Software Install Guides

Release 3.8.0

May 03, 2020

Debian-Based Documentation

These how-to guides, at least, started off as a personal means of documenting my way through my media server installations. That way I wouldn't have to hop around the net near as much, remembering and forgetting what and where.

Plus, working on this helps to keep the documentation and sourcing fresh.

If you have any suggestions for improvements, wanna submit edits, or just wanna leave a nice comment, please visit my Software-Install-Guides Github Page and send me pull requests or submit an issue.

A lot of my guides have been copied from HTPC Guides. This site is wonderful with a wealth of info, and is constantly updated with information and new guides.

Table of contents:

CHAPTER 1

Debian / Ubuntu

These guides are tested and used often by me on my personal Ubuntu 16.04 Server VM and Debian Stretch-based Raspberry Pi's. But, since Ubuntu is based off of Debian, it wouldn't take much to be able to use these as a more generalized guide for most Debian-based systems.

I have this section broken down further into specific area's of interest:

1. *WebApps* - These are programs that you have to use in the web browser to do anything with them. Normally you cannot use the terminal to run these programs, and normally they don't include a typical GUI - or to run from the desktop.
2. *Downloaders* - These are guides to setup your downloading programs, such as for torrents or Usenet services. This also has my OpenVPN guide, as this is the specific machine that would need to have encrypted traffic.

- PS. the OpenVPN guide is for setting up a client, not a server.

1. *Web Server Stuff* - This is for setting up the actual web server backend. This is the reverse proxy stuff so you can map your port numbers to web addresses for easy typing and remote access.
2. *APT or DPKG* - These guides are specifically geared towards Debian's `apt-get`-based installation systems, including `dpkg`

1.1 APT or DPKG

The docs here apply to the Debian family of `apt`, `apt-get`, and `dpkg`, and any/all of the plethora of commands and programs that fall under those umbrellas.

1.1.1 Adding Extra apt-get Lists

With Ubuntu, there are many different ways to add additional apt-get lists, either directly by `sudo nano ...`, `sudo add-apt-repository ppa:nginx/main`, etc.

Release-Specific Lists

When adding an `apt-get` list to your system, one nice way to save your code in your notes or to automate through scripting is not by specific system names, like Ubuntu's `xenial`, and Debian's `Jessie` release names; but rather to insert code into your `echo` so that it works for you!

```
echo "deb https://packages.cisofy.com/community/lynis/deb $(lsb_release -sc) main" |␣
↪sudo tee /etc/apt/sources.list.d/cisofy-linus.list
```

Breakdown

1. Most all apt-get `.list` lines begin with `deb` or `deb-src`
2. Then, the html address of the library
3. Next, usually the name of the release you are using, such as `xenial` for 16.04 Ubuntu or `jessie` for Debian 8.
4. And last, there are names for the various extra sections you can discern between - `main`, `extras` or whatever else the library maintainer uses.

Note: The key text is `$(lsb_release -sc)`. When you input `$( )` it tells bash to execute the command inside the parenthesis, and use the output inside the echo text.

PPA

or more of Ubuntu's Shenanigans

Ubuntu seems to have a small habit of taking industry- and community-standardized processes and libraries and applications and putting - or sometimes shoving - their own special twist on things.

Take Ubuntu's PPA system. As a developer on Ubuntu's Launchpad website, you get your own PPA address, apt repository, and a central means of distributing your code to Ubuntu Users.

Its super simple to add these repo's to Ubuntu:

```
sudo add-apt-repository ppa:nginx/main
sudo apt-get update && sudo apt-get install $application
```

You'll want to always run `apt-get update` to pull the lists of available programs to install, and then install the additional program or to upgrade existing programs already installed

Personal Standards

When I add apt-get lists that are seperate from the standard or even non-standard Ubuntu Lists and Libraries, such as NGINX's lists, nodesource lists for Node and NPM, etc., I have them in seperate, short lists.

The directory tree breakdown is as follows:

```
/etc/apt/sources.list
/etc/apt/sources.list.d/
├── mono-xamarin.list
├── nginx-amplify.list
```

(continues on next page)

(continued from previous page)

```
├── nginx-ubuntu-development-xenial.list
├── nodesource.list
└── ondrej-ubuntu-php-xenial.list
```

This way, removing specific items is MUCH easier.

1.1.2 Personal APT Repo

https://www.howtoforge.com/setting-up-an-apt-repository-with-reprepro-and-nginx-on-debian-wheezy

1.1.3 dpkg -l Package States

This is specifically in reference to the flags at the start of each line when you perform `dpkg -l`.

```
| => dpkg -l
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                Version        Architecture   Description
+++-==================-==============-==============-=========================================
ii  accountsservice     0.6.40-2ubuntu amd64          query and manipulate user account informati
ii  acpid               1:2.0.26-1ubun amd64          Advanced Configuration and Power Interface
ii  adduser             3.113+nmu3ubun all            add and remove users and groups
ii  apparmor            2.10.95-0ubunt amd64          user-space parser utility for AppArmor
ii  apport              2.20.1-0ubuntu all            automatically generate crash reports for de
ii  apport-symptoms     0.20           all            symptom scripts for apport
ii  apt                 1.2.12~ubuntu1 amd64          commandline package manager
rc  apt-show-versions   0.22.7         all            lists available package versions with distr
ii  apt-transport-https 1.2.12~ubuntu1 amd64          https download transport for APT
ii  apt-utils           1.2.12~ubuntu1 amd64          package management related utility programs
rc  apt-xapian-index    0.47ubuntu8    all            maintenance and search tools for a Xapian i
ii  aptitude            0.7.4-2ubuntu2 amd64          terminal-based package manager
ii  aptitude-common     0.7.4-2ubuntu2 all            architecture independent files for the apti
ii  at                  3.1.18-2ubuntu amd64          Delayed job execution and batch processing
ii  augeas-lenses       1.4.0-0ubuntu1 all            Set of lenses needed by libaugeas0 to parse
ii  base-files          9.4ubuntu4.3   amd64          Debian base system miscellaneous files
ii  base-passwd         3.5.39         amd64          Debian base system master password and grou
ii  bash                4.3-14ubuntu1. amd64          GNU Bourne Again SHell
ii  bash-completion     1:2.1-4.2ubunt all            programmable completion for the bash shell
ii  bc                  1.06.95-9build amd64          GNU bc arbitrary precision calculator langu
ii  bind9-host          1:9.10.3.dfsg. amd64          Version of 'host' bundled with BIND 9.X
ii  binutils            2.26.1-1ubuntu amd64          GNU assembler, linker and binary utilities
ii  biosdevname         0.4.1-0ubuntu9 amd64          apply BIOS-given names to network devices
ii  bsdmainutils        9.0.6ubuntu3   amd64          collection of more utilities from FreeBSD
ii  bsdutils            1:2.27.1-6ubun amd64          basic utilities from 4.4BSD-Lite
ii  build-essential     12.1ubuntu2    amd64          Informational list of build-essential packa
ii  busybox-initramfs   1:1.22.0-15ubu amd64          Standalone shell setup for initramfs
ii  busybox-static      1:1.22.0-15ubu amd64          Standalone rescue shell with tons of builti
ii  byobu               5.106-0ubuntu1 all            text window manager, shell multiplexer, int
ii  bzip2               1.0.6-8        amd64          high-quality block-sorting file compressor
ii  ca-certificates     20160104ubuntu all            Common CA certificates
ii  camlp4              4.02.1+3-2     amd64          Pre Processor Pretty Printer for OCaml
ii  cgmanager           0.39-2ubuntu5  amd64          Central cgroup manager daemon
ii  command-not-found   0.3ubuntu16.04 all            Suggest installation of packages in interac
ii  command-not-found-d 0.3ubuntu16.04 amd64          Set of data files for command-not-found.
ii  console-setup       1.108ubuntu15. all            console font and keymap setup program
ii  console-setup-linux 1.108ubuntu15. all            Linux specific part of console-setup
```

In the above image, far left, you can see the column that has `ii` or `rc`. That would be the specific info I'm referencing, and knowing this info doesn't hurt!

Definitions

1st Letter	2nd Letter	Opt. 3rd Letter
u - unknown	n - not-installed	r - reinst-required
i - install	i - installed	
r - remove	c - config-files	
p - purge	u - unpacked	
h - hold	f - half-configured	
.	h - half-installed	
.	w - triggers-awaited	
.	t - triggers-pending	

1.1.4 OS Edition Name

For when you are wanting to add a specific, custom apt sources list item (because its best to have custom items in their own, specific *.list* file) and the string asks for the Ubuntu/Debian Edition Name (Xenial, Jessie, etc.), you are able to run commands within the *echo* line.

```
sudo sh -c 'echo "deb http://archive.getdeb.net/ubuntu $(lsb_release -sc)-getdeb apps
↪" >> /etc/apt/sources.list.d/getdeb.list'
```

Lets break that down:

sudo sh -c

When you use this specific starting command, and then wrap everything after in apostrophes, it "sudos" that entire block of commands. This is awesome for when you're wanting to redirect with > or >> to a file or location that requires sudo.

$(lsb_release -sc)

1. *lsb_release*- this prints distribution-specific info
2. *-sc*- the *-s* is "short", the *-c* is "codename"

So, almost all apt sources addresses have the distribution's code name as one of the options, so using this really helps out in programatic coding.

1.1.5 DPKG or APT General Errors

This is a general catch-all doc for any smaller errors you might come across involving the `apt` and `dpkg` installation systems.

StatOverride File

Sometimes, for a reason unknown to me, you'll get an error while installing, updating/upgrading - what have you - about the `statoverride` file having an unknown user:

```
dpkg: unrecoverable fatal error, aborting:
unknown user 'cockpit-ws' in statoverride file
E: Sub-process /usr/bin/dpkg returned an error code (2)
```

For me, it usually happens after uninstalling a program - which makes sense, seeing as something extra has decided to hang around in our installation program files.

File Name

The specific file that is giving us issues is `/var/lib/dpkg/statoverride`.

How To Fix

There are 2 different ways I found to remove the extra bit of info from the offending file, of which, both require you to at least see the contents of the file:

```
cat /var/lib/dpkg/statoverride
```

– or –

```
sudo nano /var/lib/dpkg/statoverride
```

First Way

The second option is the first way we can fix this issue, by removing the offending information, which in the above case - *StatOverride File* - the issue is about `cockpit-ws`. Below is the output of the `/var/lib/dpkg/statoverride` file.

```
root postdrop 2555 /usr/sbin/postqueue
root sasl 660 /etc/sasldb2
postfix postdrop 2710 /var/spool/postfix/public
root root 4755 /usr/sbin/mount.davfs
root crontab 2755 /usr/bin/crontab
root sasl 710 /var/run/saslauthd
root mlocate 2755 /usr/bin/mlocate
root root 1733 /var/lib/php/sessions
root ssl-cert 710 /etc/ssl/private
root messagebus 4754 /usr/lib/dbus-1.0/dbus-daemon-launch-helper
root cockpit-ws 4750 /usr/lib/cockpit/cockpit-session
root postdrop 2555 /usr/sbin/postdrop
```

You can see the line that includes `cockpit-ws` above. If you have the above file open in your text editor, you can simply delete that line out, and rerun your prior apt command again.

Second Way

The second way to fix it is a more official way, using one of **`dpkg`**'s commands, **`dpkg-statoverride`** - which makes even more sense....

```
dpkg-statoverride --remove /usr/lib/cockpit/cockpit-session
```

For this one - and ironically, this is the ONE item that makes NO sense - you have to use the path that is listed in the file, rather than the specific "user" that **`&apt`** errors out with... And thats why I'm using this option as my second fix, because its more or less an extra step after viewing or editing the `statoverride` file above.

1.2 WebApps

This section is specific for webapp-specific installations and how-to's.

1.2.1 Radarr Movie Downloader

Radarr is built in the likes of what Sonarr used to look like before their redesign.

1.2.2 CouchPotato Movie Downloader [HTPC-CP]

CouchPotato is a web program, built on python, specifically tailored towards automating Movie Downloads, either through public or private torrent sites or using Usenet Services.

I honestly don't even use CouchPotato anymore, as it became a huge process hog and super slow to do everything. I personally use *Radarr Movie Downloader*.

Shoutout

First, I'd like to go ahead and say that I wouldn't have been able to learn as much as I have as quickly or easily without the help of HTPCGuide's CP-HowTo.

They are an amazing site, they are slowly getting larger, and they are the real, awesome source.

Base Requirements

First, lets install the basic required programs to help run all-the-things

```
sudo apt-get install git-core libffi-dev libssl-dev zlib1g-dev libxslt1-dev libxml2-
↪dev python python-pip python-dev build-essential -y
sudo pip install --upgrade lxml cryptography pyopenssl
```

Note: When running pip with sudo, that then installs those specified programs globally, so the entire system has access.

If you are using a more shared environment - where your pip install might interfere with another users python programs - its best to invoke virtualenv from within the directory you are going to save and run the main CouchPotato program from.

This creates a virtual-like environment for installing your python programs within JUST that directory. So, if there are differing versions elsewhere, they wont clash.

Currently, I am not using virtualenv, so that is currently outside the scope of this document.

Clone the Repo

Now, to really kick things off, we're going to first clone the github repo, as this is the - well, only way right now - to install and run the software.

But, the other big plus to this is that for running updates, not only does the program have the ability to simply run `git pull` or what have you from within itself, but if that isn't pulling a fresher update due to other various settings it has, YOU are able to go in and just run `git pull` on the CouchPotato directory.

Which, is why...

Note: I keep all of my cloned git repos inside of one, singular directory:

`~/git`

This way, I don't have to hunt all over my system for where my repo's are and it makes it easier to keep them updated. Then, I symlink the library to wherever either the developer wants/requires it or where is easiest.

The other way of handing this is to clone your PROGRAMS into the /opt directory - so /opt/couchpotato, /opt/NzbDrone, /opt/plexpy and so on. Then clone your working repos for projects into `~/git/[repo]`

Now, onto the cloning:

```
git clone https://github.com/RuudBurger/CouchPotatoServer ~/git/couchpotato
sudo ln -s ~/git/couchpotato /opt/couchpotato
```

Which, again, your other option is to:

```
git clone https://github.com/RuudBurger/CouchPotatoServer /opt/couchpotato
```

Note: See *User Management* for notes on adjusting user permissions with regards to programs and allowing the web access to your machines.

Test if it Works

Now, we'll run the python program just within the Command Line output, which shows all the text output, including any errors and what not.

```
sudo python /opt/couchpotato/CouchPotato.py
```

This will run only as long as you allow it directly inside the terminal, and it will also give each step that the program runs, so you can see if it gives any errors or what else might need to be changed.

Then, to stop the CL output and control, hit `ctrl-C` to quit the program.

Copy/Edit Default File

Note: The `/etc/default` is generally where a lot of programs like to keep their default settings files. Its a nice, centrally located spot that init or systemctl program files can reference when wanting a central place that a user can amend different settings, like the user that is running the program, or the directory location of different files.

So, we want to copy over the defaut `/etc/default` file from the github location, and then make any necessary changes.

```
sudo cp /opt/couchpotato/init/ubuntu.default /etc/default/couchpotato
sudo nano /etc/default/couchpotato
```

The below code field is not the entire file, but rather just an excerpt of items of interest.

```
# COPY THIS FILE TO /etc/default/couchpotato
# Accepted variables with default values -if any- in parentheses:

# username to run couchpotato under (couchpotato)
CP_USER=couchpotato
# directory of CouchPotato.py (/opt/couchpotato)
CP_HOME=/opt/couchpotato
```

```
# directory of couchpotato's db, cache and logs (/var/opt/couchpotato)
CP_DATA=/var/opt/couchpotato
# full path of couchpotato.pid (/var/run/couchpotato/couchpotato.pid)
CP_PIDFILE=/var/run/couchpotato.pid
# full path of the python binary (/usr/bin/python)
PYTHON_BIN=/usr/bin/python
```

1. `CP_USER` would be the system account we created earlier.
2. `CP_HOME` is where it runs from
3. `CP_DATA` is where it stores files like the metadata for your movie directory. This one I like to have stored on a mounted, shared drive. This way, if I ever need to reinstall CouchPotato, or the VM fraks up and needs to be spun fresh, the big time stuff is saved elsewhere. So, mine is `/media/sf_Ext1/shared/couchpotato`

Copy or Edit the init.d File

Now, if you're running Ubuntu, the `./init/ubuntu` script gets copied and amended thusly:

```
sudo cp /opt/couchpotato/init/ubuntu /etc/init.d/couchpotato
sudo chmod +x /etc/init.d/couchpotato
sudo update-rc.d couchpotato defaults
```

So the `chmod +x` makes the file executable - instead of running a bash script as `bash ./script.sh`, when you `chmod +x` it, you're able to just say `./script` and remove the .sh from the file name as well. Then, the system pulls the language from the first line, `#!/bin/bash` or `#!/bin/sh` etc.

Then, the `update-rc.d` inputs the startup script into the actual upstart, startup system, telling ubuntu to run it on boot - if the script wants that.

Then, you can run `sudo service couchpotato start`, and so long as it doesn't output errors, you can now access it at http://127.0.0.1:5050

I will have reverse-proxying stuff posted in the future, but for now you can look at HTPCGuides.com as they have a lot of those specific how-to's.

1.2.3 Tautulli (previously PlexPy) [HTPC-Tautulli]

Tautulli is specifically for monitoring Plex. Plus, you can setup notifications so you can see when Plex adds files, has updates, items are played and stopped and what not, as well as if you have any friends or family who have access.

It also gives you a breakdown of what is played the most, how many concurrent streams have played at the same time, and a bunch of other things as well.

Prerequisites

Python2.7 is the biggy here.

```
sudo apt-get install python2.7 python-pip python-dev git git-core
```

Clone the Repo

Note: I keep all of my cloned git repos inside of one, singular directory:

`~/git`

This way, I don't have to hunt all over my system for where my repo's are and it makes it easier to keep them updated. Then, I symlink the library to wherever either the developer wants/requires it or where is easiest.

The other way of handing this is to clone your PROGRAMS into the /opt directory - so /opt/couchpotato, /opt/NzbDrone, /opt/plexpy and so on. Then clone your working repos for projects into `~/git/[repo]`

So, of course, you can amend the ending to the below code to wherever you want the repo to sit inside your system.

```
git clone https://github.com/Tautulli/Tautulli ~/git/tautulli
```

Note: If you previously installed this git repo before, you'll notice that the entire repo and app name has changed from PlexPy to Tautulli. Both python scripts exist in the beta line of the repo. (Plexpy.py and Tautulli.py) So its really up to you on which naming scheme you want to use, currently. This info - obviously - will change in a future (unknown) update.

Edit the Default File

Now, you want to:

```
sudo touch /etc/default/tautulli
```

The `/etc/default` directory is where a LOT of programs like to store files that make adjustments from the defaulted norm of their programs. Items like the User that the system will run the program under. Or where the program will store `.pid` files, log files, etc.

That will make sure to stop any possible errors or warnings. It also is where you need to make any changes, in case you don't use the default settings that are in the various init scripts. You can see the options inside of `./tautulli/init-scripts/init.ubuntu` if you're using Ubuntu.

Create Tautulli User

Next, I do like to create and use a seperate, `tautulli` user for running Tautulli.

Note: See *User Management* for notes on adjusting user permissions with regards to programs and allowing the web access to your machines.

But, use the `adduser` line below instead of from the User Management doc

---

```
sudo adduser --system --group tautulli --no-create-home tautulli
```

Change Ownerships and Symlink

First, we will change the ownership of the original location we stuck the repo at.

```
sudo chown -R tautulli:tautulli ~/git/tautulli
```

Next, we symlink from our `git` location to `/opt`.

Note: `/opt` seems to be the favorite location to install the directories for WebApps in general - CouchPotato, NzbDrone (aka sonarr), userify, HTPCManager - all like this location. Its a comfy, familiar place to have these files.

```
sudo ln -s ~/git/tautulli /opt/tautulli
```

Testing the Program First

First, we want to, basically, blindly run `tautulli` to see if any settings files need adjusting or anything in our systems.

So, we do:

```
sudo python /opt/tautulli/Tautulli.py
```

This will start Tautulli where it will output all of its startup functions, like when you turn on a linux machine, to the console so you can see any errors right there.

