finally, we need to tell Phabricator to use Aphlict. In Phabricator, go to Config→All Settings (`https://phab.qub3d.org/config/all`). Look for `notification.servers`. Enter the following in the field:

```
[
    {
    "type": "client",
    "host": "phab.qub3d.org",
    "port": 22280,
    "protocol": "https"
    },
    {
    "type": "admin",
    "host": "127.0.0.1",
    "port": 22281,
    "protocol": "http"
    }
]
```

Navigate to the Notification Servers section of Config (`https://phab.qub3d.org/config/cluster/notifications/`) to ensure the system is running correctly.

If all's well, let's add the Aphlict startup to our PHD start script.

```
$ sudo vim /opt/scripts/phab/phd_start
```

Add the line…

```
sudo -u phabdaemon /opt/phab/phabricator/bin/aphlict start > /opt/log/phab/phd_start.
↪log
```

Save and close.

SOURCE: Notifications Setup and Configuration (Phabricator)

Phabricator Git SSH

The system already has a `www-data` user, and we set up a `phabdaemon` user earlier. We'll use both of those for use for this. We also need to add a `git` user, and then give these users appropriate sudo permissions.

```
$ sudo useradd -m git
$ sudo /opt/phab/phabricator/bin/config set diffusion.ssh-user git
$ sudo visudo
```

Add these lines to that file:

```
# Configuration for Phabricator VCS
www-data ALL=(phabdaemon) SETENV: NOPASSWD: /usr/bin/git, /usr/lib/git-core/git-http-
↪backend
git ALL=(phabdaemon) SETENV: NOPASSWD: /usr/bin/git, /usr/bin/git-upload-pack, /usr/
↪bin/git-receive-pack
```

Note: We had to comment out the recommended version for `git` and put in the second version, in order for SSH to work with our repositories. We need to find out what all binaries `git` is needing to use, and add them to the first path. When this is acheved, be sure to swap the comments…do NOT leave them both uncommented!

Also ensure that if there is the line `Defaults requiretty`, it is commented out. If it's not there, we're good.

Save and close.

Now, we need to edit a couple other files.

```
$ sudo vim /etc/shadow
```

Find the line for `git` and change the `!` in the second field to `NP`. Save and close.

Next, run…

```
$ sudo vim /etc/passwd
```

Find the line for `git` and set (or change) the last field to `/bin/sh`. Save and close.

Let's also add the `git` user to our `phab` group, so it can write to logfile locations.

```
$ sudo usermod -a -G phab git
```

Now let's configure the ports and SSH settings.

```
$ sudo /opt/phab/phabricator/bin/config set diffusion.ssh-port 2222
$ sudo ufw allow 2222
```

Now we need to copy the SSH hook script to our scripts directory. We will need to create a special subdirectory that is owned by root and has permissions `755`, otherwise it won't start.

```
$ cd /opt/scripts
$ sudo chown root:root /opt/scripts
$ sudo mkdir -p root_scripts
$ sudo chmod 755 root_scripts
$ cd root_scripts
$ sudo cp /opt/phab/phabricator/resources/sshd/phabricator-ssh-hook.sh ./phabricator-
↪ssh-hook
$ sudo chmod 755 ./phabricator-ssh-hook
$ sudo vim /opt/scripts/root_scripts/phabricator-ssh-hook
```

Edit that file so it matches the following…

```
#!/bin/sh

# NOTE: Replace this with the username that you expect users to connect with.
VCSUSER="git"

# NOTE: Replace this with the path to your Phabricator directory.
ROOT="/opt/phab/phabricator"

if [ "$1" != "$VCSUSER" ];
then
exit 1
fi

exec "$ROOT/bin/ssh-auth" $@
```

Save and close. Now we need to set up SSHD's configuration.

```
$ sudo cp /opt/phab/phabricator/resources/sshd/sshd_config.phabricator.example /etc/
↪ssh/sshd_config.phabricator
$ sudo vim /etc/ssh/sshd_config.phabricator
```

In that file, set the following lines:

```
AuthorizedKeysCommand /opt/scripts/root_scripts/phabricator-ssh-hook
AuthorizedKeysCommandUser git
AllowUsers git

# You may need to tweak these options, but mostly they just turn off everything
# dangerous.

Port 2222
```

Save and close.

Now we try running SSHD in debug mode first.

```
$ sudo /usr/sbin/sshd -d -d -f /etc/ssh/sshd_config.phabricator
```

Make sure you've added your SSH public key to your Phabricator profile. Then, on the guest computer you use for SSH, run…

```
echo {} | ssh git@phab.qub3d.org -p 2222 conduit conduit.ping
```

After all is said and done, it should print out something like `{"result":"qub3d","error_code":null, "error_info":null}`.

Note: If it gives the message "Could not chdir to home directory /home/git: No such file or directory", that means you didn't create the `git` user with a home directory. If that's the case, you can add one by running `$ sudo mkhomedir_helper git` (on the server).

Once you're assured of this working, run…

```
$ sudo /usr/sbin/sshd -f /etc/ssh/sshd_config.phabricator
```

Double-check functionality by re-running the earlier command on the computer you SSH from. Run this two or three times to be certain.

```
echo {} | ssh git@phab.qub3d.org -p 2222 conduit conduit.ping
```

If it works, then all's well! Add the sshd start command to the system cron.

```
$ sudo crontab -e
```

On that file, add the line:

```
@reboot /usr/sbin/sshd -f /etc/ssh/sshd_config.phabricator
```

Save and close.

Hook Up Mailer

We now need to configure Phabricator to send email. (`thepassword` should be replaced with the actual password for the email account you're connecting to).

```
$ cd /opt/phab/phabricator
$ sudo ./bin/config set metamta.default-address noreply@qub3d.org
$ sudo ./bin/config set metamta.domain qub3d.org
$ sudo ./bin/config set metamta.reply-handler-domain qub3d.org
$ sudo ./bin/config set phpmailer.smtp-host mail.qub3d.org
$ sudo ./bin/config set phpmailer.smtp-port 465
$ sudo ./bin/config set phpmailer.smtp-protocol ssl
$ sudo ./bin/config set phpmailer.smtp-user noreply@qub3d.org
$ sudo ./bin/config set phpmailer.smtp-password thepassword
```

That's it! Now Phabricator can send email. (Test by going to your Profile and Manage, and then click *Send Welcome Email*).

Phabricator Auto Update Scripts

The Phabricator needs to be consistently updated to receive the latest patches. The easiest way to do this is to run a cron job once a day.

The contents of the script would be:

```
#!/bin/bash
PHAB=/opt/phab

# Stop Phabricator and all related services.
$PHAB/phabricator/bin/phd stop
$PHAB/phabricator/bin/aphlict stop
service apache2 stop

# Pull down latest version.
cd $PHAB/arcanist      # Navigate to the Arcanist folder and pull the latest updates from the corresponding repository.
git pull
cd $PHAB/libphutil     # Navigate to the 'libphutil' folder and pull the latest updates from the corresponding repository.
git pull
cd $PHAB/phabricator   # Navigate to the Phabricator folder and pull the latest updates from the corresponding repository.
git pull

# Upgrade databases
$PHAB/phabricator/bin/storage upgrade --force

#Restart the whole kit-and-caboodle
service apache2 start
/opt/scripts/phab/phd_start
```

Save it in /opt/scripts/phab/ as phab_update

Then run…

```
$ sudo crontab -e
```

Add the following line to the end:

```
00 06 * * * /opt/scripts/phab/phab_update
```

A Guide on Defining Cron Jobs

The Phabricator will now automatically update itself at 6am every morning, unless specified otherwise.

Generate Documentation

It is useful to have the Phabricator documentation on hand.

```
$ cd /opt/phab/phabricator
$ sudo ./bin/diviner generate
```

WOO HOO! Phabricator is set up! You can now proceed with the fun part… using the web interface to configure it to your heart's desire! Everything should *just work*. Par-tay time!