AutoStart System Files

Next, we will perform a few different commands to add Tautulli to Ubuntu's autostart system - so that the program starts at boot correctly, and runs nicely in the system.

init.d

The `/etc/init.d` directory is where the autostart scripts are stored. Thus, why I call them the `init.d` files. There is also a `/etc/init` location for Upstart scripts, which... Ubuntu seems to like to change their autostart programs every so often. And to add ontop of that, the base Debian system has THEIR own means, that is even useable inside Ubuntu - systemctl.

But, in most programs that you use git to install on your system, AND they include autostart files for you, often times the `init.d` based scripts will be referenced as just `init`, making the whole thing slightly more complex. The best way to know for sure if its `init.d` or `init` based? Just edit the file and take a look!

If the file looks like this:

```
#! /bin/sh
### BEGIN INIT INFO
# Provides: NzbDrone
# Required-Start: $local_fs $network $remote_fs
# Required-Stop: $local_fs $network $remote_fs
# Should-Start: $NetworkManager
# Should-Stop: $NetworkManager
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: starts instance of NzbDrone
# Description: starts instance of NzbDrone using start-stop-daemon
### END INIT INFO

############### EDIT ME ##################
# path to app
```

(continues on next page)

(continued from previous page)

```
APP_PATH=/opt/NzbDrone

# user
RUN_AS=<Your UserName>

# path to mono bin
DAEMON=$(which mono)

# Path to store PID file
PID_FILE=/var/run/nzbdrone/nzbdrone.pid
PID_PATH=$(dirname $PID_FILE)

# script name
NAME=nzbdrone

# app name
DESC=NzbDrone

# startup args
EXENAME="NzbDrone.exe"
DAEMON_OPTS=" "$EXENAME

############### END EDIT ME ##################

NZBDRONE_PID=`ps auxf | grep NzbDrone.exe | grep -v grep | awk '{print $2}'`

test -x $DAEMON || exit 0

set -e

#Look for PID and create if doesn't exist
if [ ! -d $PID_PATH ]; then
    mkdir -p $PID_PATH
    chown $RUN_AS $PID_PATH
fi

if [ ! -d $DATA_DIR ]; then
    mkdir -p $DATA_DIR
    chown $RUN_AS $DATA_DIR
fi

if [ -e $PID_FILE ]; then
    PID=`cat $PID_FILE`
    if ! kill -0 $PID > /dev/null 2>&1; then
        echo "Removing stale $PID_FILE"
        rm $PID_FILE
    fi
fi

echo $NZBDRONE_PID > $PID_FILE

case "$1" in
    start)
    if [ -z "${NZBDRONE_PID}" ]; then
        echo "Starting $DESC"
        rm -rf $PID_PATH || return 1
        install -d --mode=0755 -o $RUN_AS $PID_PATH || return 1
```

(continues on next page)

(continued from previous page)

```
        start-stop-daemon -d $APP_PATH -c $RUN_AS --start --background --pidfile
↪$PID_FILE --exec $DAEMON -- $DAEMON_OPTS
    else
        echo "NzbDrone already running."
    fi
    ;;
    stop)
        echo "Stopping $DESC"
        echo $NZBDRONE_PID > $PID_FILE
        start-stop-daemon --stop --pidfile $PID_FILE --retry 15
    ;;
    restart|force-reload)
        echo "Restarting $DESC"
        start-stop-daemon --stop --pidfile $PID_FILE --retry 15
        start-stop-daemon -d $APP_PATH -c $RUN_AS --start --background --pidfile $PID_
↪FILE --exec $DAEMON -- $DAEMON_OPTS
    ;;
    status)
        # Use LSB function library if it exists
        if [ -f /lib/lsb/init-functions ]; then
            . /lib/lsb/init-functions
            if [ -e $PID_FILE ]; then
                status_of_proc -p $PID_FILE "$DAEMON" "$NAME" && exit 0 || exit $?
            else
                log_daemon_msg "$NAME is not running"
                exit 3
            fi
        else
            # Use basic functions
            if [ -e $PID_FILE ]; then
                PID=`cat $PID_FILE`
                if kill -0 $PID > /dev/null 2>&1; then
                    echo " * $NAME is running"
                    exit 0
                fi
            else
                echo " * $NAME is not running"
                exit 3
            fi
        fi
    ;;
    *)
        N=/etc/init.d/$NAME
        echo "Usage: $N {start|stop|restart|force-reload|status}" >&2
        exit 1
        ;;
    esac

exit 0
```

then its an `init.d` file. Specifically with the `###### BEGIN INIT INFO` block of text.

But, if it looks like this:

```
# tautulli
#
# This is a session/user job. Install this file into /usr/share/upstart/sessions
```

(continues on next page)

(continued from previous page)

```
# if tautulli is installed system wide, and into $XDG_CONFIG_HOME/upstart if
# tautulli is installed per user. Change the executable path appropiately.

start on desktop-start
stop on desktop-end

env CONFIG=""$XDG_CONFIG_HOME"/tautulli"
env DATA=""$XDG_DATA_HOME"/tautulli"

pre-start script
    [ -d "$CONFIG" ] || mkdir -p "$CONFIG"
    [ -d "$DATA" ] || mkdir -p "$DATA"
end script

exec Tautulli.py --nolaunch --config "$CONFIG"/config.ini --datadir "$DATA"
```

Then thats an `init` upstart file. Much shorter, and without all that required text at the top.

But, for now, we use `/etc/init.d` and the commands `sudo service` since its what I know to how to use.

Make `init.d` file Executable

First, we need to make the `init.d` file executable by the system. This basically changes the way the script is called, while still being a bash shell script. Its how the system can use `/etc/init.d` files without having to use `bash` or `sh` in the terminal command - or even ANY file thats technically a script without first calling the scripts program/compiler in the command line.

```
sudo chmod +x ~/git/tautulli/init-scripts/init.ubuntu
```

The `chmod` is modifing the access flags for the file. The `+x` means 'add the executable flag'. There is also `+w` and `+r` which is write and read, in that order. And, there is another way to change that information as well. But thats for another document.

Link or Copy

Then, link the `init.ubuntu` file to `/etc/init.d`. Notice that, in the `/etc/init.d` part, we changed the files name.

```
sudo ln -s ~/git/tautulli/init-scripts/init.ubuntu /etc/init.d/tautulli
```

Update Ubuntu's Autostart Program

And now we add Tautulli to the autostart system that Ubuntu uses, `update-rc.d` being the command to add it. If you want to remove it, you run `sudo update-rc.d tautulli remove` AFTER deleting the `tautulli` file from `/etc/init.d`.

```
sudo update-rc.d tautulli defaults
sudo service tautulli start
```

SystemCTL

If you want to use Ubuntu/Debian's newest startup system, SystemCTL (*systemctl* being the command), follow these instructions.

Note: I am in no way fully learned in how to utilize this newest startup system, nor do I know all of the proper places to store these startup files. But what follows is what has worked for me.

These files do not need to be flagged as executable, as the `init.d` scripts did from above. I believe, in fact, that it possibly will throw an error message if you do so.

If you want to look at some example files, the `/lib/systemd/system` directory is where the files are stored.

Link or Copy

Tautulli's premade `systemd` script is located at `~/git/tautulli/init-scripts/init.systemd`. Of which, there are some items you'll probably want to change, depending on your setup.

With `systemd`, you are able to include using a `defaults` file, like the `init.d` files from above, but I was unable to get that to work properly here.

To link the existing file from Tautlli's repo on your filesystem:

```
sudo ln -s ~/git/tautulli/init-scripts/init.systemd /lib/systemd/system/tautulli.
↪service
```

making sure to add the `.service` to the end of the name.

Update `SystemCTL`'s Autostart Program

And now, adding the service file to `systemctl` system, you use:

```
sudo systemctl daemon-reload
sudo systemctl enable tautulli.service
```

The `systemctl` command has a bunch of command line options. If you want to dive into how to use it, you can `man systemctl` to start you down the rabit hole.

The above command will actually create another symlink from `/lib/systemd/system` to `/etc/systemd/system`, which enables the service script to run.

Accessing

Now, you can access the web interface at http://localhost:8181 with `8181` being the port it is running on.

1.2.4 Sonarr Installation [HTPC-Sonarr]

Sonarr (old name was NzbDrone, which you will still see everywhere. Even in their startup file...) is, honestly, my favorite of all of the HTPC Apps. Even moreso than Plex because Plex can really be a huge resource hog.

The day that Sonarr really hopped into my all time fav basket was when I had began downloading a TV show outside of Sonarr. Then, when I had started adding it to Sonarr, it started searching through my directories I had inputted into

its settings as a possible final download location, found the files, moved them into their correct, final spot, renamed and everything. Without missing a beat.

Its not without its issues, of course. But they often get fixed quickly, or they aren't with this program, but another site or service.

Base Requirements

Make sure apt-transport-https is installed:

```
sudo apt-get install apt-transport-https
```

Add the Apt Source and Install

Luckily, here, they have a repo we can add to our apt-get lists (of which I always have seperate list files for the "outside of ubuntu/debian" standards) so we get to have easy updates and whatnot.

First, gotta add the key, then the repo, and then update and install.

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys FDA5DFFC
echo "deb https://apt.sonarr.tv/ master main" | sudo tee -a /etc/apt/sources.list.d/
↪sonarr.list

sudo apt-get update && sudo apt-get install nzbdrone -y
```

Now, you'll notice there was a TON of apps installing here... Don't panic. This is because Sonarr - aka NzbDrone here - is actually ran using an .exe file! Can you believe it?!? But, there is an open source program called mono that, I believe if the program is compiled for it only, can we run .exe's here! So that was all the Mono modules installing.

But, now we need to run Sonarr, with it outputting the actual output to your shell. This way, you can easily spot any errors, read when it tells you to fix something. Plus, its honestly cool to watch that output live as you click around in a program.

First Run

```
sudo mono /opt/NzbDrone/NzbDrone.exe
```

Now, to stop the program, `ctrl-C` sends the SIGHUP signal to try to gracefully quit the program.

Create Autostart init.d File

Now, we need to make the autostart file, of which, one is not supplied by the installer. I also didn't have much luck with the ones listed on their github site. But the text below was coped from HTPCGuides.com site , [HTPC-Sonarr] which has served me very well.

First, edit the init.d file.

```
sudo nano /etc/init.d/nzbdrone
```

Then, copy and paste the text from below. You'll notice the big *EDIT ME* text in there. Take a looksee, make your edits as you see fit.

```
#! /bin/sh
### BEGIN INIT INFO
# Provides: NzbDrone
# Required-Start: $local_fs $network $remote_fs
# Required-Stop: $local_fs $network $remote_fs
# Should-Start: $NetworkManager
# Should-Stop: $NetworkManager
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: starts instance of NzbDrone
# Description: starts instance of NzbDrone using start-stop-daemon
### END INIT INFO

############### EDIT ME ##################
# path to app
APP_PATH=/opt/NzbDrone

# user
RUN_AS=<Your UserName>

# path to mono bin
DAEMON=$(which mono)

# Path to store PID file
PID_FILE=/var/run/nzbdrone/nzbdrone.pid
PID_PATH=$(dirname $PID_FILE)

# script name
NAME=nzbdrone

# app name
DESC=NzbDrone

# startup args
EXENAME="NzbDrone.exe"
DAEMON_OPTS=" "$EXENAME

############### END EDIT ME ##################

NZBDRONE_PID=`ps auxf | grep NzbDrone.exe | grep -v grep | awk '{print $2}'`

test -x $DAEMON || exit 0

set -e

#Look for PID and create if doesn't exist
if [ ! -d $PID_PATH ]; then
    mkdir -p $PID_PATH
    chown $RUN_AS $PID_PATH
fi

if [ ! -d $DATA_DIR ]; then
    mkdir -p $DATA_DIR
    chown $RUN_AS $DATA_DIR
fi

if [ -e $PID_FILE ]; then
```

(continues on next page)

(continued from previous page)

```
    PID=`cat $PID_FILE`
    if ! kill -0 $PID > /dev/null 2>&1; then
        echo "Removing stale $PID_FILE"
        rm $PID_FILE
    fi
fi

echo $NZBDRONE_PID > $PID_FILE

case "$1" in
    start)
    if [ -z "${NZBDRONE_PID}" ]; then
        echo "Starting $DESC"
        rm -rf $PID_PATH || return 1
        install -d --mode=0755 -o $RUN_AS $PID_PATH || return 1
        start-stop-daemon -d $APP_PATH -c $RUN_AS --start --background --pidfile
↪$PID_FILE --exec $DAEMON -- $DAEMON_OPTS
    else
        echo "NzbDrone already running."
    fi
    ;;
    stop)
        echo "Stopping $DESC"
        echo $NZBDRONE_PID > $PID_FILE
        start-stop-daemon --stop --pidfile $PID_FILE --retry 15
    ;;
    restart|force-reload)
        echo "Restarting $DESC"
        start-stop-daemon --stop --pidfile $PID_FILE --retry 15
        start-stop-daemon -d $APP_PATH -c $RUN_AS --start --background --pidfile $PID_
↪FILE --exec $DAEMON -- $DAEMON_OPTS
    ;;
    status)
        # Use LSB function library if it exists
        if [ -f /lib/lsb/init-functions ]; then
            . /lib/lsb/init-functions
            if [ -e $PID_FILE ]; then
                status_of_proc -p $PID_FILE "$DAEMON" "$NAME" && exit 0 || exit $?
            else
                log_daemon_msg "$NAME is not running"
                exit 3
            fi
        else
            # Use basic functions
            if [ -e $PID_FILE ]; then
                PID=`cat $PID_FILE`
                if kill -0 $PID > /dev/null 2>&1; then
                    echo " * $NAME is running"
                    exit 0
                fi
            else
                echo " * $NAME is not running"
                exit 3
            fi
        fi
    ;;
    *)
```

(continues on next page)

(continued from previous page)

```
        N=/etc/init.d/$NAME
        echo "Usage: $N {start|stop|restart|force-reload|status}" >&2
        exit 1
        ;;
    esac

exit 0
```

Note: See *User Management* for notes on adjusting user permissions with regards to programs and allowing the web access to your machines.

Add to Autostart Program

Then, make the script executable and add to your autostart program. In the below example, its using Ubuntu's `system <program> start` program.

```
sudo chmod +x /etc/init.d/nzbdrone
sudo update-rc.d nzbdrone defaults
```

Final Steps

Then, start the program with `sudo service nzbdrone start` and if you see no error codes - which for some reason Ubuntu 16 wont always show on init.d scripts... Check your HTOP processes to see if its running. If not, `sudo service nzbdrone status` - then the program should be running.

You can see it at http://localhost:8989 if its running on the same machine as your browser. Otherwise put in the machines IP address instead of `localhost`.

1.3 Downloaders

This includes how to install Transmission's headless downloading program and how to install OpenVPN's stuff with direct use of PIAs service and files.

1.3.1 Transmission [TRANS-HTPC]

Transmission is my personal fav of all the Torrent Downloaders. For one, its the easiest I've found to use and setup. For two, it has a headless program for running on a server - and comes standard with a GUI interface that way. And three, its super light weight.

Add Repo and Install

Add the most up-to-date repo for transmission:

```
echo "deb http://ppa.launchpad.net/transmissionbt/ppa/ubuntu $(lsb_release -sc) main"␣
↪| sudo tee -a /etc/apt/sources.list.d/transmission-bt.list
echo "deb-src http://ppa.launchpad.net/transmissionbt/ppa/ubuntu $(lsb_release -sc)␣
↪main" | sudo tee -a /etc/apt/sources.list.d/transmission-bt.list
```

(continues on next page)

(continued from previous page)

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 365C5CA1

sudo apt-get update && sudo apt-get install transmission-daemon transmission-cli -y

sudo service transmission-daemon stop
```

The last line, `stopping the service`, is necessary for you to be able to edit the `settings.conf` file. That file is silently rewritten by the `transmission-daemon` service, if that service is still running.

Note: See *User Management* for notes on adjusting user permissions with regards to programs and allowing the web access to your machines.

User/Group Account Modifications

Add the Transmission group to your regular user account, as well as the group that owns the directory that you will be saving the downloaded files, as this will - hopefully - prevent permission issues.

```
sudo usermod -aG debian-transmission $USER
sudo usermod -aG debian-transmission root
```

Note: If you are running this inside of a VirtualBox Linux Guest, and using VirtualBox's Folder Syncing Feature... You'll notice that the group owner of those mounts is *vboxsf*. Thats only inside of your Guest OS, but will futz with transmission's ability to save anything to those directories. Make sure to also include *vboxsf* with the above commands.

Editing the `settings.conf` File

```
sudo nano /etc/transmission-daemon/settings.json
```

The settings file is quite long with lots of options. The Most Important parts are:

- `"rpc-whitelist": "0.0.0.0",` This blocks off all but only the IP listed from accessing....
- `"rpc-whitelist-enabled": true,` This is the boolean to turn whitelisting on/off. Doesn't always take for some reason.....

```
"rpc-whitelist": "*.*.*.*",
"rpc-whitelist-enabled": false,
```

You can also change the IP address to `192.168.1.0/34` or whatever your home IP address ranges are, depending on what your personal security wants are.

```
"rpc-password": "password",
"rpc-username": "username",
```

Change the download-dir to where ever you want it....

```
"download-dir": "/var/lib/transmission-daemon/downloads",
```

This is the amount of items being downloaded at once. I usually stay at 5 max, no matter what machine I'm using.

```
"download-queue-size": 5,
```

For the seeding queue, I honestly lower it down to 5 as well, since I like to double the upload amount.

```
    "seed-queue-size":  5,
```

Set umask to 002 to avoid permission issues…

```
    "umask":  002,
```

I set the blocklist up as well. [TRANS-BlockList]

```
"blocklist-enabled": true,
"blocklist-url": "http://john.bitsurge.net/public/biglist.p2p.gz",
```

Save the file, and restart the service.

```
sudo service transmission-daemon restart
```

1.3.2 Alternate Transmission Web GUI

The command line tools for Transmission also come with a directory that runs a web-based GUI for Transmission. Not only does this allow running a torrent program on a headless server, but say you have hardware that the graphics card isn't too quick. Rather than install the GUI Desktop version, install the command line version! Plus, if you run a reverse proxy web server in front of it, you can access it from anywhere.

Using Kettu's Web GUI for Transmission

First, clone endor/kettu's custom-built web GUI

```
git clone https://github.com/endor/kettu.git ~/git/kettu
```

Note: I keep all of my cloned git repos inside of one, singular directory:

`~/git`

This way, I don't have to hunt all over my system for where my repo's are and it makes it easier to keep them updated. Then, I symlink the library to wherever either the developer wants/requires it or where is easiest.

The other way of handing this is to clone your PROGRAMS into the /opt directory - so /opt/couchpotato, /opt/NzbDrone, /opt/plexpy and so on. Then clone your working repos for projects into `~/git/[repo]`

Transmission Web GUI File Locations

Next, we need to link the clone repo to the spot that Transmission expects. Since I don't care for the look of the web GUI that comes with the tool, I just delete the entire directory, then symlink the github repo in its place, but naming it `web`.

Note: Transmission's web GUI is located at `/usr/share/transmission/web` in unix systems.

If you're using macOS and their main Transmission App downloaded from their site - the Transmission GUI app, NOT the Command Line tool - the web files are located at `/Applications/Transmission.app/Contents/Resources/web/`, Because even their full-fledged desktop tool comes with a web GUI.

First, before making any changes, its always a good idea to stop transmission before doing ANYTHING to it. It not only will revert any changes made to its `config` file if its still running, but if it does that, who knows how mad it'll get over its web GUI files going missing.

```
sudo service transmission-daemon stop
```

Then, we remove the existing `web` directory, and then symlink-in the replacement.

```
sudo rm -r /usr/share/transmission/web
sudo ln ~/git/kettu /usr/share/transmission/web
sudo service transmission-daemon start
```

Then, of course, confirm that the web interface has changed over. Of which, it usually is running at http://localhost:9091

1.3.3 OpenVPN [PIA-VPN]

Secure Downloading

This specific how-to is in the *Downloaders* section of my how-to's because this is for setting up an OpenVPN connection to a paid-for service, not from your, say, cell phone back home again.

Info

OpenVPN is basically the defacto standard for open source VPN software. You are able to both connect to other VPN servers OR make your own, private VPN service.

Install

```
sudo apt-get install openvpn unzip
```

PIA OpenVPN Files

PIA is Private Internet Access, a widely used and referenced VPN service for the fact that they advertise themselves as one of the more secure and anonymous VPN services on the internet. This page is a good jump-off point for explaining their services and why you need a VPN.

The OpenVPN files are configuration files tailored to be used with OpenVPN for PIA, making that ENTIRE setup SO much easier!

Download and uncompress the PIA OpenVPN profiles:

```
wget https://www.privateinternetaccess.com/openvpn/openvpn.zip
unzip openvpn.zip -d openvpn
```

Make sure you include the `-d` flag, as it'll just uncompress into the current directory, flinging files all over it.

Copy the Files

Copy the PIA OpenVPN certificates and profile to the OpenVPN configuration location.

Note: I'm using *Japan.ovpn* as an example location. You can/should change that to whichever location you want to use.

```
sudo cp openvpn/ca.rsa.2048.crt openvpn/crl.rsa.2048.pem /etc/openvpn/
sudo cp openvpn/Japan.ovpn /etc/openvpn/Japan.conf
```

You'll notice I changed the file extension from `.ovpn` to `.conf`. OpenVPN likes `.conf` files.

Create the Login File

Create `/etc/openvpn/login` containing only your username and password, one per line. Of which, PIA randomly creates your username and password, which is extra-awesome for both security and anonymity

```
user12345678
MyGreatPassword
```

Change the permissions on this file so only the root user can read it, keeping anyone else from snooping.

```
sudo chmod 600 /etc/openvpn/login
```

Edit the Config File

Setup OpenVPN to use your stored username and password by editing the the config file for the VPN location, as well as our `ca` and `crl` files.

```
sudo nano /etc/openvpn/Japan.conf
```

Change the following lines:

From This	To This
ca ca.crt	ca /etc/openvpn/ca.rsa.2048.crt
auth-user-pass	auth-user-pass /etc/openvpn/login
crl-verify crl.pem	crl-verify /etc/openvpn/crl.rsa.2048.pem

Test VPN

At this point you should be able to test that the VPN actually works.

Running it this way outputs the program info, as its running, into the terminal prompt. This way, you see up front without hunting in the logs for if/when/where there is any issues.

```
sudo openvpn --config /etc/openvpn/Japan.conf
```

If all is well, you'll see something like:

```
sudo openvpn --config /etc/openvpn/Japan.conf
Sat Oct 24 12:10:54 2015 OpenVPN 2.3.4 arm-unknown-linux-gnueabihf [SSL (OpenSSL)]␣
↪[LZO] [EPOLL] [PKCS11] [MH] [IPv6] built on Dec  5 2014
Sat Oct 24 12:10:54 2015 library versions: OpenSSL 1.0.1k 8 Jan 2015, LZO 2.08
Sat Oct 24 12:10:54 2015 UDPv4 link local: [undef]
Sat Oct 24 12:10:54 2015 UDPv4 link remote: [AF_INET]123.123.123.123:1194
```

(continues on next page)

(continued from previous page)

```
Sat Oct 24 12:10:54 2015 WARNING: this configuration may cache passwords in memory --␣
↪use the auth-nocache option to prevent this
Sat Oct 24 12:10:56 2015 [Private Internet Access] Peer Connection Initiated with [AF_
↪INET]123.123.123.123:1194
Sat Oct 24 12:10:58 2015 TUN/TAP device tun0 opened
Sat Oct 24 12:10:58 2015 do_ifconfig, tt->ipv6=0, tt->did_ifconfig_ipv6_setup=0
Sat Oct 24 12:10:58 2015 /sbin/ip link set dev tun0 up mtu 1500
Sat Oct 24 12:10:58 2015 /sbin/ip addr add dev tun0 local 10.10.10.6 peer 10.10.10.5
Sat Oct 24 12:10:59 2015 Initialization Sequence Completed
```

With the `Initialization Sequence Completed` being the most important.

Exit this with `ctrl+C`

Setup OpenVPN's Autostart Configuration

Edit the `/etc/default/openvpn` file

```
sudo nano /etc/default/openvpn
```

Next, since I use only the one `.conf` file, I uncomment the `AUTOSTART="all"` line. If you have a different setup, go through and make those changes.

Then, to start the service:

```
sudo service openvpn start
```

Then, to check that my public-facing IP address has changed - since I usually am running this on a headless machine as a downloader - I have an alias assigned in one of my dotfiles. Of which, my breakdown of dotfiles is:

- aliases
- bash_profile
- bashrc
- exports
- functions
- gitconfig
- gitignore
- profile

So, my `.aliases` file is, well, my aliases that I use. So, for checking my public-facing IP address:

```
alias pubip="dig +short myip.opendns.com @resolver1.opendns.com"
```

Add that to either your `.aliases` or `.bash_profile` or `.bashrc` file, whichever you are currenty using, and source the file. It adds the alias entry to the active session.

Then, type `pubIP` and hit enter. You should get your public IP address.

1.3.4 NZBGet [NZBGet-HTPC]

NZBGet is a downloading program specifically for downloading from [Usenet-Servers]. They are inherintely more secure and are, in general, much quicker to download than torrents.

UnRar

First, install UnRAR. This is specifically for handingling what are called `.rar` files. Those mysterious crazy files that you randomly come across on the internets.

Rar files are usually large files, taken apart into seperate items, along with a couple extra, documentation files for how they should be pieced back together again. `unrar` handles these guys along with NZBGet.

```
sudo apt-get install unrar
```

NZBGet Installation

NZBGet Is installed... a bit different than other, apt-based applications.

[NZBGet-GitHub] is nice enough to provide us shell scripts that take care of a lot of installation steps. Though, if you're like me, you might, at LEAST, want to look over what this script does first.

But, if you want the installation command, thats:

```
wget -O - http://nzbget.net/info/nzbget-version-linux.json | \
sed -n "s/^.*stable-download.*: \"\(.*\)\".*/\1/p" | \
wget --no-check-certificate -i - -O nzbget-latest-bin-linux.run
```

Note: Check out *Using \ as New Lines* for some breakdown as above.

Sadly, a lot of times this command doesn't always work out, and its almost never a "one-size-fits-all" scenario on why. The best first step is to look at [NZBGet-GitHub] to see what they say, and if anyone else has had the same issues.

1.4 Web Server Stuff

Here, I will mostly be concentrating on the NGINX web server, as that is what I personally use for m y own setup. Plus, when I began looking around for which server systems to use, NGINX kept standing out more and more, simply for the number of people online writing how-to's for it.

I use NGINX mostly as a reverse proxy server to my downloading programs. This way, rather than having every port for every app forwarded through my router, leaving more and more security holes open to be exploited, but it just makes it easier to remember program names over port numbers when trying to access those sites.

Now, I'm also not going to pretend that I am some kind of expert with NGINX. Moreover, most would probably consider me a half-step above newbie, as I do lack a more fundamental unerstanding of the different options inside an NGINX configuration.

1.4.1 NGINX from PPA

The PPA is Ubuntu's way of trying to make it easier to add extra repo's to the apt-get installer system. It auto-adds auth keys, along with repo addresses. And it adds a means of verifying what programs and whatnot are inside of the repo as well, since you can look up the info on Canonical's Website.

NGINX's PPA

Note: For the `$nginx` variable at the end of the first code line, replace it with either `stable` for their Stable line or `development` for their Mainline. Mainline is what they consider their "beta" line. Stable being their, well, "stable" line.

```
sudo add-apt-repository ppa:nginx/$nginx
sudo apt-get update && sudo apt-get install nginx
```

I will create a NGINX configuration "How-To" at some point in the future.

You can set the server address either inside of `/etc/ansible/hosts` or in a hostfile in the repo. Then, change the name of the host inside of your ansible-playbook as well.

1.4.2 Compiling NGINX From Source [NGINX-Copied]

The way listed here uses `apt-get` to our advantage. By adding the NGINX PPA repo, we get a baseline of required files, configuration files, and other things that make this much easier for us all!

Add NGINX PPA

For the `$nginx` variable at the end of the first code line, replace it with either `stable` for their Stable line or `development` for their Mainline.

Note: Mainline is what they consider their "beta" line. Stable being their, well, "stable" line.

```
sudo add-apt-repository ppa:nginx/$nginx
```

Then, go in and remove the comment from the *deb-src* line inside the *apt/sources.list.d/nginx.list* file. The file most likely will be named something else.

Then update apt-get.

```
sudo apt-get update
```

Download Source Packages

Prerequisites

First, we install the pre-requisites. AKA the package creation tools. This helps in building from source using apt.

```
sudo apt-get install dpkg-dev -y
```

Build Directory

The directory `/opt/rebuildnginx` is simply a potential, central location for all of the building files that we are wanting to use. You can stick it anywhere you want, name it anything you want.

```
sudo mkdir /opt/rebuildnginx
cd /opt/rebuildnginx
```

Source Files Download

Next, running `apt-get source` rather than `apt-get install` installs the source files for the program you've listed.

```
sudo apt-get source nginx
```

Install Build Dependencies

```
sudo apt-get build-dep nginx
```

If the current, main NGINX build doesn't have the specific modules that you are needing, you can add them into a specific file inside the build directory.

The detailed instructions for that specialized need is at ServersForHackers.com

Compile and Install

```
cd /opt/rebuildnginx/nginx-{release}
sudo dpkg-buildpackage -uc -b
```

This will take around a few minutes.

Install NGINX

Once the build is complete, we'll find a bunch of `.deb` files added in `/opt/rebuildnginx`. We can use these to install NGINX.

The `full` package, quite aptly, has the most pre-built modules. So, if thats what you're needing, concentrate on those files.

Next, you'll want to check if you're on 64bit or otherwise. If you're on 64bit, most likely you wanna use `amd64` files. Also, the `dbg` is specifically for debugging tools.

Do you have the file you wanna use? Lets install it then!

```
sudo dpkg --install nginx-full_{ release }+trusty0_amd64.deb
```

Now, you can run `nginx -V` (capital V) and it'll show you the flags and modules and whatnot compiled with NGINX.

Block NGINX from Auto-Update

Next, mark NGINX to be blocked from further apt-get updates, as this potentially will remove the modules you added.

```
sudo dpkg --get-selections | grep nginx
```

and for every nginx component listed run:

```
sudo apt-mark hold {component}
```

And from now on, make sure to watch NGINXs opensource web page for more updates, and perform the same steps again.

1.4.3 NGINX Basic Password Auth

This sets up a basic authorization popup, blocking access to either the entire NGINX server or specific locations, depending on where you put the calling options in your NGINX configuration.

There are a couple of options for the authentication process, using either the simple, htpasswd process or installing/compiling an extra module to tie into another authentication system.

htpasswd

Install the htpasswd Program

First, the program we will be using is inside of the `apache2-utils` item on apt. I know, I know... We are using NGINX to try to stay away from Apache2, or at least thats my general mindset - one day I'll accept Apache, and use NGINX as the proxy in front of it... BUT, the `apache2-utils` only installs what we are needing, and not the general apache server.

```
sudo apt-get install apache2-utils
```

Creating/Adding Usernames/Passwords

Then, you want to call `htpasswd`, which will create a file that will include, per line:

```
username:hashedpassword
```

The passwords in this file are hashed in a specific way through `htpasswd`, so even if someone has access to seeing the contents, can't see the password. So, when you input the password in the popup box, it hashes that password, and compares that info. THATS why a complicated password is IMPORTANT...

So, we will:

```
sudo htpasswd -c /etc/nginx/.htpasswd username
```

The `-c` flag tells it that you are creating a new file, in that specific location and with the name `.htpasswd`

Then, to add more usernames, or alternative CaSeS of usernames (because evidently htpasswd is case-sensitive...) you would call it thusly

```
sudo htpasswd /etc/nginx/.htpasswd UserName
```

Making sure you use the existing filename, so you are adding onto whats already there.

Updating NGINX Configuration

There are two lines you want to add to your nginx configurations:

```
auth_basic "Enter whatever you want here";
auth_basic_user_file /etc/nginx/.htpasswd;
```

1. So, the text after `auth_basic` can be whatever you want, but this will show on the popup for the username/password.
2. Then, the `auth_basic_user_file` is where and which `htpasswd` file you want to be using.

I put these lines in specific `location` blocks, so I can block off certain areas and apps. You can place these lines in the general `server` block, to block off your enter site. And, this pops before any other loading occurs in NGINX. As in, any proxy loads don't occur yet. Only the call to .htpasswd occurs, after entering your username and password.

Auth Pam

There are a series of modules that exist for NGINX, built by either NGINX themselves or by the community at-large, that are all amazing!

This one, ngx_http_auth_pam_module, is one I've been using myself on my own server, and ties in with my instructions on *Compiling NGINX From Source [NGINX-Copied]*.

First Steps

First, you want to clone their git repo locally, either in your traditional `~/git/` location or within the `/tmp` directory you use to build the rest of your NGINX program.

When compiling NGINX, make sure to include an `--add-module=$PATH_TO_MODULE` option, pointing at the git repo's directory - not the `.c` file itself - to compile this into your NGINX build. At which point you proceed through the make and install steps.

Configuration

There are two directives you use with this module:

- `auth_pam`: This is whats called the "Authentication Realm". Like other parts of NGINX (cache, and what not) you can use different so-called "realms" for different things. Essentially, a name goes here.
- `auth_pam_service_name`: This is the pam-specific service name, by default its `nginx`

Examples

For this, I will be liberally copying and pasting from sto's github page:

To protect everything under `/secure` you will add the following to the nginx.conf file:

... code-block:: bash

    location /secure { auth_pam "Secure Zone"; auth_pam_service_name "nginx"; }

Note: The module runs as the web server user, so the PAM modules used must be able to authenticate the users without being root; that means that if you want to use the `pam_unix.so` module to autenticate users you need to let the web server user to read the `/etc/shadow` file, if that does not scare you (on Debian like systems you can add the `www-data` user to the shadow group).

I personally have a separate, so-called "snippets" file that I have this configuration block saved into:

```
# Using Pam Auth
auth_pam "Secure Zone";
auth_pam_service_name "nginx";

allow 10.0.100.0/24;
allow 192.168.1.0/24;
satisfy any;
```

And to use it, I use NGINX's include directive:

```
include /etc/nginx/snippets/basic-auth.conf;
```

inside of any part that I want behind a password, such as:

```
/anysite {
    .. code goes here ..;
    include /etc/nginx/snippets/basic-auth.conf;
}
```

That then puts `/anysite` behind the basic HTTP authentication pop-up, but utilizing the ID's and Passwords saved on the local OS.

1.4.4 Securing NGINX

This is a high level, very basic, more of a "config-only" document. There will be other docs that better explain the items listed here.

One big, generalized note to keep in mind is each line ends with `;`

These following steps are almost singularly taken from Bjornjohansen website. His NGINX Configuration how-to's were the singular guides I used for securing my setup. I cannot recommend his website more!

Basic HTTPS Setup

Obtaining the Certificate

First off, in order to actually have a secure connection to a website, that website has to have a trusted certificate. Its fine and dandy for you to be your own Certificate Authority, or CA, if only you or people who will trust your certs will access your site. If you want anyone else ever to access your site, services or apps, you need to use an already-trusted CA. Of which, you no longer have to pay for this, if you so choose.

Enter CertBot. They are a means of having a free, short term certificate that is SO SO SO much easier to install than almost any other CA I've ever found. Though, the main downside being that its only valid for 3 months (other CA's offer for much much longer), but it being BOTH free and easy is world-shattering in how long it took for this to occur.

The system works, at least in the way I use it, by creating a temporary webserver itself, then trying to access said webserver through the address you've requested a cert for. As in, your web address has to be pointing to the machine running the certbot server at that time.

Then, once it validates that info, it creates the cert, registers it with the Acme CA (their CA repo that they use/created) and instantly, your website's secure certs are trusted by browsers everywhere, due to the Acme CA being a registered and trusted CA.

The options are wide and varied, but the way I use it is:

```
cd ~/
wget https://dl.eff.org/certbot-auto
chmod a+x certbot-auto
sudo nginx -s stop
./certbot-auto certonly
```

Then follow the prompts. Make sure nothing is hogging the ports `80` or `443`, as thats what certbot's server uses to authorize you.

You'll also see letsencrypt floating around, that was the original name, and is the parent projects name.

Take note of the location of the saved certs.

Server Redirect

To default-direct all requests to the secured, 443 items, you can use the following in your `80` server section.

```
server {
  listen 80 default_server;
  listen [::]:80 default_server;
  server_name jpcdi.com www.jpcdi.com;
  return 301 https://$server_name$request_uri;
}
```

The `return 301 https://$server_name$request_uri;` tells the requesting browser or program to use `https`, which then routes to the next server block, which is `https`

adding `http2`

The `http2` is the new standard for the `http` protocol. This is the first update to the protocol in over 15 years, and adds important updates.

From their FAQs:

> **What are the key differences to HTTP/1.x?** At a high level, HTTP/2:
>
> - is binary, instead of textual
> - is fully multiplexed, instead of ordered and blocking
> - can therefore use one connection for parallelism
> - uses header compression to reduce overhead
> - allows servers to "push" responses proactively into client caches

It is, in my opinion, a very important and necessary addition to your NGINX configuration.

On your 443 `listen` directives, add `http2;`, as in:

```
listen 443 default_server ssl http2;
listen [::]: 443 default_server ssl http2;
```

The second line being the ipv6 version of the listen directive

Certificate Locations in NGINX

There are 2 specific items you need to start.

`ssl_certificate` and `ssl_certificate_key`

The `ssl_certificate` is specifically the `fullchain.pem` file that is in the `/etc/letsencrypt/live` directory. The `ssl_certificate_key` is the `privkey.pem` file in the same location as above.

Normally, with almost ANY other CA, there are a million and one steps between start and finish. Between creating accounts, paying them, downloading maybe one file and then sussing out the certificate chains, and all sorts of other things.

Foward Secrecy

Luckily, and most likely the ONLY, NGINX has Forward Secrecy (FS) enabled by default on its connections. So, awesome on less manual steps.

Optimizing NGINX's Secure Connection

Connection Credentials Caching

Almost all of the overhead with SSL/TLS is during the initial connection. So, we setup caching parameters.

```
ssl_session_cache shared:SSL:20m;
ssl_session_timeout 180m;
```

From Bjornjohansen's Optimizing NGINX website: This will create a cache shared between all worker processes. The cache size is specified in bytes (in this example: 20 MB). According to the Nginx documentation, 1MB can store about 4,000 sessions, so for this example, we can store about 80,000 sessions, and we will store them for 180 minutes. If you expect more traffic, increase the cache size accordingly.

I personally use `ssl_session_cache shared:SSL:20m builtin:1000;`. You'll notice the added `builtin` option, defined below, pulled from nginx's website.

builtin a cache built in OpenSSL; used by one worker process only. The cache size is specified in sessions. If size is not given, it is equal to 20480 sessions. Use of the built-in cache can cause memory fragmentation.

SSL Protocols

So, here is where we disable SSL. (?)

The reason for this: SSL is old news, with TLS being what has replaced SSL. But, SSL is still a bit in the common vernacular to this day. SSL also has been broken through several weaknesses and is easily bypassed through protocol downgrade attacks.

The only browser left to not support TLS is IE6. Which, I personally believe if someone is still using that browser, they don't really need my site anyways.

We add::

```
ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
```

to cite the specific secure protocols we use on our site.

Optimizing the Cipher Suites

The cipher suites are how the data is encrypted. We list which suites we will use with the browsers.

I personally use::

```
ssl_prefer_server_ciphers on;
ssl_ciphers ECDH+AESGCM:ECDH+AES256:ECDH+AES128:DH+3DES:!ADH:!AECDH:!MD5
```

The `ssl_prefer_server_ciphers on;` is to tell the client we have a preferred order of cipher suites. Then, the list of the suites. Take a look at SSL Labs Deployment Best Practices for detailed info on the suites.

For the long list they present, you can make a seperate file, save it inside your `/etc/nginx` directory, and reference it in your nginx configuration `include cipher_suites`, and it will use the contents of that file.

Generate DH Parameters

Create the DH Parameters file with 2048 bit long safe prime:

```
sudo mkdir /etc/nginx/cert
sudo openssl dhparam 2048 -out /etc/nginx/cert/dhparam.pem
```

And add it to your config with

```
ssl_dhparam /etc/nginx/cert/dhparam.pem;
```

Enable OCSP Stapling

Note: WARNING! If you are currently trying to do any type of setup or development work internally that will be utilizing some form of your public-facing URL, like `home.your.url.com` where `home` is the internal piece, this step will throw a HUGE WRENCH into this process!! Essentially, it'll force you to have an SSL cert on even a newly-installed application or program, because ALL web browsers will be expecting one for however long you put down in your configuration below. YOU'VE BEEN WARNED!!