Etherpad

Next, we'll set up Etherpad. We are going to be using Docker to run Etherpad so the only dependencies are Docker and Docker Compose.

```
$ sudo apt-get install \
  apt-transport-https \
  ca-certificates \
  curl \
  python-pip \
  software-properties-common
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
$ sudo apt-key fingerprint 0EBFCD88
$ sudo add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) \
  stable"
$ sudo apt-get update
$ sudo apt-get install -y docker-ce
$ sudo pip install docker-compose
$ sudo pip install --upgrade pip
```

Before we configure Etherpad, we need to set up the database. Be sure to replace `PASSWORD` with a secure password for the etherpad mysql user.

```
$ mysql -u root -p
create database `etherpad-lite`;
grant all privileges on `etherpad-lite`.* to 'etherpad'@'localhost' identified by
↪'PASSWORD';
exit
```

Configuring Etherpad

Now we create a directory that will contain the Etherpad config and our Docker Compose file.

```
$ sudo mkdir /opt/etherpad
$ sudo chown itdude:www-data /opt/etherpad
```

We pull down the Docker image and create our base etherpad config file from the `settings.json.template` in the image.

```
$ cd /opt/etherpad
$ sudo docker pull tvelocity/etherpad-lite
$ sudo docker run -it --rm --entrypoint="cat" tvelocity/etherpad-lite cat settings.
↪json.template > settings.json
```

Next, we need to edit the configuration…

```
$ cp /opt/etherpad
$ vim settings.json
```

Comment out or remove the following section…

```
"dbType" : "dirty",
  //the database specific settings
  "dbSettings" : {
                   "filename" : "var/dirty.db"
                 },
```

Edit the next section so it matches the following, where `PASSWORD` is the SQL password you specified earlier. Make sure you also are uncommenting this section!

```
/* An Example of MySQL Configuration */
"dbType" : "mysql",
"dbSettings" : {
                "user"    : "etherpad",
                "host"    : "localhost",
                "password": "PASSWORD",
                "database": "etherpad-lite",
                "charset" : "utf8mb4"
              },
```

Also, look for the following section, and change the passwords. Again, make sure you are uncommenting this section!

```
/* Users for basic authentication. is_admin = true gives access to /admin.
 If you do not uncomment this, /admin will not be available! */
"users": {
"admin": {
  "password": "changeme1",
  "is_admin": true
}
},
```

For security reasons, let’s **not** show the contents of this file in the admin panel.

```
// Option to hide/show the settings.json in admin page, default option is set to true
"showSettingsInAdminPage" : false,
```

Finally, look for this section and change it to match the following, thereby pointing to our Let’s Encrypt certificates which will be mounted into the container.

```
/*
// Node native SSL support
// this is disabled by default
//
// make sure to have the minimum and correct file access permissions set
// so that the Etherpad server can access them
*/

"ssl" : {
        "key"  : "/certs/privkey.pem",
        "cert" : "/certs/cert.pem",
        "ca": ["/certs/chain.pem"]
      },
```

Next, we’ll create the `docker-compose.yml` that will stand up the Etherpad container.

```
$ vim docker-compose.yml
```

Paste the following into the file.

```
version: '2'
services:
  etherpad-lite:
    image: tvelocity/etherpad-lite
    restart: always
    network_mode: host
    ports:
      - 9001:9001
```

(continues on next page)

(continued from previous page)

```
    volumes:
      - ./settings.json:/opt/etherpad-lite/var/settings.json
      - /etc/apache2/certs:/certs
      - etherpad-plugins:/opt/etherpad-lite/node_modules
    entrypoint: "bin/run.sh"
    command:
      - "--root"

volumes:
  etherpad-plugins:
    driver: local
```

Now we can run Etherpad for the first time.

```
$ sudo docker-compose up -d
```

You can follow the logs as Etherpad starts up.

```
$ sudo docker-compose logs -f
```

If everything looks fine, you can stop the container.

```
$ sudo docker-compose stop
```

Now we need to make some more changes to the database.

```
$ mysql -u root -p
alter database `etherpad-lite` character set utf8 collate utf8_bin;
use `etherpad-lite`;
alter table `store` convert to character set utf8 collate utf8_bin;
exit
```

With all that done, let's allow the Etherpad port through, and then start Etherpad again.

```
$ sudo ufw allow 9001
$ sudo docker-compose up -d
```

Navigate to `https://qub3d.org:9001` to test that Etherpad is working.

SOURCE: Install Etherpad web-based real time collaborative editor on Ubuntu 16.04 Linux (LinuxConfig.org)

SOURCE: How to Install Etherpad For Production with Node.js and MySQL on a VPN (DigitalOcean)

SOURCE: tvelocity/etherpad-lite Docker image on Docker Hub (Hub.Docker.com)

Apache2 Proxy

Let's set up a proxy for Etherpad.

```
$ sudo vim /etc/apache2/sites-available/pad.conf
```

Set the contents of that file to the following…

```
<IfModule mod_ssl.c>
    <VirtualHost *:443>
        ServerName pad.qub3d.org
```

(continues on next page)

(continued from previous page)

```
        ServerAdmin webmaster@qub3d.org

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        SSLEngine on
        SSLCertificateFile  /etc/apache2/certs/fullchain.pem
        SSLCertificateKeyFile /etc/apache2/certs/privkey.pem
        Include /etc/letsencrypt/options-ssl-apache.conf

        <IfModule mod_proxy.c>
            # the following allows "nice" urls such as https://etherpad.example.org/
↪padname
            # But, some users reported issues with this
            RewriteEngine On
            RewriteRule /p/*$ https://pad.qub3d.org/ [NC,L]
            RewriteCond %{REQUEST_URI} !^/locales/
            RewriteCond %{REQUEST_URI} !^/locales.json
            RewriteCond %{REQUEST_URI} !^/admin
            RewriteCond %{REQUEST_URI} !^/p/
            RewriteCond %{REQUEST_URI} !^/static/
            RewriteCond %{REQUEST_URI} !^/pluginfw/
            RewriteCond %{REQUEST_URI} !^/javascripts/
            RewriteCond %{REQUEST_URI} !^/socket.io/
            RewriteCond %{REQUEST_URI} !^/ep/
            RewriteCond %{REQUEST_URI} !^/minified/
            RewriteCond %{REQUEST_URI} !^/api/
            RewriteCond %{REQUEST_URI} !^/ro/
            RewriteCond %{REQUEST_URI} !^/error/
            RewriteCond %{REQUEST_URI} !^/jserror
            RewriteCond %{REQUEST_URI} !^/redirect
            RewriteCond %{REQUEST_URI} !^/favicon.ico
            RewriteCond %{REQUEST_URI} !^/robots.txt
            RewriteCond %{REQUEST_URI} !^/list/
            RewriteCond %{REQUEST_URI} !^/public/
            RewriteRule ^/p/(.+)$ https://pad.qub3d.org/p/$1 [L]

            SSLProxyEngine On
            SSLProxyVerify none
            SSLProxyCheckPeerCN off
            SSLProxyCheckPeerName off
            SSLProxyCheckPeerExpire off

            ProxyVia On
            ProxyRequests Off
            ProxyPass / https://qub3d.org:9001/
            ProxyPassReverse / https://qub3d.org:9001/
            ProxyPreserveHost on
            <Proxy *>
                Options FollowSymLinks MultiViews
                AllowOverride All
                Order allow,deny
                allow from all
            </Proxy>
        </IfModule>
    </VirtualHost>
</IfModule>
```

Save and close, and then load it up.

```
$ sudo a2ensite pad.conf
$ sudo a2enmod proxy
$ sudo a2enmod proxy_http
$ sudo systemctl restart apache2
```

Go to `https://pad.qub3d.org` to check that the new site works.

Adding Plugins

We can add various plugins to Etherpad. Simply go to `https:/pad.qub3d.org/admin` and select `Plugin Manager`.

We are enabling the following plugins:

- activepads
- adminpads
- authorship_toggle
- autocomp
- cursortrace
- print
- scrollto
- wrap

2.2 Qub3d Standards

2.2.1 C++ Coding Standards

This document outlines the current coding standards that should be followed when writing C++ code for the Qub3d project.