Online Certificate Status Protocol (OCSP) is a protocol for checking the revocation status of the presented certificate. When a proper browser is presented a certificate, it will contact the issuer of that certificate to check that it hasn't been revoked. This, of course, adds overhead to the connection initialization and also presents a privacy issue involving a 3rd party. Thus, the reason for OCSP Stapling:

The web server can, at regular intervals, contact the certificate authority's OCSP server to get a signed response and staple it on to the handshake when the connection is set up. This provides for a much more efficient connection initialization and keeps the 3rd party out of the way.

To make sure the response from the CA is not tampered with, we also set up Nginx to verify the response using the trusted certificate provided by CertBot, which is the `chain.pem` file provided.

So, for the NGINX configuration:

```
ssl_stapling on;
ssl_stapling_verify on;
ssl_trusted_certificate /etc/letsencrypt/live/jpcdi.com/chain.pem;
resolver 8.8.8.8 8.8.4.4;
```

The resolver uses whatever DNS server you specify, so NGINX can find the resolver through the internet.If you want to use another public DNS, use them.

Strict Transport Security

This setting tells your browser, after its attempted an unsecure connection once, will default to the secure connections only within the cached timeframe you have listed.

```
add_header Strict-Transport-Security 'max-age=31536000; includeSubDomains; preload';
```

The preload is if you want to add your server to the Google Maintained list of sites that are for sure secure.

The includeSubDomains, obviously is to include all subdomains. And I have this in my `http` block, above the `server` block.

Configuration Example

Here is the tl;dr configuration, with the above in one place, plus more lines from my personal config file:

```
http {

  # beginning of your config file

  add_header Strict-Transport-Security 'max-age=31536000; includeSubDomains; preload';
  add_header X-Content-Type-Options "nosniff" always;
  add_header X-Frame-Options "SAMEORIGIN" always;
  add_header X-XSS-Protection "1; mode=block";
  add_header X-Robots-Tag none;
  add_header X-Download-Options noopen;
  add_header X-Permitted-Cross-Domain-Policies none;

  # Giant sea of SSL stuff...
  ssl_session_cache shared:SSL:20m builtin:1000;
  ssl_session_timeout 180m;
  ssl_protocols TLSv1 TLSv1.1 TLSv1.2;

  # Using list of ciphers from https://bjornjohansen.no/optimizing-https-nginx
  ssl_prefer_server_ciphers on;
  ssl_ciphers ECDH+AESGCM:ECDH+AES256:ECDH+AES128:DH+3DES:!ADH:!AECDH:!MD5;

  # rest of http block
}
server {
  listen 80 default_server;
  listen [::]:80 default_server;
  server_name jpcdi.com www.jpcdi.com;
  return 301 https://$server_name$request_uri;
}
server {
  listen 443 default_server ssl http2;
  listen [::]:443 default_server ssl http2;

  # Letsencrypt ssl_certs and one dhparam.pem
  ssl_certificate /etc/letsencrypt/live/jpcdi.com/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/jpcdi.com/privkey.pem;
```

(continues on next page)

(continued from previous page)

```
  ssl_dhparam /etc/nginx/cert/jpcdi_dhparam.pem;

  # HSTS/SSL Stapling stuff...
  ssl_stapling on;
  ssl_stapling_verify on;
  resolver 8.8.8.8 8.8.4.4 valid=86400;
  ssl_trusted_certificate /etc/letsencrypt/live/jpcdi.com/chain.pem;
}
```

1.4.5 DDClient [DDCLIENT-Source]

Preface

This guide is a bit more hands on, but is not only a good lesson to get through and learn, but ends up building a nice, very useful and light weight program!

DDClient, specifically, is a Dynamic DNS program. Through its config files, it listens for public IP address changes on the machine its running on - IPv4 specifically - and then relays that information back to your Domain Name Host - Google Domains for example - so that, if you are self-hosting stuff attached to your url name, and its hosting from an IP address that likes to change - which consumer-based internet access will change that IP address - this handles that issue.

Plus, Google Domains also has a specific, easier-to-use config setup for DDClient, so it makes your life just that much easier.

Software Requirements

Most of what is needed usually comes with Ubuntu, like Perl and CPAN.

But, one program that is needed is `libio-socket-ssl-perl`, specifically to assist with SSL encryptions of the ddclient traffic.

```
sudo apt-get install libio-socket-ssl-perl
```

Clone the Github Repo

Lets start off by cloning the github repo, so you have the most up-to-date code to build with.

Note: I keep all of my cloned git repos inside of one, singular directory:

`~/git`

This way, I don't have to hunt all over my system for where my repo's are and it makes it easier to keep them updated. Then, I symlink the library to wherever either the developer wants/requires it or where is easiest.

The other way of handing this is to clone your PROGRAMS into the /opt directory - so /opt/couchpotato, /opt/NzbDrone, /opt/plexpy and so on. Then clone your working repos for projects into `~/git/[repo]`

```
git clone https://github.com/ddclient/ddclient.git ~/git/ddclient
```

Or, like with couchpotato, you are welcome to save the git directory where ever works best for you.

Copying Files and Creating Directories

Next, its a matter of sticking the ddclient program into the right place, creating its directories, backing up the existing configs and all that jazz.

```
cd ~/git/ddclient/
sudo cp ddclient /usr/sbin/
sudo mkdir /etc/ddclient/
sudo mkdir /var/cache/ddclient
```

1. So the first `cp` is the actual program itself, thus why its going inside of `/usr/sbin/`.
2. Then, creating `/etc/ddclient` because thats where it expects its config file.
3. Then, `/var/cache/ddclient` in order to save the programs caching files.

The `ddclient.conf` File

If you already have a `ddclient.conf` file made - like Google Domains, since their support files explain exactly how to make a config file just for their service - copy that over to::

```
/etc/ddclient/ddclient.conf
```

But, if you don't have a starting point yet, if you look in the DDClient repo files, there are several sample files of all creeds and uses. This is a good chance for you to poke around a bit and see what each one is meant for and look through your specific config file, since it will have a billion and one explanations for everything. If, you're interested.

Otherwise, most likely the service you have your URL hosted with will have a how-to specific for their service on configuring DDClient.

Then, once you have a good config file you like, save it into a github repo of your own so you have it saved, sourced, and backed up to a cloud server if anything goes awry.

Autostart

Since we're on Ubuntu, copy the `init.d` file over and set it up to **always** autostart:

```
sudo cp sample-etc_rc.d_init.d_ddclient.ubuntu /etc/init.d/ddclient
sudo chmod +x /etc/init.d/ddclient
sudo update-rc.d ddclient defaults
```

So, we're copying a sample file - specifically the `init.d_ddclient.ubuntu` file. Obviously if you wanted to run DDClient in another fashion - like with the cron file or with the specific wrapper `.sh` script - you could if you were so inclined.

Then, the `chmod +x` makes the file executable by the sytem, and `update-rc.d` adds it to upstart for autostarting.

Perl/CPAN

Now, DDClient is dependent on some Perl Libraries to help it run. So, the specific Perl Library Management tool we'll be using is CPAN.

And, I believe if I am correct, Perl is a tool that comes with Ubuntu Server, along with CPAN. But, if your system isn't allowing the `cpan` command to run, you can do a `apt-cache search cpan` and it'll show the various programs that might include it.

If Perl completely isn't installed, then:

```
sudo apt-get install perl
```

Then, once thats done, you'll want to install the library DDClient requires::

```
sudo cpan install Data::Validate::IP
```

Since I don't know hardly anything about Perl other than I need it, I simply press enter through the various questions CPAN asks, since it will autofill with the defaults.

Running DDClient

Once you have all that taken care of, you can run `sudo ddclient` and it'll run it once, it'll talk to the server you have configured, and respond with output. For me, it usually tells me that my IP address is unchanged, and to run DDClient unneccessarily is considered abusive.

Which is why we use Upstart, systemctl or the wrapper script or cron job!

Now, since we added this to Upstart, type `sudo service ddclient start` and that script will take over the management!

1.4.6 Certbot WildCard SSL Cert

This how-to is specific for grabbing a wildcard certificate for your domain using Certbot and Letsencrypt, with Certbot's dns system. Specifically using their manual system.

It is fairly difficult to find an all-in-one how-to for getting a wildcard cert manually using Certbot.

Installation

Depending on the route you'd like to take, there's plenty of ways to install Certbot for use. Even the installer script listed on their site is self-updating and, obviously, the most up-to-date way to use it.

I, currently, am using the `Stretch-Backports` installation route rather than downloading the binary directly. Installation through the `apt` system sets up a few behind-the-scenes steps for automating your renewal.

Debian

```
sudo apt-get install python3-certbot-nginx
```

If you are using a DNS provider that has a wonderful API, like Cloudflare (which I use), you can choose to install the plugin for that API as well!

```
sudo apt-get install python3-certbot-dns-cloudflare
```

This will install Certbot, along with the `nginx` plugins through python, along with the manual DNS flag that we're planning on using here.

Setup

Most everything online will give you stupidly long command lines with a TON of flags and what not. I'd much rather use a nice configuration file!

```
# This `cli.ini` is for manually getting a wildcard cert
manual
preferred-challenges = dns
cert-name = jpcdi.com
rsa-key-size = 4096
email = email@address.com
logs-dir = /etc/letsencrypt/logs/
keep-until-expiring
expand
# Use the ACME v2 staging URI for testing things
#server = https://acme-staging-v02.api.letsencrypt.org/directory
# Production ACME v2 API endpoint
server = https://acme-v02.api.letsencrypt.org/directory
domains = *.jpcdi.com, jpcdi.com
```

So, to break all that down:

Break Down

1. `manual` - means to run with the `--manual` flag that you will see in a ton of commands online
2. `preferred-challenges` - This is the challenge you're wanting certbot to use, which for this how to is `dns`
3. `cert-name` - This is specifically referencing the SSL Certs name for the Directories themselves. If you don't pick a name here, you'll end up with A TON of directories, with a ton of numbers and names. (Unless you want that, of course. This helps with automation)
4. `rsa-key-size` - is just what it says, how big you want your key to be. `2048` or `4096`, with `2048` being the default.
5. `email` - this is an administrative need, to stop the program from asking interactively to give them your email and what not
6. `logs-dir` - your logs directory
7. `keep-until-expiring` - Makes sure to not create a bunch of duplicate certs
8. `expand` - if a prior certificate already existed with one of your requested addresses, rather than overwriting that cert, it expands it with the new names you've requested to add.
9. `server` - This one is more specific to our DNS address requesting, as a way to make sure we are getting a wildcard certificate. They are only available through the V2 Acme Protocol, so you have to specify the address to use. I've included the testing address as well, if you want to run certbot a bunch of times and not let it count against your weekly alotment.
10. `domains` - This is where you include the domain addresses you want to use. Make sure you don't JUST include the wildcard, but the root of the domain as well.

Running Certbot

When you use the manual DNS system, it will give you an address and TXT record that you'll need to add to your main DNS Server's addressing. It'll be a good idea to make sure you're logged into their website, or have the configuration pulled up to where you can make the addition and have it propogate out.

Luckily, the system is good and patient, willing to wait for you to take a while to get that TXT record setup.

1.5 Extra Items

These are Ubuntu-Specific items, apply to - usually - all of my VM's, no matter what they are being used for.

AKA:

1. *LVM How To* for using LVM for your drives
2. *Using Postfix as Mail Relay* for the system to be able to send emails;
3. *Mounting Unformatted Drive in Ubuntu*
4. *Renaming Network Devices*

1.5.1 LVM How To

This how to, for now - like all the others - is more for my remembering how to do something, and as I learn more about `LVM`, I'll try to add those nuggets here.

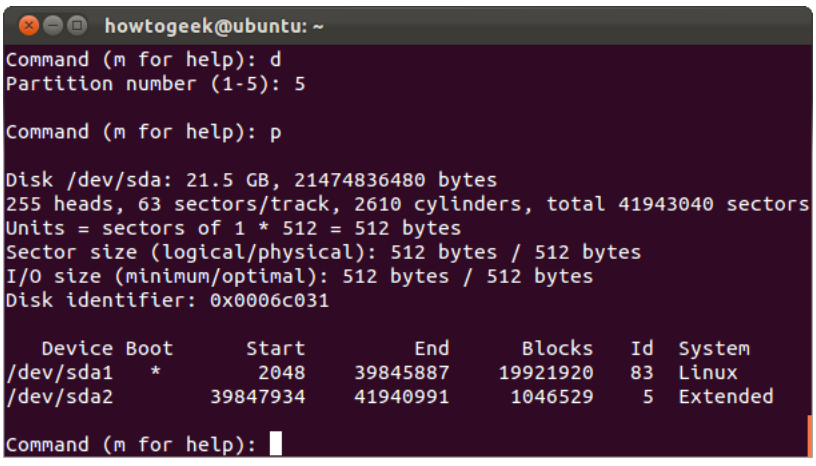
Old Hard Drive?

Do you have an old drive that you need to erase and reformat? There are several ways to erase a disk on debian-based systems, but we'll use good old **`fdisk`** for this one.

Like above, you wanna run `sudo lsblk` or `sudo fdisk -l` to find where your drive is located. Then, use that designation for the next commands:

```
sudo fdisk /dev/<drive_label>
```

Next, you'll want to use `d` to delete all of the partitions, using the command for as many partitions are on the drive.



Then, you can use the `p` command to print the current table.

Cleaned Hard Drive

These steps are for a fresh hard drive, either brand new or freshly erased, and we'll be using **`fdisk`** once again.

```
sudo fdisk -l
```

This command lists the hard drives that the OS can find, whether actually mounted or not. And, it outputs a LOT of information. An alternative command you can use, that I usually prefer:

```
sudo lsblk
```

This one is really great for if you already have drives mounted somewhere, so you can keep your bearings on which drives are what. Example output below:

```
NAME                 MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda                    8:0    0   1.8T  0 disk
└─sda1                 8:1    0   1.8T  0 part /media/server1
sdb                    8:16   0   1.8T  0 disk
└─sdb1                 8:17   0   1.8T  0 part /media/server2
sdc                    8:32   0 465.8G  0 disk
├─sdc1                 8:33   0   456G  0 part /
└─sdc2                 8:34   0   9.8G  0 part [SWAP]
sdd                    8:48   0   1.8T  0 disk
└─sdd1                 8:49   0   1.8T  0 part
  └─mdusa-externals 252:0    0   2.7T  0 lvm  /media/mdusa
sde                    8:64   0 931.5G  0 disk
└─sde1                 8:65   0 931.5G  0 part
  └─mdusa-externals 252:0    0   2.7T  0 lvm  /media/mdusa
sr0                   11:0    1  1024M  0 rom
```

Next, you'll use the ID of the drive in the command:

```
sudo fdisk /dev/<disk_id>
```

This starts **`fdisk`** with your disk selected. If you want to see all of the commands, `m` is the help option.

We are wanting to add a new partition, so type `n` and press enter, and then select `p` for `primary`. Next, it'll ask for the sector locations of where you want the partitions to exist. You'll notice that the program gives you a default selection to choose from. For the sector locations, you can choose the default options that **`fdisk`** provides, pressing `enter` to keep going.

Next, you'll use the `t` option, which changes the partition type/id. In here, there is a super long list of options, and how you select the id can change from version to version, so you'll need to list the options.

```
Command (m for help): t
Selected partition 1
Partition type (type L to list all types): l

0  Empty           24  NEC DOS         81  Minix / old Lin bf  Solaris
1  FAT12           27  Hidden NTFS Win 82  Linux swap / So c1  DRDOS/sec (FAT-
2  XENIX root      39  Plan 9          83  Linux           c4  DRDOS/sec (FAT-
3  XENIX usr       3c  PartitionMagic  84  OS/2 hidden or  c6  DRDOS/sec (FAT-
4  FAT16 <32M      40  Venix 80286     85  Linux extended  c7  Syrinx
5  Extended        41  PPC PReP Boot   86  NTFS volume set da  Non-FS data
6  FAT16           42  SFS             87  NTFS volume set db  CP/M / CTOS / .
7  HPFS/NTFS/exFAT 4d  QNX4.x          88  Linux plaintext de  Dell Utility
8  AIX             4e  QNX4.x 2nd part 8e  Linux LVM       df  BootIt
9  AIX bootable    4f  QNX4.x 3rd part 93  Amoeba          e1  DOS access
a  OS/2 Boot Manag 50  OnTrack DM      94  Amoeba BBT      e3  DOS R/O
b  W95 FAT32       51  OnTrack DM6 Aux 9f  BSD/OS          e4  SpeedStor
c  W95 FAT32 (LBA) 52  CP/M            a0  IBM Thinkpad hi ea  Rufus alignment
```

(continues on next page)

(continued from previous page)

```
e  W95 FAT16 (LBA) 53  OnTrack DM6 Aux a5  FreeBSD         eb  BeOS fs
f  W95 Ext'd (LBA) 54  OnTrackDM6      a6  OpenBSD         ee  GPT
10  OPUS            55  EZ-Drive        a7  NeXTSTEP        ef  EFI (FAT-12/16/
11  Hidden FAT12    56  Golden Bow      a8  Darwin UFS      f0  Linux/PA-RISC b
12  Compaq diagnost 5c  Priam Edisk     a9  NetBSD          f1  SpeedStor
14  Hidden FAT16 <3 61  SpeedStor       ab  Darwin boot     f4  SpeedStor
16  Hidden FAT16    63  GNU HURD or Sys af  HFS / HFS+      f2  DOS secondary
17  Hidden HPFS/NTF 64  Novell Netware  b7  BSDI fs         fb  VMware VMFS
18  AST SmartSleep  65  Novell Netware  b8  BSDI swap       fc  VMware VMKCORE
1b  Hidden W95 FAT3 70  DiskSecure Mult bb  Boot Wizard hid fd  Linux raid auto
1c  Hidden W95 FAT3 75  PC/IX           bc  Acronis FAT32 L fe  LANstep
1e  Hidden W95 FAT1 80  Old Minix       be  Solaris boot    ff  BBT
Partition type (type L to list all types):
```

The id you're wanting to pick from here is `Linux LVM`, which with the above options is `8e`.

And, finally, use `w` to write all of these changes to the drive, and then **`fdisk`** exits you out.

LVM Commands

Now, we start using the actual commands for `LVM`.

LVM Physical Volume

First, before we make the `Volume Group`, we need to finish working on the freshly wiped hard drive. You'll need to run **`pvcreate`** to finish that off.

```
sudo pvcreate /dev/<disk_id>
```

It'll most likely throw a warning saying that an `existing ext4 signature was detected. Are you sure you want to continue?` Enter `y` to confirm, and it formally formats it correctly for you to be able to use it in an LVM Volume Group.

Volume Group

We need to create the `Volume Group` first:

```
sudo vgcreate <pool_name> /dev/<disk_label>
```

Replace the `<pool_name>` with the name you want to use. I like to use the computer's host name as the volume name, as I usually only have one volume on my systems.

Logical Volume

Next, we create the logical volume that LVM will use. This is the individual volumes within the group, like partitions on a hard drive.

```
sudo lvcreate -l 100%FREE -n <volume_name> <pool_name>
```

Lets break this down a bit:

1. `-l` : This option is for selecting the size of the volume. There are several different options not only within the `-l` flag, but there is also a `-L` flag for using a specific size, like `3G` for 3 GBs. To get a handle on this info, its best to start looking at **`lvcreate`** s **`man`** page. The `100%FREE` option here is telling the program to use all available free space.
2. `-n` : this is for saving the volume's name.
3. Then, you finish it off with the `<pool_name>` from earlier.

Filesystem

And, finally, we have to format an actual file system inside the volume.

```
sudo mkfs -t <filesystem> /dev/<pool_name>/<volume_name>
```

Here, the location of the Volume Group and Logical Volume are within the `/dev` directory. But, the first time you run this command, the normal bash-completion might not yet have this location ready for you, so you'll need to type out the entire location.

Mount Point

Now, create the mount point and mount the volume!

```
sudo mkdir <your_mount_point>
sudo mount /dev/<pool_name>/<volume_name> <your_mount_point>
```

So, from now on, you are able to reference just the `volume_name` for however many hard drives you place within your volume.

1.5.2 Mounting Unformatted Drive in Ubuntu

Here, we will be mounting a new drive in Ubuntu. The beginning items will be for adding a partition, and then formatting it to an EXT4 filesystem.

Personally, the reason I needed this was that I had created a 2nd VirtualDisk in my VirtualBox Host, and needed to figure out how to format it and then mount it to my Ubuntu 16.04 Guest.

Install/Create the Drive

I'm not going to really detail this part, because if you purchased a new, internal drive - or even an external, USB drive - it either came with directions or you know how to install that.