The aims of these conventions are to:

- Ensure code readability, allowing developers to understand new code quickly and easily
- Ensure code is cross compiler compatible
- Make the code easier to maintain (roughly 80% of the lifetime of a codebase goes to maintenance)

Commenting

For any comment that takes up a single line use `//` where the `//` and comment is separated by a single space. For example:

```
// This is a single line comment
int doSomething();
```

For multiline comments use `/* */` where the `/*` and `*/` are on their own separate lines and the comment text is indented - where each line starts with a `*`. A correct example would be:

```
/*
 *   This is a multiline comment
 *   As in a comment with multiple lines.
 *   Etc...
 */
```

Care should be taken to ensure that all comments are descriptive and as a rule of thumb should say *why* code is there and not *just* what it does.

Naming

Files

C++ Source files should use the `.cpp` file extension and Header files should use the `.hpp` file.

File names should follow camel case naming, where the first world begins with a lowercase letter and following words being with a capital. An example of which is as follows:

```
qub3dHeaderFile.hpp
qub3dSourceFile.cpp
```

Functions

Functions and methods names should also use camel case (lowercase first letter and subsequent words start with a capital). This is an acceptable function name:

```
int functionName();
```

and these are not:

```
void FunctionName();
void function_name();
void Function_Name();
etc...
```

Function names should be descriptive and describe the purpose of the function. For instance, a function to calculate player health should look like this `calculatePlayerRemainingHealth()` rather than a generic `calculate`.

Variables

In a similar manner, functions names should use camel case and the name should describe what the variable is for. For example:

```
int playerHealth = 430;
```

and **not**:

```
std::string random_variable_name;
int AnotherVariable;
```

Do **not** use any form of Hungarian notation when naming variables (e.g. encoding information a variable and its type in its name - such as `int iHealth;` or `std::string* spName`)

The only valid prefixes for variable names are

`g_`	For global variables.
`m_`	For private class member variables.

Namespaces

Namespaces should also use camel case - where the first letter is lowercase and any subsequent words start with a capital. For example:

```
namespace qub3dEngine
{
} // namespace qub3dEngine
```

and **not**

```
namespace Qub3dEngine
{}

namespace qub3d_engine
{}
```

The second brace in a namespace declaration should also have a comment on the same line that signifies which namespace it belongs to - in the format `} // namespace <name-of-the-namespace>`. An example is given above in the first example.

Constants and Macros

Constants should follow the convention of being all upperspace characters, where each word is broken via an underscore. For example:

```
const int TOTAL_PLAYER_HEALTH = 1000;
```

and **not**

```
const int playerHealth = 0001;
```

Macros should also follow this convention. For example:

```
#define STRINGIFY(str) #str
#define SOME_CONSTANT (666)
```

Classes

Classes should be named using Pascal Case - where the first letter is _capitalised_ and the first letter of each subsequent word is also capitalised. For instance:

```
class ClassNamesUsePascalCase
{
public:
    void whereasFunctionsUseCamelCase();
};
```

and **not**

```
class DO_NOT_USE_THIS {};

class or_this {};

class orEvenThis {};
```

Formatting

Braces

All braces should be on their own lines with the brace aligned with the start of the statement (e.g not indented). For example:

```
if (someBoolean)
{
}

void functionName()
{
}

namespace qub3d
{
} // namespace qub3d
```

and **not**

```
int getHealth(){
}
```

or

```
if (condition)
    {
    }
```

Pointers

Pointer operators should always be aligned next to the type and not the variable name. For example:

```
char* someString;
```

and **not**

```
char * someString;
char *someString;
```

Indentation

Tabs should be used for indentation and **not** spaces. Tabs can be configured to whatever size the developer wants in their own editor - making them more flexible than spaces.

Class access modifiers should not be indented - for example like this:

```
class Dave
{
public:
    Dave();

protected:
    virtual void virtualFunction();

private:
    int m_memberVariable;
};
```

and **not** this

```
class Dave
{
    public:
        Dave();

    protected:
        virtual void virtualFunction();

    private:
        int m_memberVariable;
};
```

There should also always be a space between a condition keyword and the start of the expression. For example:

```
if (condition) { ... }

while (condition) { .. }
```

and **not**

```
if(condition){}

for(int x=0; x<10;x++){}
```

C++ Constructs

Namespaces

- Do not use `using namespace std;` at global scope
- Use of the `using namespace` is acceptable for other namespaces, however, developers should use their discretion - it should only be used where it does not pollute the global scope *too* much
- `using namespace` is acceptable within a function body.

Dependencies

- Always use `#pragma once` at the start of header files to prevent against multiple includes. Do not use `#ifndef` guards instead.

- Try to minimise coupling in general.
- Use forward declarations instead of including a header, if possible.

Modern C++ Features

In order to be the most cross compiler compatible as possible stick to using `C++11` - many compilers do not support more modern features.

Here are some examples of C++11 features that are supported, and encouraged.

- Using range based for loops - for example `for(auto& x : list)` - is encouraged to be used wherever possible as it keeps code easier to maintain and understand, compared to directly writing iterator based loops directly.
- Use the `auto` keyword when it simplifies declarations and makes code easier to understand - for example when dealing with iterators, lambdas or in template code where the type of an expression cannot be easily discerned.
- Lambdas are encouraged when it makes the code simpler - however, they should be kept short.

Miscellaneous

This is the collection of style and standards to follow that do not fit in any major categories.

- Use `nullptr` instead of `NULL` dealing with null pointers.
- Initialise default values for member variables in the class member initialisation list (and not in the class declaration)

```
struct Dave
{
    Dave():
        m_variable1(123),
        m_variable2("Hello World")
    {}

    int m_variable1;
    std::string m_variable2;
}
```

- `const` should some before the type and not after - e.g `const int CONSTANT = 123;` and *not* `int const CONSTANT = 123`.
- Never submit anything that includes commented out code - unless it describes some part of the code.
- Line length - try to keep lines under 80 characters. You may have more than 80 characters on one line, but no more than 120.

Includes

Use `#include <>` for library headers (e.g any third party headers). Use `#include ""` for any headers belonging to that codebase.

Third party header includes should also come before any internal includes in a file. Such as:

```
#include <thread>
#include <iostream>

#include "gameEngine.hpp"
```

When including C standard header files from C++ use the C++ version - for example `#include <cstdio>` and *not* `#include <stdio.h>`

Conditions

Conditions should always read left to right. For example

```
if (result != GLEW_OK) { ... }
```

and **not**

```
if (GLEW_OK != result) { ... }
```

2.2.2 Lua Coding Standards

25 Febuary 2018, 16:50 GMT

This file is dedicated to Lua format standards provided by the Qub3d Engine Group.

Format Standards

Localization

The code must be written in ASCII. We use the English ASCII localization standard for writing Lua. Strings may be in unicode, preferably UTF-8, unless you insert a unicode character that ASCII doesn't support. For more information on which characters are supported by ASCII, see the tables.

2.2.3 Terminology

Tuesday, 21 March 2018, 02:32 GMT

Client Pack

A combonation of the following Texture Packs override the textures of the Client, Game, and Mods according to the users choice. Shader Packs override the shaders of the Client, and/or Game according to the users choice. Model Packs override the models for entities (and their textures) in the Game and Mods according to the users choice. Audio Packs override the sounds of the Client, Game, and Mods according to the users choice.

Engine

A library written in C++, which provides the core functionality to the Client for users to create voxel-based games.

Game

Written in Lua, these define the content that the Client displays: mobs, blocks, recipes, plants, biomes, structures, etc. They can be extended by Mods.

Git

The software used for version control, more information can be found at the git website.

Landing - Version Control

When a Diff Request has been accepted and merged with the main branch. Synonymous with merging on GitHub.

Launcher

An executable compiled from C++, that of which makes everything run. It is what you launch to get to game.

Launching - Version Control

When you are posting your code requesting for it to be merged with the repository. Synonymous with creating a Pull Request on GitHub.

Mods

Written in Lua and YAML, these are addons/modifications to a Game. After starting the client, you select a Game (or make your own) and the Mods that you want to load. The Client then runs the selected Game with the selected compatible Mods.

Qube

A single block in the world.

2.3 Miscellaneous

2.3.1 Frequently Asked Questions

3 March 2018, 01:05 GMT

This file is dedicated to addressing questions that get asked frequently across the Qub3d community.