As for the VirtualDisk for your VM of choice, that usually is fairly easy. At least with VirtualBox it is.

Finding the Drive

So, obviously, we need to find the system-assigned ID for the drive, as this is the info we use for formatting, mounting, etc.

So, we will be using `fdisk` for these next few steps:

```
sudo fdisk -l
```

Note: In my Ubuntu instance inside of VirtualBox, not only did it show my Drives, but `/dev/ram` drives as well. I'm assuming this is just one form of memory and disk I/O management VirtualBox uses.

So, my 2nd hard drive is labeled as `/dev/sdb`, and because I had just created the virtual drive on my host machine, there are no partitions.

```
Disk /dev/sda: 15 GiB, 16106127360 bytes, 31457280 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x000457d0

Device     Boot    Start      End  Sectors Size Id Type
/dev/sda1  *        2048 27262975 27260928  13G 83 Linux
/dev/sda2       27265022 31455231  4190210   2G  5 Extended
/dev/sda5       27265024 31455231  4190208   2G 82 Linux swap / Sol


Disk /dev/sdb: 400 GiB, 429475428352 bytes, 838819196 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

Create Partitions

Now we want to create the partitions for the drive. For me, since I only wanted one large partition, this is easy with fdisk. If you're wanting/needing a more complicated setup, it might be best to lookup `parted` and `gparted` (graphical version of parted) for better partitioning.

So, now, we want to run

```
sudo fdisk /dev/sdb
```

Or, if you're drive's name is different, replace the /dev/sdb with whatever yours is.

You now will be dropped into the `fdisk` shell prompt. To start the creation of a new partition, press `n` and enter to start.

Next, since we're doing one, whole-drive partition, select `p` for `primary partition`. Then, you can just hit `enter` through `Partition Number`, `First sector` and `Last sector` as those have defaults pre-filled for you, if you would rather use those.

This creates, in memory - WITHOUT formally writing to the disk - a partition for the drive. You have to hit `w` in order to formally write the information onto the disk. But, before you do that, I suggest now using option `p` for `print the partition table` so you can confirm the information before fatally committing to it.

Write Partitions

Now, once you KNOW for SURE you're ready, use the `w` option to write to disk. And then, depending on your version of `fdisk`, it'll either drop you back into the normal shell immeditely, or you'll have to `q` - for quit - to quit out of fdisk.

Partprobe

Next, you want to run `partprobe` in order to tell the kernel, without restarting your machine, about the changes made to that harddrive.

```
sudo partprobe /dev/sdb
```

Or, whatever your devices name is.

Format

Now, we need to format our drive so the filesystem is readable/writable and what nots.

```
sudo mkfs /dev/sdb1 -t ext4
```

So, that is `mkfs` or makefilesystem, then our drive with the partition number. The `-t` flag is for type of filesystem, and `ext4` for, well, ext4 filesystem.

If you are wanting/needing a different system, please consult those specific formatting directions.

Mount Point

Now, you need to create a directory that you want this disk mounted to, as that is how you are able to access this drive.

I personally tend to mount all my drives inside of `/media`. No real reason for it. Just started that way and became habit. You can mount this drive anywhere you want. But, beside `/media`, there is also the `/mnt` directory, which is much more of a literal "duh, stick it here" kind of naming that linux loves.

So, make the directory, adjust the ownership info, then mount the ext4 filesystem drive:

```
sudo mkdir /media/<name of folder>
sudo chmod -R 777 /media/<name of folder>
sudo mount /dev/sdb1 /media/<name of folder> -t ext4
```

So, lets break that down:

1. `chmod` - changing the ownership levels to the mode `0777` which translates to: anyone can do anything with this.
2. `mount` - obviously the mount program

1. The first option has to be the device with partition number you want to mount
2. The second option being the directory to mount ONTO
3. `-t ext4` being "type ext4"

After Mount

Now, if you didn't receive any error messages, you can basically, with 99% confidence, say its mounted. But, doesn't hurt to `cd` into it and create a blank file real quick to test the read/write.

```
cd /media/<name of folder>
touch ./test.txt
ls -lah
sudo rm ./test.txt
```

So, if those all worked, its mounted and you can read/write to it.

Permanent Mounting

The best way to be able to mount your drives using `/etc/fdisk` is by referencing the drive's UUID number. How do you check that?

```
sudo blkid
```

it will give you a line for every drive and partition that it can find. You can use either the `UUID` or the `PARTUUID` in your `fstab` file.

```
PARTUUID=86e32033-01  /boot           vfat    defaults          0       2
PARTUUID=86e32033-02  /              ext4    defaults,noatime  0       1
```

1.5.3 Using Postfix as Mail Relay

So, postfix is a fairly full-featured, backend mailserver software, that, to be honest, the reason I'm using it is because the most recent, up to date how-to's all use postfix. Which, I'm assuming that means its probably one of the easier mail systems to configure.

But, we will be using it to forward all of the system emails to our personal email address. I use gmail, so my examples will be more geared towards gmails smtp address and port.

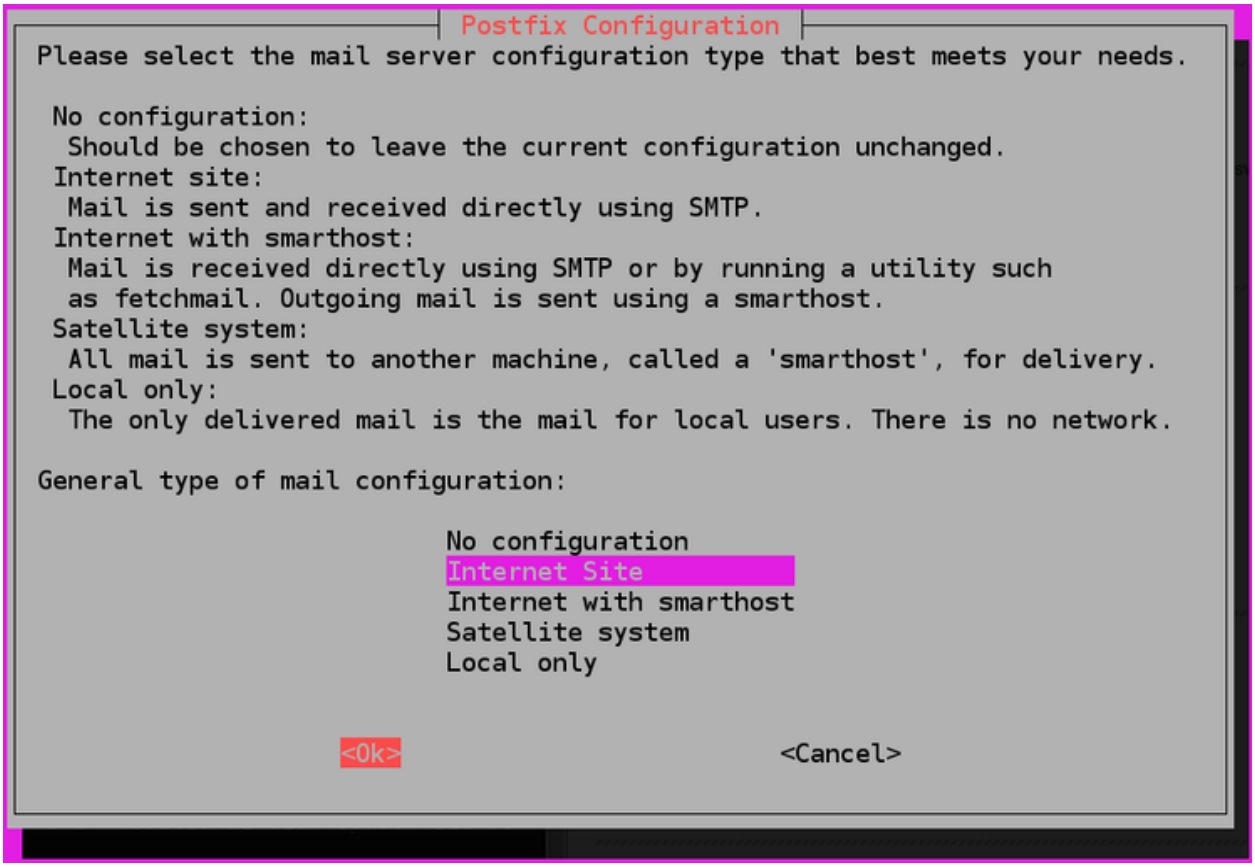
This was lifted - quite almost literally - from HowToForge. [POSTFIX-HowTo]

Install Postfix

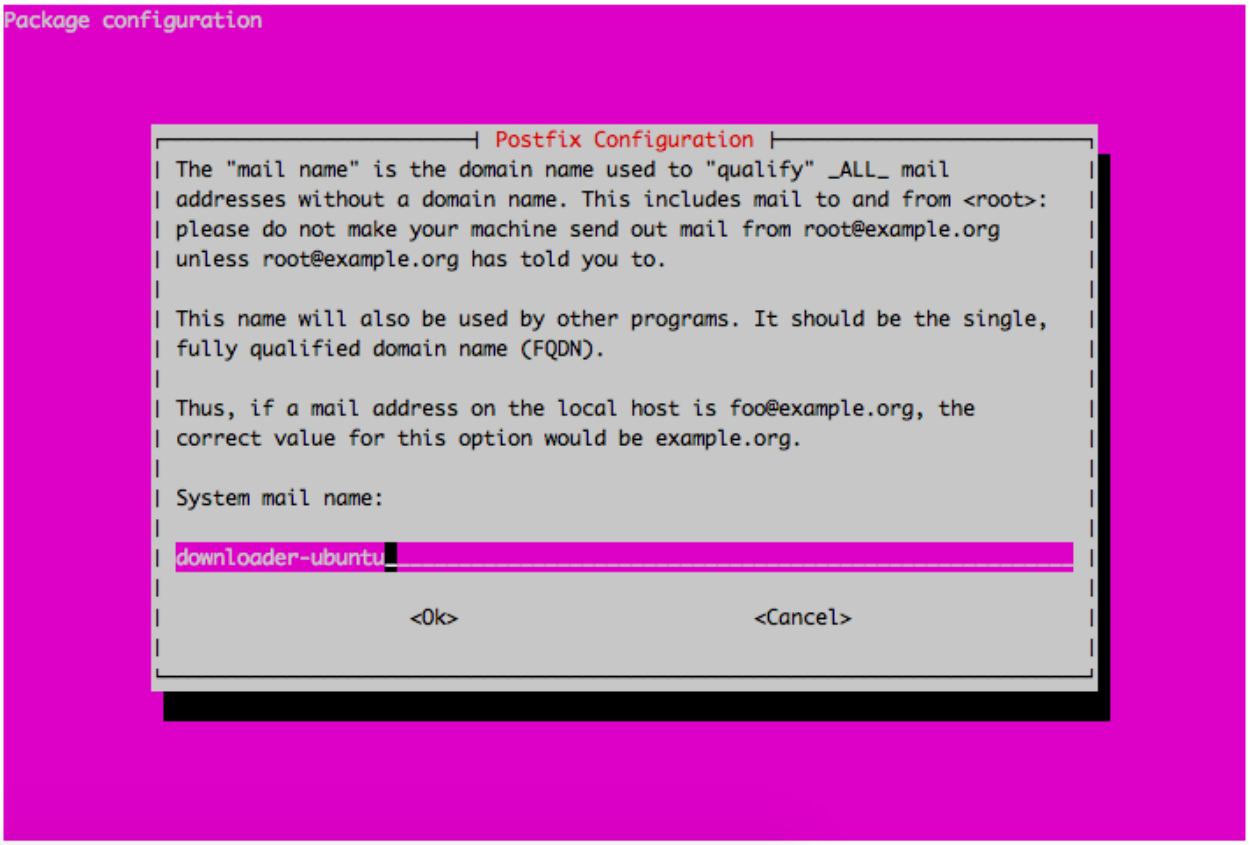
```
sudo apt-get install postfix mailutils
```

Now, during this installation, the system will prompt you with Configuration Option's. Since we will be using an outside service to send our mail - aka smtp.gmail.com - we will select `Internet Site`.

If we were to use postfix in other ways, we'd pick another option.



Then, it will continue on with `System Mail Name`, which, technically you would normally want a `FQDN` address listed here. But, using your systems basic hostname is also fine, especially if you have just a couple of machines.



Configure Postfix

We will be setting the system to process emails only from "the server on which it is running," aka the `localhost` or `loopback interface`. That way, when postfix "receives" an email from the system - for say, root - it will use Postfix to forward the email off through our specified smtp server. Thus, using the loopback as the "catch-all" for the emails.

Generic File

You'll want to create a `generic` file inside of the postfix configuration directory. This should contain your systems hostname and then your email address, aka:

```
ubuntu-server   admin@jpcdi.com
```

Password File

First, we're going to make a seperate, locked down password file that Postfix will use to authenticate with gmail.

```
sudo nano /etc/postfix/sasl_passwd
```

And add the line:

```
smtp.gmail.com:587  username@gmail.com:password
```

Which, of course, if you use a different mail service, input their info and it should work just the same. And, also, `username@gmail.com:password` needs to be replaced with your info.

Now, lock that file down so only root can view it.

```
sudo chmod 600 /etc/postfix/sasl_passwd
sudo chown root:root /etc/postfix/sasl_passwd
```

Process Password and Generic File

Remember when you installed `mailutils`? That was for `postmap`, which compiles and hashes the contents of our `sasl_passwd` and `generic` files, and creates a new file in the same spot, with `.db` added to the end, making it a database file easier to parse as it runs.

```
sudo postmap /etc/postfix/sasl_passwd
sudo postmap /etc/postfix/generic
```

Main Configure File

In the `main.cf` file, there are 6 specific parameters we will be using for the relay setup:

1. `relayhost` which specifies the mail relay host and port number. The host name will be enclosed in brackets to specify that no MX lookup is required.
2. `smtp_use_tls` which enables (or disables) transport layer security.
3. `smtp_sasl_auth_enable` which enables (or disables) SASL authentication.
4. `smtp_sasl_security_options` which in the following configuration will be set to empty, to ensure that no Gmail-incompatible security options are used.
5. `smtp_sasl_password_maps` which specifies the password file to use. This file will be compiled and hashed by postmap in a later step.
6. `smtp_tls_CAfile` which specifies the list of certificate authorities to use when verifying server identity.
7. `smtp_generic_maps` tells postfix to read your system name and email address from your generic file

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

The `main.cf` is postfix's config file.

You will most likely have to add most of the above options, possibly deleting one or two in order to clump them all together in one, single block of text.

```
relayhost = smtp.gmail.com:587
smtp_use_tls = yes
smtp_sasl_auth_enable = yes
smtp_sasl_security_options =
smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd
smtp_tls_CAfile = /etc/ssl/certs/ca-certificates.crt
smtp_generic_maps = hash:/etc/postfix/generic
```

The `smtp_sasl_security_options` is left empty.

Restart Postfix

Restart postfix, enabling our various changes:

```
sudo systemctl restart postfix.service
```

– or –

```
sudo service postfix restart
```

Send Test Emails

This is testing if the actual forwarding part works, as you're able to send emails through the command line.

To send a test email over the command line:

```
echo "This is the body of the email" | mail -s "This is the subject line"␣
↪user@example.com
```

Making sure to put your email address in place of `user@example.com`. You should receive the email within a few seconds if its successful.

1.5.4 Renaming Network Devices

So, as of Ubuntu 16.04, or rather 15 something, they changed up their naming schemes for the network devices, coming up with some crazy names.

```
en3p0b or something.....
```

Not easy like the old school `eth0` was...

Well, I found the specific info from AskUbuntu specifying how to change this!

How To

First, get your devices MAC address, by running `ip link`

The output will show something similar to:

```
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DORMANT␣
↪group default qlen 1000
      link/ether 20:68:9d:xx:xx:xx brd ff:ff:ff:ff:ff:ff
```

The important info is after `link/ether`, with 5 `:` between 6 sets of 2 numbers. Thats the devices MAC address.

```
20:68:9d:xx:xx:xx
```

Next, we need to create a new file, `/etc/udev/rules.d/10-network.rules`.

Note: Obviously, if you are already using other `.rules` inside of `/etc/udev/rules.d/`, you know - hopefully - what naming schemes are needed or know where to look. This info is beyond this current page.

So, use your fav text editor:

```
sudo nano /etc/udev/rules.d/10-network.rules
```

Add the following line, replacing `aa:bb:cc:dd:ee:ff` after `ATTR{address}` with your devices MAC address and `etho` after `NAME` with the name that YOU want to use, in lieu of what the devices name is currently.

```
SUBSYSTEM=="net", ACTION=="add", ATTR{address}=="aa:bb:cc:dd:ee:ff", NAME="eth0"
```

Then, the best way to make it appear is to simply restart your machine.

1.5.5 Access Control Lists

OR

Default Permissions

This is a how-to specifically for whats called ACL's or Access Control Lists. These dictate the default Username, Groupname, and/or permissions that apply to the top directory and back down, as you so choose. This is awesome for when you mount secondary hard drives, or will be using a directory as a shared folder and need unified permissions.

tl;dr

```
sudo setfacl -Rdm g:groupnamehere:rwx /base/path/members/
sudo setfacl -Rm g:groupnamehere:rwx /base/path/members/
```

setfacl is to set the acl. -R is for recursively d is for default, setting the defaults first time round m is for "modifying" the acl

Then, to make sure all files, recursively, receive the update, run it again without the d

1.6 User Management

Technically, for proper security and the Linux Way, you're supposed to have specific, security-neutered, non-home-directory-having users running these different programs. Helps stop any random, drive-by-login attempts, or rogue access if your password or keys were to ever get out.

So, if we want to go the right way, we would create a user that has no shell access, isn't allowed to actually log in, but is able to run programs.

```
sudo adduser --system --group <program-name> --home <default/program-files/location>
↪<username>
```

the <> blocks, replace the text and **<>** with your info.

So, lets break that down:

1. `--system` dictates that this is a system-type user account, which means that, by default, this user is:

- placed into the `nogroup` group by default
- a home directory is normally created for this user with the same rules as a normal user.
- the new user will have the shell `/bin/false` set
- and the user will have logins disabled by default.

1. `--group` - the man document states: "When combined with –system, a group with the same name and ID as the system user is created."
2. `--home` states that the location following the flag is the users home directory, which, if you already have the files there, it will display a **non-fatal** error. You can ignore it for now.

Then, after creating that user, you want to make sure the program files are owned by that user.

```
sudo chown -R <user>:<group> /opt/files
```

The `-R` means its recursively applied. To all files.

The other thing we also want to pay attention to is whether we have external drives mounted on our system. If we are running our Linux Software in a VirtualMachine, thus changing the way items might be mounted; and we need to pay attention to what users/groups are assigned to those external drives/directories that we might need access to, in order to process/watch/download/etc. properly!

So, for running this inside of a VirtualBox instance locally, using folder mounting through VBox, they have a custom group for the file ownership in the machine, **vboxsf**. So, to allow users to have access to read-write on these directories:

```
sudo usermod -aG vboxsf $USER
```

So, the -aG part is adding a user to a group by addition, not replacing. Then, the first name is the group, and the 2nd is the user.

Usually in order to have the addition take in the filesystem, you would log the user out then back in, but the system users normally don't have login/out abilities. So, its best to just restart the actual virtual machine.

CHAPTER 2

Raspberry Pi

2.1 Boot your Raspberry Pi from an External USB

Tired of using an SD card for your raspberry pi? Have you found them to be incredibly unstable, forcing you to reinstall the OS from scratch all the time? Using write-intensive tasks, such as running a database, or a webserver with a ton of logfiles??

Got unused hard drives laying around?

Then stop using that SD Card and start using an external hard drive!

2.1.1 First Steps

Basically, you're going to run through most of these steps just like you would the SD Card: hooking it up to a computer, copying over the OS's files (through, say, DD or DDRescue), plugging it into your Raspberry Pi and powering up.

Some OS's sometimes need a little bit of extra TLC to make the hard drive approach work correctly, tho, and can be a pain to get right.

Debian/Raspbian OS's

When using Debian or Raspbian OS's, its the simplest of processes, as you can follow the same steps as the SD Card steps.

First, you'll want to Download the version you want to use. Their website now has three different versions posted, two with the Desktop stuff (GUI) and one that they call *Lite*, or command line-only - essentially the server version, which is the one I use.

Next, their Basic Installation steps say to download and use balenaEtcher to copy the files, but I"m not sure if it'll work with an external hard drive instead of an SD Card. Otherwise, there are specific steps depending on the OS you are using to download and copy.

Linux/Unix-Based OS

2.2 DDRescue

This is my personal, favorite means of imaging an SD Card through the command line - which is the easiest, most foolproof means of doing so. Plus, its also learning, which is reason #1 for doing this....