Begin FAQ

Why this FAQ?

This file tries to answer questions before they're asked. This can save time and trouble of asking them and/or the trouble of encountering a problem. This can significantly reduce traffic in the issue/question-tracking boards. This can be useful for both you and the Qub3d Engine Group.

What is Qub3d's goal?

Its goal is to be an independent, free, and open source voxel game that is actually *fun* while taking inspiration from Minetest and Minecraft™. For more information in regards to this, see the wiki article about it.

What's Phabricator?

Phabricator is a collaborative Task Managing/Source Code Hosting platform that lags behind Github in terms of popularity. It is the development platform in use by the Qub3d Engine Group.

How do I create a Phabricator account?

Go to our Phabricator and register your account there with Github. If you don't have a Github account, then make one. It's free and grants you access to most of the Open Source development community. To register your account, look for *Log In or Register (GitHub)* and click on it.

How is "Qub3d" pronounced?

It has the exact same pronounciation as "cubed" with a hard "c" and the English standard for the suffix "-ed." To put this in other "words," it is pronounced as "k-you-bd." The "3" in "Qub3d" is pronounced "cubed" as well.

Where do I report bugs?

We take bugs very seriously. You can report bugs/issues at Maniphest after registering an account. Please be very descriptive when filing bugs.

Where can I find more information about Qub3d?

Much of the official Qub3d documentation can be found on the official wiki.

When will it be done? Why is it taking so long?

It all has to do with the nature of the software. See Why Is It Taking So Long??!?.

What is the Qub3d documentation licensed under?

It is licensed under CC BY-SA 4.0. This license also expands to all non-code content by the Qub3d Engine Group and its contributors.

How can I contribute?

The Contributors' Guide is all you need to get started with contributing to the Qub3d project.

This FAQ has not answered my question. What do I do now?

A very good step is to browse through the relevant documentation in the wiki. Another way is to read further into the development documentation we have at Read the Docs. You can ask us or our community questions on our Freenode IRC channel #qub3d. If all else fails, visit the Qub3d Engine Group at the Discord Server.

2.3.2 Checklist For Submitting Code

1. Accomplish the feature(s) it was designed to accomplish.
2. Have merged all changes from *master* into itself, and all conflicts resolved. (*$ git pull origin master*)
3. Have binaries and unnecessary cruft untracked and removed. (Keep an eye on *.gitignore*!)
4. Compile and run properly.
5. Be free of compiler errors and warnings (must compile with *-Wall -Wextra*).
6. Be Valgrind pure (no memory leaks detected).
7. Comply with Coding Standards/Style.
8. Be free of linter errors. (*$ arc lint –lintall*)
9. Be fully CSI commented.
10. Have an up-to-date build script (generally CMake) if relevant.
11. Contain relevant LIT tests on Goldilocks.
12. Have a Test Plan, generally containing a list of Goldilocks tests the reviewer should run.
13. Be reviewed, built, tested, and approved by at least one Trusted review (Staff or Contributor).
14. Have up-to-date Sphinx documentation, which compiles with no warnings.
15. Have all reviewer comments processed and marked "Done".
16. For bug fixes, please show a way of demonstrating that the diff actually fixes something.
17. If the contributor doesn't run the Goldilocks test suite on the diff, then the maintainer will.
18. If the diff fixes a bug reported in Maniphest, a brief reference to that bug must be included in the Summary.
19. Have tests run by Jenkins CI pass properly.

If you are unfamiliar with CSI, then please read the documentation for it by MousePaw Media.

You must also abide by the C++ and Lua coding standards/style provided by the Qub^{3}d Engine Group. For more information on our Coding Standards/Style, see the C++ Coding Standards and the Lua Coding Standards.

Before launching any significant diff, please double check to see if there is an issue with a *Help Wanted* tag that describes your intention, has been approved, and was not assigned to anyone else. However, if there is no such issue, create a new one in Maniphest. If there is an issue that wasn't assigned to anyone, simply leave a comment behind stating that you wish to work on it, and a Trusted Member will assign it to you, or you can scroll to the bottom of the issue page and click on *Actions...* → *Assign/Claim* to show others that you are officially working on the issue. If you're submitting a minor bug fix, documentation change, and/or other miniscule changes, there is no need to create an issue, just launch the diff.

If there aren't any bugs in Maniphest to fix, then find a new bug by running tests with the repositories, or maybe you can casually use them and spot bugs that way. There's always never a shortage of bugs to fix in any project.

If you get the `No paths are lintable.` error after typing `arc lint --lintall`, don't worry about it, for you can ignore it.

If Jenkins fails to pass the test properly, please find out why. The Qub^{3}d Engine Group will not let failed tests pass through the gates to landing for any reason.

2.3.3 Qub3d Engine CMake Documentation

This file attempts to show you how to use Qub3d's CMake build system without having to read CMakeLists.txt for reference. The default build process is in the *##Building* section in README.md in the engine's repository and the *Basic Build* section in this file.

Preliminary Requirements

- GCC to be at least 5.x.
- Clang to be at least 2.0.
- CMake to be at least 3.0.2.

Basic Build

The default way of building the engine is this:

```
$ mkdir build/
$ cd build/
$ cmake .. #-DWHATEVER_OPTION=ARGUMENT
$ make -j$(nproc)
```

CMake Options

Option	Arguments	Description
`-DCMAKE_BUILD_TYPE`	`Debug` or `Release`	Build types.
`-D32BIT`	`ON` or `OFF`	Sets up 32-bit build.
`-D64BIT`	`ON` or `OFF`	Sets up 64-bit build.

Note: `-D32BIT` and `-D64BIT` cannot be used in the same build.

Default Arguments

`-DCMAKE_BUILD_TYPE` : Release

`-D32BIT` : OFF

`-D64BIT` : OFF

Default Compiler Flags

This section is dedicated to compiler flags on each compiler, option.

Default Build

The flags for the default build are shown below:

```
-O3 -DNDEBUG -Wall -Wextra -std=c++17 -std=c11
```

Flags for the debug build:

```
-O0 -g -DNDEBUG -Wall -Wextra -std=c++17 -std=c11
```

GCC-Specific

If you have the GNU Compiler Collection newer than 5.x installed, then by default, the `-s` option is incorporated only in the default build. In the debug build, the `-fprofile-arcs -ftest-coverage` options are incorporated in order to support Gcov. If you have GCC 7.x and above, then the `-ggnu-pubnames` flag is incorporated to make debugging even easier; the flag creates the files, `.debug_pubnames` and `.debug_pubtypes`, in a format for conversion to a GDB index format. GDB must be at least version 7 for this option to take effect. Overall, a flag setup with GCC is this:

- GCC 5.x default:

```
-O3 -DNDEBUG -Wall -Wextra -std=c++17 -std=c11 -s
```

- GCC 5.x debug:

```
-O0 -g -DNDEBUG -Wall -Wextra -std=c++17 -std=c11 -fprofile-arcs -ftest-coverage
```

- GCC 7.x+ debug:

```
-O0 -g -DNDEBUG -Wall -Wextra -std=c++17 -std=c11 -fprofile-arcs -ftest-coverage -
↪ggnu-pubnames
```

Clang-Specific

If you have Clang installed, no flags will be incorporated; however, if you *do* have LLVM installed along with Clang, then the `-stdlib=libc++` flag is incorporated and you will need to install `libc++` if you haven't already. This only affects C++ files.

Dependencies

Here, CMake looks into certain directories for packages that the Qub3d repositories depend on.

PawLIB

CMake looks for PawLIB in `qub3d-libdeps` after it has been built. `qub3d-libdeps` must be in the same working directory as the engine for that to happen.

CPGF

CMake looks for CPGF in the `qub3d-libdeps` folder that is paralell to the Qub3d repositories. That is, after you build it.

SDL2

It looks for SDL2 in `qub3d-libdeps` after it has been built.

CHAPTER 3

Indices and Tables

- genindex

CHAPTER 4

Feedback

This documentation is written and maintained by the Qub3d Engine Group. Feedback is welcome via our Phabricator.

For more information about contributing to the Qub3d Engine Group's various projects, see our Contributors' Guide.

Index