```
| => sudo ddrescue -f ~/Downloads/ubuntu-16.04-preinstalled-server-armhf+raspi2.img /dev/disk2
GNU ddrescue 1.21
Press Ctrl-C to interrupt
     ipos:  446234 kB, non-trimmed:        0 B,  current rate:   2293 kB/s
     opos:  446234 kB, non-scraped:        0 B,  average rate:   3657 kB/s
non-tried:    3553 MB,     errsize:        0 B,      run time:      2m  2s
  rescued:  446234 kB,      errors:        0,  remaining time:         15m
percent rescued:  11.15%      time since last successful read:          0s
Copying non-tried blocks... Pass 1 (forwards)
```

A nice, command line output of its copying progress.... Not to mention ddrescue tends to be a more granular means of copying files over.

Plus, sans SD Card? Its a good backup tool to have for, well, backing up filesystems on a more granular, hardware-is-prone-to-fail type of level. Also allowing multiple passes over said data, so that if the sector pooped out the first time, maybe a 2nd or 3rd pass would, maybe not help but increase the odds of that one sector copying.

Its original intent, as you might can gather from its name, is to go over a potentially failing or failed hard drive, sector by sector, to try to pull ANY info whatsoever from that drive. Thus, the options for "retry passes," "max-errors" and the like.

2.2.1 Installing

So, if you're using macOS? I would assume/hope you have 'HomeBrew <brew.sh>'_ installed. If so, do:

```
brew install ddrescue
```

On Linux, the specific, newer and updated tool is called GDDRescue, since its a part of the GNU Library of Tools.

So, to install on debian-based linux:

```
sudo apt-get install gddrescue
```

2.2.2 How is it Used?

First, do your OS of choices means of discovering the location of the SD card you want to image:

macOS/OS X

```
diskutil list
```

```
| => diskutil list
/dev/disk0 (internal, physical):
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:      GUID_partition_scheme                        *121.3 GB   disk0
   1:                        EFI EFI                     209.7 MB   disk0s1
   2:          Apple_CoreStorage MacHD                   103.0 GB   disk0s2
   3:                 Apple_Boot Recovery HD             650.0 MB   disk0s3
/dev/disk1 (internal, virtual):
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:                 Apple_HFS MacHD                    +[illegible]   disk1
                                Logical Volume on disk0s2
                                [illegible]
                                Unencrypted
/dev/disk2 (internal, physical):
   #:                       TYPE NAME                    SIZE       IDENTIFIER
   0:     FDisk_partition_scheme                        *32.0 GB    disk2
   1:             Windows_FAT_32 system-boot            134.2 MB   disk2s1
   2:                      Linux                        3.9 GB     disk2s2
```

Which gives you the list of all drives mounted or visible on OS X/macOS. Which, usually your SD Card is last.

Next, we want to unmount the drive - not eject - so we can write to the root of the SD card.

```
diskutil umountDisk /dev/disk##
```

Replacing `##` with the disk drive number you found from before.

```
sudo ddrescue -f ~/.img/location/to/copy/to /dev/disk##
```

2.2.3 Break it Down

First, `-f` flag is for Force, telling ddrescue we REALLY want to image this source file onto this drive.

Second, the first file location. This is the source file, or what you want the SD card to look like. Or in the case of backing up a drive, the first location.

Then, the `/dev/disk##`. Make sure to replace `##` with the drive number, again. This is the final place you want the info copied to. If you were doing the backup, like above, you could specify a specific file location, name and format to output the info into. Which is beyond this documents scope. `man ddrescue`, or `ddrescue --help` can get you in the right direction.

Next, sit back and relax as you can watch the timer count down, the length of time passed count up, and plenty of other info while `dd` users have to sit and wait and HOPE it works.

2.3 OLD Boot Raspberry Pi from USB

This is now a deprecated, old means of doing this! Do not follow these steps!

So why keep this? History... Posterity? Because I can?

This how-to will show you how to begin using an external USB device - say a USB thumb drive or an External HD - as, not the "boot" device, but rather the storage location for the system files.

2.3.1 Find the USB Drive

If you're using a simple low-powered thumb drive, simply plug it into the Raspberry Pi. If you're planning on using a larger, HDD-like drive, I would suggest you use an externally powered USB hub device - to help power the external drive and not draw too much from the Raspberry Pi.

Next, type `lsusb` to see a basic break down of what your RPI can see. It doesn't exactly get you a lot of info, but it doesn't hurt to start here.

Usually, I see:

```
Bus 001 Device 004: ID 0930:6545 Toshiba Corp. Kingston DataTraveler 102 Flash Drive /
↪ HEMA Flash Drive 2 GB / PNY Attache 4GB Stick
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp. SMSC9512/9514 Fast␣
↪Ethernet Adapter
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

The top line being the flash drive that I have plugged in. But, you'll notice, it doesn't give you the nice pretty `/dev/sd[x]` that we're needing here.

Now, the website I'm using as a guide for this instructional [AFRUIT-USB] suggests using `dmesg` as a way to find your device. Though this can be messy because `dmesg` is basically your systems "this is what I'm doing and seeing all the time" kind of messaging service. Anytime a USB device is plugged or unplugged, anytime you start, stop or restart your machine, or any number of events occurring, it adds itself to `dmesg`. So, don't get flustered if you can't find your USB device.

If the last thing you did was insert your USB, it should be the last item appearing on `dmesg`

Using `sudo dmesg` should show something resembling:

```
[  459.896922] usb 1-1.2: new high-speed USB device number 4 using dwc_otg
[  460.018734] usb 1-1.2: New USB device found, idVendor=0930, idProduct=6545
[  460.026951] usb 1-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[  460.035530] usb 1-1.2: Product: TransMemory-Mx
[  460.041271] usb 1-1.2: Manufacturer: TOSHIBA
[  460.046801] usb 1-1.2: SerialNumber: 60A44C429E6BED81F000ED00
[  460.055140] usb-storage 1-1.2:1.0: USB Mass Storage device detected
[  460.064034] scsi host0: usb-storage 1-1.2:1.0
[  461.068295] scsi 0:0:0:0: Direct-Access     TOSHIBA  TransMemory-Mx   PMAP PQ: 0␣
↪ANSI: 6
[  461.081327] sd 0:0:0:0: [sda] 60929280 512-byte logical blocks: (31.2 GB/29.1 GiB)
[  461.093654] sd 0:0:0:0: [sda] Write Protect is off
[  461.100343] sd 0:0:0:0: [sda] Mode Sense: 45 00 00 00
[  461.101667] sd 0:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesnt␣
↪support DPO or FUA
[  461.114459] sd 0:0:0:0: Attached scsi generic sg0 type 0
[  463.171778]  sda: sda1
[  463.182905] sd 0:0:0:0: [sda] Attached SCSI removable disk
```

You'll notice the `sda` monikers there. That would be the `/dev/sda` that we're looking for. So, the newest USB device has been assigned `/dev/sda` on my machine. Now, of course, yours might be different. You never know.

Or, if you're `dmesg` is stupidly long and you can't find it quickly, you can always `sudo dmesg | grep sd[a-z]`

Note: `grep` being a searching program and the `sd[a-z]` being whats called a "regex" search. Basically, grep takes `sd` then adds each letter from a-z to the end and searches for those terms. It then will present you with the information,

making it easier for you to find things. The same thing works with `[0-9]` as well. I like to use it to delete a large amount of log files that tend to build up inside of `/var/log`. `sudo rm ./*.[0-9].*` or what have you.

2.3.2 Install Adafruit's USB Program

This specific program they've written automates the wiping and partitioning of the USB drive, moving of the system files, and setting up the boot files.

Adafruit's `apt-get` Repo Script

So, to install it, you'll want to add Adafruit's Linux Repo to your apt-get stuffs. You can either:

```
curl -sLS https://apt.adafruit.com/add | sudo bash
```

Which will automatically add their repo and do a `sudo apt-get update` for you. Or, you can go the long way round:

2.3.3 Adafruit's Repo the Long Way Round

- First, adding the repo to your `sources.list.d`

```
echo "deb http://apt.adafruit.com/raspbian/ jessie main" | sudo tee  /etc/apt/sources.
↪list.d/adafruit.list
```

- Then, grabbing their gpg key:

```
wget -O - -q https://apt.adafruit.com/apt.adafruit.com.gpg.key | sudo apt-key add -
```

- And then run `sudo apt-get update`

Now, before we go to far, why don't we break all that down?

1. The `echo deb http://apt.adafruit.com/raspbian/ jessie main` part first

- `echo` so that we can copy and pipe that quoted text
- `sudo tee` is a way to take text thats been piped into it, and either overwrite or append that text to a file. In this case, overwrite to `/etc/apt/sources.list.d/adafruit.list`

2. The GPG Key

- `wget` is a program to download things from the interwebs
- `-O - -q`: - the `-O` is for amending the output of the download - the `-` is saying the output is `STDOUT`, or `copy it to output so we can pipe it` - the `-q` is a flag saying "run it quietly" or "no output preferred other than the file"
- `sudo apt-key add -` is to add the downloaded key, and the `-` is saying "take the `STDOUT` from the `wget` and use that"

2.3.4 Installing USB Program

Next, you'll want to install Adafruit's USB program.

```
sudo apt-get install adafruit-pi-externalroot-helper
```

2.3.5 Running the Program

Then, once you know for sure the file location moniker of your USB device:

```
sudo adafruit-pi-externalroot-helper -d /dev/sda
```

The `-d` flag is to tell it the file location moniker of your USB device.

Note: Make SURE you get this right, as you don't want to really wipe your SD card. Though I'm fairly certain not only would that not work, it luckily is an easy fix to get either a clean OS on it, or a backup you've made.

This program does take at least a few minutes to run through. You'll need to restart your RPI as well once its finished.

2.3.6 After Running

It will give you a series of messages once its done. Make sure to read through them, in case there are any errors. It also tells you how to make sure it worked:

```
[boot config] Ok, your system should be ready. You may wish to check:
[boot config]   /mnt/etc/fstab
[boot config]   /boot/cmdline.txt
[boot config] Your new root drive is currently accessible under /mnt.
[boot config] In order to restart with this drive at /, please type:
[boot config] sudo reboot
```

fstab

The key lines are:

```
Ok, your system should be ready. You may wish to check:
    /mnt/etc/fstab
    /boot/cmdline.txt
```

It really means you need to check `/etc/fstab`. Not sure why they included the extra `/mnt` in there.

My `/etc/fstab` now shows:

```
/dev/mmcblk0p1 /boot vfat defaults 0 2
#/dev/mmcblk0p2 / ext4 errors=remount-ro,noatime,nodiratime,commit=120 0 1
tmpfs /tmp tmpfs defaults,nodev,nosuid 0 0
/dev/disk/by-uuid/94551cfd-d0fc-42df-b742-b7a6434c0d8a    /    ext4    defaults,
↪noatime  0       1
```

Notice the commented out line `#/dev/mmcblk0p2`

SD Card Info

That was the original line for the sd card. The line prior was also there before, as the SD card was both the boot media and the OS media. But, now the SD is ONLY the boot, and the external USB is the OS media.

Running `df -h`

You can also `df -h`. This shows your filesystem stuffs. The `-h` being human readable format on the sizes.

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        29G  1.2G   26G   5% /
devtmpfs        483M  4.0K  483M   1% /dev
none            4.0K     0  4.0K   0% /sys/fs/cgroup
tmpfs           487M     0  487M   0% /tmp
none             98M  232K   98M   1% /run
none            5.0M     0  5.0M   0% /run/lock
none            487M     0  487M   0% /run/shm
cgmfs           100K     0  100K   0% /run/cgmanager/fs
none            100M     0  100M   0% /run/user
/dev/mmcblk0p1   61M   36M   26M  58% /boot
tmpfs            98M     0   98M   0% /run/user/1000
```

The location `/dev/root` is the USB drive now. And, it shows the large size of the USB drive as well.

2.3.7 Recovering from a Failed Boot

If the RPI should ever not wanna work correctly with the USB drive this way, you can always:

1. take the SD card out of your RPI, plug it into your regular machine that you used to install the OS.
2. Then, open the first partition and find the file `cmdline.txt`.
3. Replace the text `root=PARTUUID=....rootdelay=5` with `root=/dev/mmcblk0p2`, which will point the root partition back to the 2nd partition of your SD card. It should roughly look like this:

```
dwc_otg.lpm_enable=0 console=ttyAMA0,115200 console=tty1 root=/dev/mmcblk0p2␣
↪rootfstype=ext4 elevator=deadline rootwait fbcon=map:10 fbcon=font:VGA8x8
```

Then, place the SD card back into the RPI, and it should boot normally.

CHAPTER 3

macOS

For right now, these are just a few of my own, personal, musing for macOS.

macOS is my primary computer that I use for everyday use - aka, everything. But, most of my personal development stuff occurs on my Ubuntu VM's or my Raspberry Pi's.

3.1 Updating X-Code After Big Updates

Created/Published 9/11/16

3.1.1 tl;dr

Reminder! You need to:

```
xcode-select --install
```

after a big macOS update!

3.1.2 The Story

So, after almost 2 days now of futzing with my different terminal apps and what not, and terminal giving me issues not wanting to compile anything… I TOTALLY forgot to run `xcode-select --install` to download the new developer tools.

Moral of the story?

Love your X-Code!

3.2 Download Beta Xcode with `wget`

So, I was looking for a way to download the Beta version of Xcode over the terminal, using either `wget` or `curl`, and almost every suggestion I found got it wrong somehow.... Some made it more complicated than it should be or was probably too old a suggestion for Apple's current login systems...

Either way, I finally found a solution from Codesd's website that worked wonderfully, and it was the tidbit at the bottom of their post that made the specific difference:

```
for some reason cookies.txt only works if I only export cookies for the developer.
↪apple.com website and doesnt work if I export all cookies from browser.
```

3.2.1 How To

1. First, you'll need to login to the developer portal at *Beta Xcode Download <xcode-beta>*
2. You'll want to copy the download link for whichever Xcode version you are wanting/have access to.
    1. start the download, then cancel it
    2. Then, go to your browser's "downloads" page, where you can copy the specific download URL
3. You'll need to download the cookies specifically for Apple's Developer Webpage, after you've logged in. I don't think Safari has an extension for downloading cookies, but I know Google Chrome has a few. I personally used cookies.txt.

As Codesd's Blog Post put it:

> I in no way endorse this product and many other solutions in different browsers should work just as well.

4. Once the extension is installed, go back to Apple's page and hit the extension's button. If, like me, the extension's little popup thing doesn't fully expand, these instructions will help:
    1. Once you hit it, and if the popup shrinks, you can just hit the `Enter` key
    2. That will download the website's cookies into a `txt` file for you automatically
    3. This information is in the extension's `help` page if you right click the extension and select `Options`
5. And now the actual bash stuff:

```
wget --load-cookies=cookies.txt -c https://download.developer.apple.com/Developer_
↪Tools/Xcode_9.2_Beta_2/Xcode_9.2_Beta_2.xip
```

Replace the specific URL with the URL that you want/need to use, please.

And, you should find that the download this way is much much quicker than through the browser. Which is wonderful when the file is so gargantuan.

3.2.2 Unzipping

For the final step of unzipping/unpacking/un-whatever-its-technical-term-is-for-xip-files, if you have a 3rd party "unzipping" utility like I do (The Unarchiver is my personal favorite) you'll want to make sure that your system will NOT use it!

Due to the file being a "XIP Secure Archive", as it says in Finder.app, the other tools will fail or throw errors when trying to take care of this `.xip` file. So, use the system's `Archive Utility.app` to "inflate" the file.

IT WILL TAKE A BIT OF TIME... I assume due it being "Secure", its having to unencrypt whilst expanding. Plus its huge.

3.3 Other Random Tidbits

This page will be for those little nuggets of info that I have to have documented or else when I forget again, I'll remember having known how to fix it before, but no clue on what to do or where to go. The worst feeling in the world!

This is to stop that.

3.3.1 Local WebPage Wont Load Anywhere!!!

A locally-hosted site refused to load in any web browser on the same computer, but it would load in my iPhone...

Essentially the DNS cache was poisoned and needed to be flushed. The only command that worked was:

```
dscacheutil -flushcache && sudo killall -HUP mDNSResponder
```

which I have saved as an alias.

CHAPTER 4

VirtualBox

These how-to's are directed toward using VirtualBox, geared moreso towards using it on a macOS host, as that is how I use my VirtualBox. So, apologies to anyone not using macOS as their host.

4.1 VirtualBox AutoStart

This article has been expanded to cover not just macOS but Linux (Specifically Ubuntu Tested) as well.

Note: Some steps are the same for both macOS and Linux, but there are a couple that deviate.

4.1.1 AutoStart Systems

This first step is requried for both Linux and macOS, but is different in how you get to step two.

Linux

You should have a `default` file inside of `/etc/default` for virtualbox.

```
sudo nano /etc/default/virtualbox
```

If this is your first time editing this file, it should be blank. This file is sourced by any number of VirtualBox's autostart files/scripts/etc. that the program utilizes at any one time.

Then you will insert:

```
VBOXAUTOSTART_DB=/etc/vbox
VBOXAUTOSTART_CONFIG=/etc/vbox/autostart.cfg
```

- `VBOXAUTOSTART_DB` which contains an absolute path to the autostart database directory
- `VBOXAUTOSTART_CONFIG` which contains the location of the autostart config settings.

Next, jump to *Linux Autostart*

macOS

First, we need to copy - or rather, link - over the VirtualBox-provided default .plist file - aka: what macOS uses as their "services" files. Or, in comparison, like Linux's /etc/init.d/ files or the like.

This how-to was lightly copied over from ReidRansom's Gist.

Linking The .plist

```
sudo ln -s  \
  "/Application/VirtualBox.app/Contents/MacOS/org.virtualbox.vboxautostart.plist" \
  /Library/LaunchDaemons/
```

There is another directory that has similar files - `/Library/Application Support/VirtualBox/LaunchDaemons/` - but those don't seem to like to work properly... So, we're ignoring those.

Note: In case you aren't sure, the extra - - is how you tell the linux command line "Don't run this code yet, even though I am pressing enter or return!". Its basically a newline-signifier. You'll notice it way way more often in how-to's, where it is an attempt to break up the code so its easier to see, visually, for you.

Edit The .plist

And next we make our edits to `/Library/LaunchDaemons/org.virtualbox.vboxautostart.plist`

The default option `Disabled` is set to `true`, so even if launchctl loads it mistakenly, it wont work. So, therefore, that needs to be changed to `false` to activate it.

1. Set `Disabled` to `false`
2. Adding `KeepAlive` and `<true/>`
3. Adding `<string>--start</string>` after the `.sh` line
4. Set `RunAtLoad` to `true`
5. Set `LaunchOnlyOnce` to `true` - which this one is probably the 2nd most self-explanatory line.

My personal .plist file looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
↪PropertyList-1.0.dtd">
<plist version="1.0">
<dict>

  <key>Disabled</key>
  <false/>
  <key>KeepAlive</key>
  <true/>
  <key>Label</key>
  <string>org.virtualbox.vboxautostart</string>
  <key>ProgramArguments</key>
```

(continues on next page)

(continued from previous page)

```
  <array>
    <string>/Applications/VirtualBox.app/Contents/MacOS/VBoxAutostartDarwin.sh</
↪string>
    <string>--start</string>
    <string>/usr/local/etc/vbox/autostart.cfg</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
  <key>LaunchOnlyOnce</key>
  <true/>

</dict>
</plist>
```

Next, jump to *macOS Autostart*

4.1.2 AutoStart.cfg File

The final file will look roughly the same for both Ubuntu and macOS systems, but where to place the file is different for both.

Linux Autostart

As we referenced above inside of your `/etc/default/virtualbox` file, we need a corresponding `/etc/vbox/autostart.cfg` file now, as well as a directory at `/etc/vbox`

If `/etc/vbox` is not created:

```
sudo mkdir /etc/vbox
```

And then:

```
sudo nano /etc/vbox/autostart.cfg
```

Now, the `autostart.cfg` file that we are creating here is specifically geared towards managing user permissions in relation to if the VirtualMachines will autostart upon that specific user logging into the system. So, if you have more than one user, and they don't use VirtualBox, or you don't WANT them to autostart boxes, you can deny them that access.

VirtualBox states that its best to have a "Deny by Default" setup. Even with just one single user on your machine, its basically a nice, peace of mind that, even if you ever add another user, nothing hanky will occur with VirtualBox.

And, of course, you can also use `allow` in lieu of `deny`, if you so choose.

```
default_policy = deny

berto = {
    allow = true
    startup_delay = 30
}
```

The `startup_delay` line is a means of controlling whether your Virtual Machines attempt to start exactly with the system or delayed, in seconds, afterwards. Which, I have it set to a 30 second delay, so it isn't competing with the rest of the bulk of the startup items.

And, of course, make sure to change `berto` to the username on your system that the VirtualMachines are running under.

macOS Autostart

Now, different online how-to's elsewhere often say to make a directory within your `/etc` directory. But, if you are a user of Homebrew, you'll likely know that, often times, the better location for user-used files of most any kind would prefer to reside inside of `/usr/local`. The reason for this, as the directory names suggest, is more geared towards the users specific files, binaries, configuration files, etc.

Often times, on a mac system, programs that normally on a Linux machine would stick their files within `/etc`, instead, use the `/usr/local/etc` location. One reason for this is the more lax permissions set for these locations. Plus, Apple has publicly stated that this directory is more or less hands off from their constant meddling when it comes to updates for the system. Therefore, your files are much more likely to remain untouched by the OS.

Now, the `autostart.cfg` file that we are creating here is specifically geared towards managing user permissions in relation to if the VirtualMachines will autostart upon that specific user logging into the system. So, if you have more than one user, and they don't use VirtualBox, or you don't WANT them to autostart boxes, you can deny them that access.

So, first, create the `vbox` directory: `mkdir /usr/local/etc/vbox`

And then:

```
nano /usr/local/etc/vbox/autostart.cfg
```

VirtualBox states that its best to have a "Deny by Default" setup. Even with just one single user on your machine, its basically a nice, peace of mind that, even if you ever add another user, nothing hanky will occur with VirtualBox.

And, of course, you can also use `allow` in lieu of `deny`, if you so choose.

```
default_policy = deny

berto = {
    allow = true
    startup_delay = 30
}
```

The `startup_delay` line is a means of controlling whether your Virtual Machines attempt to start exactly with the system or delayed, in seconds, afterwards. Which, I have it set to a 30 second delay, so it isn't competing with the rest of the bulk of the startup items.

And, of course, make sure to change `berto` to the username on your system that the VirtualMachines are running under.

4.1.3 File Permissions

And now, we need to go through and make sure the different files permissions are set properly.

Linux File Permissions

```
sudo chgrp vboxusers /etc/vbox
sudo chmod 1775 /etc/vbox
```

Then, for each username you will be using for the AutoStart:

```
sudo usermod -aG vboxusers $USER
```

macOS File Permissions

```
sudo chmod +x /Applications/VirtualBox.app/Contents/MacOS/VBoxAutostartDarwin.sh
sudo chown -R root:wheel /usr/local/etc/vbox
sudo chown -R root:wheel /Library/LaunchDaemons/org.virtualbox.startup.plist
```

Note: You'll notice I used the `-R` flag for the final item, even though it itself is just a file. But, remember, its a linked file, which do not change their permissions on the linked location without the `-R` flag.

4.1.4 `VBoxManage modifyvm`

Note: This final step is the same for both systems (YAY!)

Now we need to run `VBoxManage` to change the settings for the VirtualMachines that we want to start at boot.

Which, there is almost a **literal** ton of settings, commands and options you can set through the command line, of which the large majority of them you'll never see in the GUI program. Why? I have no clue.

Make sure the guest machine you're changing settings on is not currently running. Then:

```
VBoxManage modifyvm "$VM_NAME" --autostop-type acpishutdown
VBoxManage modifyvm "$VM_NAME" --autostart-enabled on
```

1. The first line specifies how VirtualBox should try to shut the machines down, if they are still running when the system starts shutting itself down. `acpishutdown` corresponds to sending the machine a `acpipowerbutton` shutdown command. Which is to say, the normal means of properly shutting down a machine. You can also select `disabled`, `savestate` or `poweroff`.
2. The second line is the actual option for telling VirtualBox that "this VirtualMachine we want to autostart."

And make sure to replace `$VM_NAME` with your VirtualMachines registered names.

4.1.5 Testing

Finally, we can now test our configs without having to restart our machine.

Note: Before running this, make sure your machines are turned off, so you can watch them turn on.

Linux

```
sudo service vboxautostart-service start
```

macOS

```
sudo launchctl load /Library/LaunchDaemons/org.virtualbox.startup.plist
```

At this point, your configured guest VM's should begin running. You can test what VM's are running by the command:

```
VBoxManage list runningvms
```

4.2 VirtualBox Guest Additions

VirtualBox Guest Additions is, more or less, VirtualBox's way of making it easier for your host machine to speak to your guest machine, and do all sorts of extra things. Examples:

1. Easier File Sharing

- Instead of dealing with mounting drives in an `/etc/fstab` file, VirtualBox has a means of setting up a simple Directory sharing scheme
- You pick certain directories on your host machine to be shared directly into the `/media/<folder name you set>` directory!

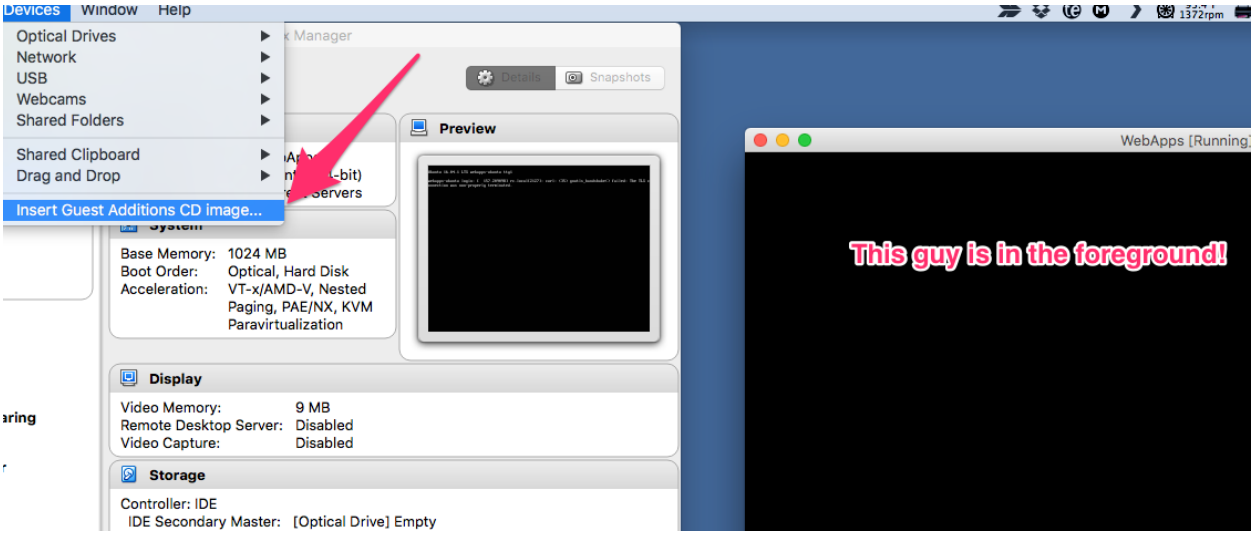
2. If you're running a GUI:

1. Shared Clipboard
2. Drag'n'Drop files back and forth
3. Mouse Pointer Integration - You don't have to "release" the mouse from the guest screen
4. Better Video Support
5. Seamless Windows - This opens individual windows in the guest as if they were individual windows on the host directly, eliminating the boundary between the host and guest OS.

First, we will show how to begin the install on our Host machine, then directly into our guest machine.

4.2.1 Host Machine

So, first we need to mount the Guest Additions CD Image, which you can find under the "Devices" menu in the Virtual Machines Menu Bar. (Make sure the Guest Machine window is in focus to show that machines menu bar.).



Then, depending on what OS your guest is running depends on what the next steps are. We will be concentrating on Ubuntu Server, Headless installs. Seeing as that needs a big more work on our part than any GUI-based system.

4.2.2 Ubuntu Server Guest Machine

Pre-Requirements

First, lets install the pre-requirements.

```
sudo apt-get install dkms make gcc -y
```

Mount the CDROM

Then, to install the Guest Additions onto your Headless Ubuntu Server addition, you first have to mount the cd onto your directories so you can access whats inside.

```
sudo mount /dev/cdrom /media/cdrom
```

Mounting Info

The `/dev/cdrom` is where your cdrom "hardware" is located in order for the software to access it. As in this would be the actual, physical cdrom drive if your server was physical.

Then, the `/media/cdrom` is where you're mounting your cdrom so you can access the files.

Run the Installer

It'll alert you to the fact that the mounted filesystem is `read only`. You're good to ignore that since we wont be making edits to the files. Then, cd into the mounted location, and run the installer.

```
cd /media/cdrom
sudo sh ./VBoxLinuxAdditions.run
```

That will take a hot second, at least, to run the installer. But, once its finished - and there were no error messages - go ahead and restart your machine.

Note: Make sure you have setup all of the configuration options in the machine settings, as in the folders you want to have shared from host to guest.

---

usermod

Now, you'll need to add the group name `vboxsf` to all of the different system and user accounts.

Note: `vboxsf` is VirtualBox's group-ownership-way of mounting these directories.

Services Info

If you have any services with custom user/group names, like `transmission-daemon`, stop those services before the next step please.

```
sudo usermod -aG vboxsf $USER
sudo usermod -aG vboxsf debian-transmission
sudo usermod -aG vboxsf root
```

Then, it doesn't hurt to restart your machine. Just to make sure all accounts have signed out and back in again, so they can access any *vboxsf* group items.

CHAPTER 5

General Bash

This section is for those items that are usually low-level enough to be fairly evenly applicable to all-things bash, independent of the OS its running on - save for Windows and them getting a bash version from Ubuntu... Now thats just weird!

5.1 Bash Conditional Expressions

Conditional Expressions are signals - also called primaries - that bash uses as a means of performing tests, using the info you provide.

Lets say you are wanting to copy over a file from one location to another, but don't need to do this if the file already exists in the final location. This is one way to test for that!

```
if [[ ! -e /etc/foo ]]; then
  echo "Looks like /etc/foo doesn't exist"
fi
```

5.1.1 Breakdown

1. `[[ ]]` So, the double brackets here are necessary for `#!/bin/bash`, of which, to be honest, using the double brackets as the standard setup seems to be what makes the most sense.
2. `!` The exclamation point is how you negate expressions through a lot of bash in general. So, here, its testing if `/etc/foo` exists.
3. `-e` True if File Exists. Its a very broad testing flag.
4. After the brackets are closed, you use `;` to tell bash that your expression line is finished.
5. Then, you do whatever you need to do.
6. `fi` then, the `fi` is what tells bash that `if` is finished.

Conditional Expression Primaries

The following test flags go with:

```
if [[ -e /bin/bash ]]; then ...
  <some code>
elif [[ ! -e /bin/dash ]]; then
  <some code>
fi
```

Use `[[ ]]` with double == and use `[ ]` with =

The first line, `-e /bin/bash` is the positive side of the `-e FILE` test.

There are a LOT of ways to test things, using either `if-then-else`, `case` or `while` loops.

Table 1: Test Expressions

Flag	Explainer
-a FILE	True if file exists
-b FILE	True if file exists and is a block special file
-c FILE	True if file exists and is a character special file
-d FILE	True if file exists and is a directory
-e FILE	True if file exists
-f FILE	True if file exists and is a regular file
-g FILE	True if file exists and its set-group-id bit is set
-G FILE	True if file exists and is owned by the effective group id
-h FILE	True if file exists and is a symbolic link
-k FILE	True if file exists and its `sticky` bit is set
-L FILE	True if file exists and is a symbolic link
-n STRING	True if the length of string is NON-zero
-N FILE	True if file exists and has been modified since it was last read
-o OPTNAME	True if the shell option `optname` is enabled[1]
-O FILE	True if file exists and is owned by the effective user id
-p FILE	True if file exists and is a named pipe (FIFO)
-r FILE	True if file exists and is readable
-R `$VARNAME`	True if the shell variable `$VARNAME` is set and is a *name reference*
-s FILE	True if file exists and has a size greater than zero
-S FILE	True if file exists and is a socket
-t FD	True if file descriptor `fd` is open and refers to a terminal
-u FILE	True if file exists and its set-user-id bit is set
-v `$VARNAME`	True if the shell variable `$VARNAME` is set (has been assigned a value)[2]
-w FILE	True if file exists and is writable
-x FILE	True if file exists and is executable
-z FILE or STRING	True if STRING or FILE is null

Table 2: Test Expressions - File Comparison (1)

Compare Flags	Explainer
file1 -nt file2	True if file1 *exists* and is *newer* than file2
file1 -ot file2	True if file1 *exists* and is *older* than file2
file1 -ef file2	True if file1 and file2 *exist* and refer to the *same file*

[1] Shell Option `optname`: The list of options appears in the description of the -o option to the set builtin. (see The Set Builtin)
[2] `$VARNAME` is replaceable with ANY **VARIABLE** name needed

Table 3: Test Expressions - File Comparison (2) Detailed

Compare Strings	Explainer
s1 == s2	True if strings `s1` and `s2` are *equal*
s1 = s2	True if strings `s1` and `s2` are *equal*
s1 != s2	True if strings `s1` and `s2` are *not* equal
s1 < s2	True if string `s1` comes *before* `s2` based on the binary value of their characters
s1 > s2	True if string `s1` comes *after* `s2` based on the binary value of their characters

Note: Bash-based conditionals - `#!/bin/bash` - usually prefer you to use double-brackets, `[[ ]]`. When you use `[[ ]]`, you have to use double-equals - ==. You can still use the single-brackets with single-equals =.

But when using `sh` - `#!/bin/sh` - it prefers single-brackets - `[ ]`. When you use the single-brackets, you use single-equals - =.

Table 4: Test Expressions - File Comparison (3) MATH SPECIFIC

Compare Strings	Explainer
n1 -eq n2	True if the integers `n1` and `n2` are algebraically equal
n1 -ne n2	True if the integers `n1` and `n2` are not algebraically equal
n1 -gt n2	True if the integer `n1` is algebraically greater than the integer
n1 -ge n2	True if the integer `n1` is algebraically greater than or equal to the integer `n2`
n1 -lt n2	True if the integer `n1` is algebraically less than the integer `n2`
n1 -le n2	True if the integer `n1` is algebraically less than or equal to the integer `n2`

Note: See `man test` for more explanations.

5.2 Bash Logging

tl;dr:

```
exec 1> >(logger -s -t "$(basename "$0")") 2>&1
```

Been looking for a way to pipe your entire bash script into a *log file*, *syslog*, *journald*, or whatever? The above line does it super simply!!

This was a line I had found a few years back, started learning ansible, and lost this nugget of knowledge, and recenlty found it once again (but lost the website I found it from... Sorry!)

5.2.1 What does it do?

- `exec`: this line tells bash "you are to run everything"
- `1> >(` this sets up piping `1` (`stdout`) through to whats inside the parenthesis
- `logger -s -t "$(basename "$0")")` the `logger` program is a means of piping output into your systems `syslog` or `rsyslog` or what have you. `basename` adds the file/scripts name onto the syslog entry.
- `2>&1` is for piping `2` or `stderr` into the `exec` command

This information was wonderfully pulled from UrbanAutomation's website.

5.3 Some Basic Linux Commands

5.3.1 SSH or Remote Use

To connect to another system from your current machine, you use `ssh`

```
ssh user@ipaddress
```

My username I use is `jpartain89`, so :

```
ssh jpartain89@192.168.1.10
```

It will ask for your users password, if your ssh keys are not setup.

5.3.2 SSH Keys

SSH-Key Generate

To generate SSH keys:

```
ssh-keygen -t rsa -b 2096
```

It will ask for location, hit `enter` for the standard default location.

If you want a password on this key, for extra extra security, you can type that in with the next option. Or, for no passphrase, just hit `enter`.

Hit `enter` again for confirmation, or retype the password if you wanted one.

Then, it'll output the key 'randomart' image, which doesn't mean much. But, the default location is `~/.ssh`. It'll tell you where.

SSH Key Location

The private key is like your front door key on your keyring. Only you have it. The public key is like the tumblers. Anyone can see it, but cannot access without your private key.

SSH-Copy-ID or Sending Your Public Key

The public key is copied to the servers that you want to access without password input. There is a program to do that. `ssh-copy-id`. On macOS, `brew install ssh-copy-id`. Linux should already have it installed.

To send your public keys, you use

```
ssh-copy-id jpartain89@192.168.1.10
```

Or `ssh-copy-id` is the program, then `your username @ the ip address of the computer`. It'll ask for your password, then copy the keys. Then, `ssh jpartain89@192.168.1.10` and it should no longer ask for your password!

5.3.3 Restarting the Machine

```
sudo shutdown -r now
```

So, breakdown:

1. `shutdown` - obviously is the shutdown program
2. `-r` - the -r flag is for restarting.
3. `now` - tells it to do it now, rather than wait. You can set times before it shutsdown or restarts, in case other users are on the machine.

5.3.4 Updating the Apps

```
sudo apt-get update && sudo apt-get upgrade -y
```

Breakdown:

1. `apt-get` is the installing program that debian/ubuntu uses. Its a super simple means of installing in the terminal.
2. `update` - obviously runs an update. This means it downloads a list or a cache of the version numbers and states of the programs in apt-get's lists.
3. the `&&` means `run this stuff after the last one succeeds successfully`
4. `upgrade` - this actually takes the programs installed by apt-get, and upgrades them.
5. `-y` - basically means `yes, do this without asking like normal`

5.4 Bash Logging

tl;dr:

```
exec 1> >(logger -s -t "$(basename "$0")") 2>&1
```

Been looking for a way to pipe your entire bash script into a *log file*, *syslog*, *journald*, or whatever? The above line does it super simply!!

This was a line I had found a few years back, started learning ansible, and lost this nugget of knowledge, and recenlty found it once again (but lost the website I found it from... Sorry!)

5.4.1 What does it do?

- `exec`: this line tells bash "you are to run everything"
- `1> >(` this sets up piping `1` (`stdout`) through to whats inside the parenthesis
- `logger -s -t "$(basename "$0")")` the `logger` program is a means of piping output into your systems `syslog` or `rsyslog` or what have you. `basename` adds the file/scripts name onto the syslog entry.
- `2>&1` is for piping `2` or `stderr` into the `exec` command

This information was wonderfully pulled from UrbanAutomation's website.

5.5 Use grep Output as if-else Conditional

This was my notation of how to use grep's standard output as an `if-else` conditional in a shell script.

Basically, just about all programs have at least two outputs; the textual output that it sends to `STDOUT`, the stuff that you read and a basic `0 1` or more, for use in shell scripts, `if-else` conditionals or loops, etc.

The `if-else` conditionals or `for` loops or what have you like to use simple numbers, `0 1` etc. for, say, when searching for `openvpn` being installed, it says `0` for yes, `1` for no or error. Therefore, you can make a conditional that says `if 0, yes, then do not attempt install. If NOT 0, then attempt install!`

5.5.1 Basic Use of grep in if-else

This would be a simple layout of how you could potentially use this in a shell script.

```
#!/bin/bash

dpkg -l | grep openvpn &> /dev/null
if [[ $? == 0 ]]; then
  echo "matched"
else
  echo "Not Matched"
fi
```

5.5.2 So, lets break that down

1. `dpkg -l` - this, by itself, will output all installed programs that dpkg manages.
2. `|` - this is the `pipe` command.

- What this says is: take the outputted text from the last thing and use it as input for the next thing
- So, the entire list of installed apps then is searched using `grep` for `openvpn`

3. `&> /dev/null` - this is a redirection of the output, negating any textual output, leaving only the `1` or `0` or more outputs to use
4. `$?` - this basically says `take the numbered output, whatever it is, and stick it here`
5. So, when you do `$? == 0` you're saying `if the output was 0, then do this command`

AND be careful, 0 might not always be yes or confirm. Make sure to check the programs MAN pages for clarification

5.6 Here Docs

```
cat << EOF > filename.sh
#!/bin/bash -e

# bunch of scripty stuff
##

EOF
```

5.7 pip Requirements File

tl;dr a pip Requirements File is a list of programs "required" to be installed. Lets say you have a program you wrote that has required python programs, you'd include them here.

I personally use the requirements file as a simple `install/upgrade` list. aka:

```
sudo -H pip install --upgrade -r requirements.txt
```

This file is much more detailed and expansive than I care to currently go into.

5.8 Random Collection of Info

- *Using find to chmod multiple files*: Use `find` to `chmod` ONLY files
- *Using awk to Modify Output*: Using `awk` to manipulate some output - small amount of examples
- *Using \ as New Lines*: Example of using the `\` to signify a return carriage to a line of commands - IE keep processing as if its one line
- *Shell Script Location*: bash line to assign the current location of the bash script to a variable.
- *Using sed to Make Updates* : Using `sed` to update the same text in multiple files.

This page is a decent place for me to put my collection of information like shell scripting one liners, or little nuggets of "this-isn't-enough-to-warrant-an-entire-solo-page-but-is-important-enough-to-document" type of ites.

5.8.1 Using `find` to `chmod` multiple files

When you find yourself needing to change a plethora of items within the current working directory - but only the files or directories and not both - one way is to use `find`

```
find . -type f -exec chmod 644 {} \;
```

Break Down

1. `find` - this is obviously the specific program/command here. It is an incredibly useful tool, but we're only covering a small portion of its abilities here.
2. `.` - the `.` signifies we're searching in the current working directory, or if you typed `pwd`, its the same idea. You can change this to any location in your directory you so choose.
3. `-type f`

5.8.2 Using `awk` to Modify Output

When a program outputs information in a standardized way, like `pip freeze` or `pip list`, you can manipulate that output to fit your wants and needs.

So, lets take `pip list`, below is a truncated output.

```
alabaster (0.7.9)
ansible (2.2.0.0)
appnope (0.1.0)
argh (0.26.2)
astroid (1.4.8)
Babel (2.3.4)
```

You can also see `pip freeze` has its own means of outputting info. Each command has its own reasons.

```
alabaster==0.7.9
ansible==2.2.0.0
appnope==0.1.0
argh==0.26.2
astroid==1.4.8
Babel==2.3.4
```

But, lets say we want to pipe all of that to a *pip Requirements File* for easy updating/reinstallation. If you're using said file for updates/upgrades, having the included version numbers would not help at all here, since `pip install -r requirements` would install those specific versions.

So, what do we do?

```
pip list | awk '{ print $1 }'``
```

or

```
pip freeze | awk -F'==' '{ print $1 }'``
```

Breakdown

1. the `pip freeze` and `pip list` we've established.
2. awk - is a language in and of itself, as complicated and large and useful as a language as well.
3. -F - this says "use the following text inside the ' ' as the break point or escape character to seperate out all of the info."
4. '{ print $1 }' - this tells awk to show the first column of information only.

If you were to say `pip list | awk '{print $1,$2}'` you would get the original information once again. Why? becuase the (#.#.#) is `$2` or option 2 or what have you. The , says "insert space". Without the comma, no space.

Again, `awk` is a massive language. This is a simple explainer here.

5.8.3 Using \ as New Lines

Often times, you'll see `\` used at the end of code lines and you've wondered what on EARTH thats about??

Well, those are used as so-called `new line` signifiers, or on the naked command line, it tells the system to keep expecting more text/code input.

5.8.4 Shell Script Location

Are you wanting an easy way for your shell script to know where its at in the plethora of unix directories? Use the below line!

```
"$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
```

Its able to give you that info, no matter where its called from, which is a typical issue with almost all other so-called "one-liners".

5.8.5 Random Number Generator

tl;dr:

```
Using the following code gets you a ``Random Number`` between ``0 and 3600``␣
↪:guilabel:`&Plus` ``3600`` or ``1 hour``
```

```
SLEEP="$(($RANDOM%3600+3600))"
```

Details

The `$RANDOM` bash variable is a builtin variable for generating random numbers, random options between *True* and *False*, mimic rolling a dice, drawing cards, etc.

its a nice and simple way to use a randomized *Sleep* length, or anything else you want randomized thats not reliant upon anything security-wise. (Its no where near random enough for using with security needs)

5.8.6 Using `sed` to Make Updates

If you are wanting to make the same change in multiple files, `sed` is the way to go!

```
sed -i 's/brightyellow/,yellow/g' /usr/share/nano-syntax-highlighting/*.nanorc
```

or, if you are on a mac, you have to add `''` after the `-i`, and before the text to replace.

```
sed -i '' 's/brightyellow/,yellow/g' /usr/share/nano-syntax-highlighting/*.nanorc
```

A great website to go look at for a plethora of how-tos is tldp_randomvar.

5.9 Ratom [RATOM-src]

Basically, this is a way of using the text editor, Atom.io remotely to edit files.

Say you are working from your regular computer, and remotely `ssh` into another machine, and you need to edit a text file, and you just know deep down that if you were able to use a full text editor, rather than VIM or NANO in the terminal, it'd be SO MUCH EASIER!

Thats where ratom comes in.

First, when you SSH, instead of the normal commands, you forward a port from the remote machine to your local machine. In the case of ratom, its `52698`. Then, you would run the prior installed `ratom` command on the remote machine, on the text file you want to edit. `ratom /etc/init.d/couchpotato` At which point it then opens that file remotely from the server to your local machine inside of Atom, as if it was local!

5.9.1 Installing Ratom

First, in order to pipe the text file over that port into your local atom, you have to install a program to handle it on the remote machine.

```
sudo curl -o /usr/local/bin/ratom https://raw.githubusercontent.com/aurora/rmate/
↪master/rmate
sudo chmod +x /usr/local/bin/ratom
```

5.9.2 SSHing with Forwarded Ports

Then, exit that ssh session, and then reconnect with the port forwarded:

```
ssh -R 52698:localhost:52698 user@example.com
```

What this line is saying is:

1. `-R` - Forward the following port FROM the remote machine to my local machine
   - the opposite would be an `-L` flag
2. `52698:localhost:52698` - map the remote 52698 port to my local 52698 port, from the remotes `localhost`, and not another address.
3. and then your username and server address.

5.9.3 Running ratom on the Server

Once you're reconnected, you can open the file from the remote system onto your local Atom by:

```
ratom /etc/init.d/couchpotato
```

So, the `ratom` command is the program, and then the text file.

5.10 Signals and Traps

When writing shell scripts, and you're creating `tmp` files or doing anything that needs to or should be cleaned up either after the script is finished or in the event of an error, you should include a "cleanup" function and possibly an "error" function that first calls the "cleanup" function and presents an error code or message.

The main way of doing this is by the `trap` function/command. Basically, it "traps" any exit codes/commands/signals presented by the script, by the user or by the system.

Below is the various signal names or number codes you should include in your trap line.

Table 5: Trap Signals

header "No"	Name	Default Action	Description
widths auto			
align center			

"1 ", "SIGHUP", "terminate process", "terminal line hangup" "2 ", "SIGINT", "terminate process", "interrupt program" "3 ", "SIGQUIT", "create core image", "quit program" "4 ", "SIGILL", "create core image", "illegal instruction" "5 ", "SIGTRAP", "create core image", "trace trap" "6 ", "SIGABRT", "create core image", "abort program (formerly SIGIOT)" "7 ", "SIGEMT", "create core image", "emulate instruction executed" "8 ", "SIGFPE", "create core image", "floating-point exception" "9 ", "SIGKILL", "terminate process", "kill program" "10", "SIGBUS", "create core image", "bus error" "11", "SIGSEGV", "create core image", "segmentation violation" "12", "SIGSYS", "create core image", "non-existent system call invoked" "13", "SIGPIPE", "terminate process", "write on a pipe with no reader" "14", "SIGALRM", "terminate process", "real-time timer expired" "15", "SIGTERM", "terminate process", "software termination signal" "16", "SIGURG", "discard signal", "urgent condition present on socket" "17", "SIGSTOP", "stop process", "stop (cannot be caught or ignored)" "18", "SIGTSTP", "stop process", "stop signal generated from keyboard" "19", "SIGCONT", "discard signal", "continue after stop" "20", "SIGCHLD", "discard signal", "child status has changed" "21", "SIGTTIN", "stop process", "background read attempted from control terminal" "22", "SIGTTOU", "stop process", "background write attempted to control terminal" "23", "SIGIO", "discard signal", "I/O is possible on a descriptor (see fcntl(2))" "24", "SIGXCPU", "terminate process", "cpu time limit exceeded (see setrlimit(2))" "25", "SIGXFSZ", "terminate process", "file size limit exceeded (see" setrlimit(2)) "26", "SIGVTALRM", "terminate process", "virtual time alarm (see setitimer(2))" "27", "SIGPROF", "terminate process", "profiling timer alarm (see setitimer(2))" "28", "SIGWINCH", "discard signal", "Window size change" "29", "SIGINFO", "discard signal", "status request from keyboard" "30", "SIGUSR1", "terminate process", "User defined signal 1" "31", "SIGUSR2", "terminate process", "User defined signal 2"

5.11 Default Optional Agruments

If you are needing to set an optional argument in a shell script, say if something has a default, therefore doesn't have to be included as an optional argument, you can use:

```
foo=${1:-foo}
```

Where `$1` is the position after the program name. So the `1` means "grab the item from position 1" and then `:foo` would be the defaulted option. As in "if no item in position 1, then use foo instead."

So, lets say:

```
portnumber=${1:-22}
```

That says "take the port number from position 1 OR if empty, use the number 22."

Here's the direct-from-manual info:

> If parameter is unset or null, the expansion of word is substituted. Otherwise, the value of parameter is substituted.

Here's the info if you omit the colon `:`

Omitting the colon results in a test only for a parameter that is unset. Put another way, if the colon is included, the operator tests for both parameter's existence and that its value is not null; if the colon is omitted, the operator tests only for existence.

CHAPTER 6

Git

This is not, in any way, meant to be exhaustive - or even highly detailed or maybe even very correct - guide on using your Git command line tools. This is, to be honest, mostly a means of a personal notebook to start. Though, I've found when I start it that way, it can turn into something better, different, than originally intended.

6.1 Local Git Repo Address

When you have a git repo locally downloaded on your machine, the comand line `git` program doesn't distinguish between you using Github, BitBucket, GitLab or your own creation as the remote repo. So, for this reason, if you ever decide to change where you host your repo, or the address you use to access that repo changes, its much easier than any proprietary system.

And, you can view that repo's saved address as well, in case you have forgotten or lost that.

6.1.1 Viewing the Repo's Address

First, while in your terminal program, you want to be sitting in the repo's directory to begin. I am assuming you know how to do that.

To view that git repo's address:

```
git remote -v
```

You will see, at minimum, two lines showing the fetch and push addresses, along with the branch name, usually origin.

```
git remote -v
  icefox      https://github.com/icefox/git-map.git (fetch)
  icefox      https://github.com/icefox/git-map.git (push)
  origin      https://github.com/JPCDI/git-map.git (fetch)
  origin      https://github.com/JPCDI/git-map.git (push)
```

6.1.2 Background

In that above code-block, you'll notice four lines, rather than 2. Along with an additional name other than `origin`. Thats because there is one remote location that I've named `icefox` and then my personal location named `origin`.

Origin is the defacto name `git` gives to cloned repos. You'll also see it referred to as `Master/Origin`. Because the Master is the remote location, with it being the Origin of the code. Or, at least thats my way of reconciling that info.

You are able to, say, pull from icefox, make a branch and changes, and rather than pushing or making a pull request for icefox, you can instead push out to origin.

6.1.3 Changing The Address

Now, if you are wanting to change the address to another http(s) or utilize an SSH address with your SSH keys, you are able and allowed to change that.

Now, onto the commands.

```
git remote set-url origin https://<remote-address>/
```

Each online git service has their own address layout specific to them. And, its obvious what those are as you begin using them.

6.1.4 Code Breakdown

First, `git` doesn't require the use of dashes, but rather allows empty spaces to delimit different options. So, don't freak out about the lack of – in these lines.

1. `git` - obviously this is calling the `git` program itself
2. `remote` - tells `git` that we are doing something specific with the remote info
3. `set-url` - This is telling `git` that we're setting the URL for the remote, either first-round setting, amending the current info, or adding another repo.
4. `origin` - This is the name you are giving to the repo. If you want something other than `origin`, you're welcome to but don't forget that.
5. `https://<remote-address>` - Obviously, this is the remote repo's address. The one you utilize will be longer, as most services break down that location between the given username/company name and the specific repo name.

6.1.5 Final Steps

Then, verify that the info has been updated with another:

```
git remote -v
```

This, rather than `git pull` or `git fetch --all` is a safer way to make sure that info is correct. If you put in the wrong address, then pull or fetch, it could remove your info thats local, and we all know the hinky stuff that arises then.

Once verified,

`git pull` or `git fetch --all`

The `--all` can also be amended to the repo name you gave it. So, if more than one repo address is included, you can fetch specific repo's rather than all of them.

6.2 Saving your Git Repo Password

When pushing back up to your own git repo - and you are using HTTPS for the address - it should ask for your password. And, if you have your username in the repo address, it'll bypass asking you for that info. But, you are able to cache your password; if you have fairly frequent pushes, and/or trust the system you're using.

Of course, depending on the system you are using, depends on how long that info is cached:

- If you're on a linux machine, there is a specific time you can program in for caching
- If you're on macOS, it caches that info up to your Keychain, therefore its accessible so long as you're logged into the machines account its saved to

These caching programs have the moniker `credential helpers`

6.2.1 OR

You can use the ssh version of the URL instead!:

```
git clone git@github.com:username/your-repo
git clone git@gitlab.com:username/your-repo
```

And that will utilize the SSH keys you have setup with those respective services, which you do have setup, right??? (You give them the public key, and then you can use your private key for authentication.)

6.2.2 macOS

In your macOS's terminal, to see if `osxkeychain helper` is already installed:

```
git credential-osxkeychain

usage: git credential-osxkeychain <get|store|erase>
```

If its not downloaded, it'll prompt you to retrieve the Xcode Command Line Tools that you have to have to do much of anything terminal-wise on macOS. That output will tell you what to do in that event.

Next, you have to tell `git`, through config files, to utilize the credential helper.

```
git config --global credential.helper osxkeychain
```

Now, the next time you try to do anything requiring that git password, it'll prompt you for the info, and then automatically save it in your keychain.

CHAPTER 7

Website Testing

7.1 Testing your Web Server Configuration

This is a running list of websites that will, well, test your website for different reasons, mostly security.

7.1.1 List

1. securityheaders.io an analyzer and rating system for your HTTP response headers
2. CFSSL "CFSSL is CloudFlare's PKI/TLS swiss army knife. It is both a command line tool and an HTTP API server for signing, verifying, and bundling TLS certificates."
3. Qualys SSL Labs This is the big behemoth of testing sites. They tend to be considered the defacto-standard for security testing.

CHAPTER 8

Extras

I'm calling this section "Extras" for two reasons, really:

1. Honestly unable to come up with a better name
2. These will be items that can fit into the generic Unix world, or are unable to be shoehorned into Ubuntu/Linux or macOS.

8.1 Random Ansible Snippets

My Pass a List snippet is probably the one reason I come back to my own docs site more than anything else, finding myself constantly forgetting how to properly pass a random list of objects - like a string of apps to install, on a whim of testing, across all three of my pi's at once - to ansible's CLI setup. As what I'll be doing isn't quite enough to facilitate a full play, as its not even tested yet!

8.1.1 Ansible CLI - Basic Example

Yes, I know this is like, the first thing that the Ansible Documentation site teaches you. But, again, it IS, like, the first thing this massive thing teaches you, and not in too much detail.

So, here's a repeat, refresher, with screenshots!!

```
ansible ubuntu -a 'free -m'
```

```
jpartain89 @ Justins-MacBook-Air on -bash: ~ $ ansible linux -a "free -m"
192.168.1.34 | SUCCESS | rc=0 >>
             total       used       free     shared    buffers     cached
Mem:           973        354        618          7         46        117
-/+ buffers/cache:        190        782
Swap:            0          0          0

192.168.1.20 | SUCCESS | rc=0 >>
             total       used       free     shared    buffers     cached
Mem:          1015        940         74          0         15        833
-/+ buffers/cache:         91        923
Swap:         1044          0       1044

192.168.1.30 | SUCCESS | rc=0 >>
             total       used       free     shared    buffers     cached
Mem:          2000       1231        769          0        179        650
-/+ buffers/cache:        401       1599
Swap:         2045          0       2045
```

This just runs ansible's `command` module. As in it doesn't support shell variables and things like piping. Thus, why the only flag given is the `-a` flag. If you wanted to change the module used, you'd include the `-m` flag before `-a`, as the `-a` flag tells ansible the text within the ' ' is the actual text for the module requested.

Next, we change the module to `ping`

```
ansible all -m ping
```

This doesn't require the `-a` flag, unless there are other options you want to include with the `ping` module. Otherwise, this will run a simple ping on the hosts you requested.

```
jpartain89 @ Justins-MacBook-Air on bash: ~/git $ ansible all -m ping
192.168.1.20 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
192.168.1.30 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
192.168.1.10 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
192.168.1.15 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
192.168.1.34 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh.",
    "unreachable": true
}
```

8.1.2 Pass a List

I've been wanting to figure this one out for forever!

Basically, how to pass multiple items into a command option when using ansible's `ad-hoc` mode on the command line.

```
ansible ipsec --become -m apt -a 'name={{ list }}' -e '{"list": [strongswan,
↪strongswan-plugin-eap-mschapv2,moreutils,iptables-persistent]}'
```

8.2 Ansible Random Nuggets

Setting a default to a variable, in order to test if its set or not:

```
when: secret_gmail_passwd|default(None) != None
```

This will allow the task to run if and only if the variable `secret_gmail_passwd` is NOT None.

8.3 Fixing mysql/mariaDB 767 Character Error Code

A seemingly super duper common error - that, for some reason, a lot of the "old hat" database users act like they've NEVER SEEN BEFORE and CAN'T EVER POSSIBLY REPRODUCE - that always is somehow difficult to find the CORRECT means of fixing, is going to be laid out here for a good spot to find it again....

```
ERROR 1071 ... : Specified key was too long; max key length is 767 bytes
```

Type that into Google and even Alice herself would throw in the towel....

Plenty of websites will halfway get you to the fix, and then complain that you have no idea how to fix it already....

Well, I'm not going to be doing that here today.... I'll be giving you the MAIN fix, along with the commands you can also run while inside mysql/mariaDB.

8.3.1 tl;dr

Place the below block into your `/etc/mysql/mariadb.conf.d/50-server.cnf`, within the `[mysqld]` section of the configuration file.

```
innodb_file_format = Barracuda
innodb_file_per_table = on
innodb_default_row_format = dynamic
innodb_large_prefix = 1
innodb_file_format_max = Barracuda
```

This way, these settings are persistent on the server. These are specifically for the default installation that has the `db charset` set to `utf8mb4`, table type of `InnoDB` and table charset of `utf8mb4_unicode_ci`.

You can also set them within the `mysql/mariadb` environment, but that seems as though it doesn't want to ever stay set this way.

```
SET GLOBAL innodb_file_format = Barracuda;
SET GLOBAL innodb_file_per_table = on
SET GLOBAL innodb_default_row_format = dynamic
SET GLOBAL innodb_large_prefix = 1
SET GLOBAL innodb_file_format_max = Barracuda
```

Note: Make sure you restart the sql server after changing any settings in the configuration files.

Bibliography

[HTPC-CP] These directions were liberally copied from HTPCGuides's CP-HowTo

[HTPC-Tautulli] These instructions were copied, mostly, from the original DrZoidberg33's GitHub Repo, but he no longer manages this, it seems. So, you can look at Tautulli's Github

[HTPC-Sonarr] These directions were liberally copied from HTPCGuides.com

[TRANS-HTPC] Copied from HTPC-Guides and StackExchanges Raspberry Pi Forum on Transmission Permissions

[TRANS-BlockList] Transmission BlockList from 'GiulioMacs Personal Blog'_

[PIA-VPN] Copied from the bottom half of Superjamies gist

[NZBGet-HTPC] Copied from HTPCGuides on NZBGet

[NZBGet-GitHub] NZBGet GitHub Page

[Usenet-Servers] Usenet Servers are the old-school Internet, more-or-less, before there was a real "internet." Usenet Wikipedia

[NGINX-Copied] These instructions are happily borrowed from ServersForHackers.com

[DDCLIENT-Source] These directions are liberally copied from DDClient's Github

[POSTFIX-HowTo] Copied very liberally from HowToForge Postfix How-To

[AFRUIT-USB] Adafruit's Raspberry Pi on USB

[RATOM-src] These directions are copied from randy3k/remote-atom