Install pyenv on Ubuntu and Debian - bgasparotto

Thanasisn github.com/thanasisn

2025-05-14

Contents

pdf file

For correction or contributions contact ThanasisN at github.com/thanasisn.

***Notice:* These notes are for my own use. They are opinionated and may contain any kind of errors.**

Intro

Here I am collecting my notes about deploying GNU/Linux systems. **This is mainly for my reference**, for setting up a system with my preferred setup and functionality. In general, I am guided by minimal and functional choices. I will go for simple and minimal solutions, but will not avoid bloating it, if it is easier, at least for a first approach solution. Many times it is easier to start with a GUI for new stuff and ditch it later, as you learn more.

Most stuff are tested and used on "Debian", but I have an eye for the "NixOS" way.

This is not trying to be a step by step guide. You can google a thousand tutorials for each subject.

If you find it useful, you can contribute to my coffee addiction, with a donation here:

A lot of my code lives here: github.com/thanasisn, along with Linux tools and research stuff.

As Bill† used to say …

> Before touching the keyboard, make a plan and design, what you are going to do with it.

0.1 Backup your precious data

0.1.1 First approach

Don't back up anything. You only live once and your data won't matter to the Universe.

0.1.2 Another approach

- **Backup everything**, your files, your responsibility.
- Verify your backups still work, regularly.

- Get things that you may need from all of your systems.
  - `/etc` config files
    - /etc/tinc
    - /etc/collectd
    - /var/lib/ganglia
    - /var/lib/collectd
    - /var/spool/cron
  - ssh keys from users
  - user profile files `.bashrc` `.history` etc.
  - lists of installed packages
  - other software installed manual
  - things that are not in your `$HOME`
  - files from 'root' user account
  - list of libraries from your programming languages
  - ....

Your backup should be automated, and the configuration collection scripted (example).

0.2 Distro hopping

Do it, and get over it. Use a test machine or a virtual machine to fool around, keep your main machine always available.

- Get you favorite distro. Keep it simple and robust with "Debian or Arch".
- Prefer net install media. We start with a minimal install and will build up.
- Check if you need firmware. After formatting is not easy to get to the net for answers and software.

0.3 Design and reasoning

0.3.1 For building a system, I want:

Fast setup: I tend to set up and format secondary machines regularly.

Reproducible environments: I like to have the same functionality on any machine I use.

Common things: Security, syncing, settings among the hosts.

Evolution: The ability, and the opportunity to improve my setups.

0.3.2 My systems usually are:

Main personal system: A laptop at home, everything starts from here.

Desktop system for work: Work horse and backup.

Grid/cluster computing account: Configs and programs copied there to increase functionality.

Server systems for work: Storage, compute nodes and more.

Secondary personal system: A laptop for working outside with extra security.

Home server: Mainly backup storage and cloud uploader.

Secondary and old machines: Media players, test machines …

So, these notes help me, and reminds me stuff to keep all this running and happening.

0.3.3 Current system goal

Root on an `btrfs` with raid 1: Two disk for redundancy end error correction checking against bit rot. Easier installation on Debian.

Home partition on encrypted `zfs` raid 1: Redundancy, error checking and safety. Not so safe as with full disk encryption but good enough. Now I am using a 'gocrytpfs' filesystem on top of a btrfs partition.

Auto login on boot without mounting users encrypted home: So I can ssh from my other machines and act as a honeypot in case of losing it.

Manual mounting/unmounting of users encrypted home: So, I have control of the data exposure. Use a different .ssh/key for the two states, so other application can work normally while encrypted (sync, etc).

0.3.4 Security and privacy

You should really consider some things about security and privacy.

Check that: wiki.debian.org/SetupGuides/SecurePersonalComputer

Also you should use email aliases, and pgp encryption, especially when using gmail and similar providers.

1 System Install

1.1 The `.iso`

- Grub the net installer for your favorite distro.
- Check for firmware dependencies.
- Use the magic of ventoy.net

1.2 Partitioning and file system.

Think about partitioning and file system you are going to use.

- is a multi-boot?
- want encrypted partitions?
- keep server functioning even with encrypted partition?

It is easier to set it or prepare it, at system install, especially if you want to make something complex later.

```
NAME          MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINT
sda             8:0    0 465.8G  0 disk
  sda1          8:1    0   953M  0 part  /boot
  sda2          8:2    0     1K  0 part
  sda3          8:3    0 421.1G  0 part
    crypthome 254:0    0   421G  0 crypt /home/athan
  sda5          8:5    0    14G  0 part  /
  sda6          8:6    0    14G  0 part  /home/arch
  sda7          8:7    0   1.9G  0 part  [SWAP]
```

Here is my old scheme. Dual boot with a spare partition and an encrypted home partition, all on `btrfs` file system.

1.3 Install the basics

Install the most basic options your distro provides.

- No desktop environment
- No GUI apps

Start just with a terminal and an ssh server, and get only the things you need. Usually I have a minimal DE like `lxde` for fail back or guest users, I use `dwm` as window manager, and mix GUI apps from different desktops. Of course the looks of the windows become inconsistent, but you can configure your way out of this if you want.

I try to purge any program I don't need or stopped using.

2 System configuration

Mostly based on `debian`.

2.1 System mail over gmail with `postfix`

i.e. I couldn't make `exif4` work for me

2.1.1 Prepare

Issue a unique password from https://myaccount.google.com/apppasswords to use for each machine.

2.1.2 Install

```
sudo apt-get install libsasl2-modules postfix
```

2.1.3 Configure

Choose:

```
Internet Site
```

Edit:

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

```
 . . . . .
## add
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
 . . . . .
## edit
```

```
relayhost = [smtp.gmail.com]:587
 . . . . .
```

Create password file:

```
sudo vim /etc/postfix/sasl_passwd
```

```
[smtp.gmail.com]:587    here_is_your_mail@gmail.com:pasw0rd_from_google_app_unique_passwords
```

Prepare password for use:

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

Edit system aliases:

```
sudo vim /etc/aliases
```

Add to whom the mail is going.

Restart services:

```
sudo newaliases
sudo systemctl restart postfix
```

2.2 `collectd`

I use this to monitor multiple machines, and have graphs of all on my `conky`.

2.2.1 Install

```
apt-get install lm-sensors hddtemp collectd ntp
```

2.2.2 Configure

```
sudo /lib/systemd/systemd-sysv-install enable hddtemp
sudo sensors-detect
sudo vim etc/collectd.conf
```

2.2.3 Starting the daemon

You need to restart the daemon.

```
sudo /etc/init.d/collectd restart
```

or

```
sudo systemctl enable collectd.service
sudo systemctl restart collectd.service
```

For `hddtemp` have to enable daemon and start service

```
sudo vim /etc/default/hddtemp
sudo systemctl enable hddtemp.service
```

2.3 `ntp`

Keep your clock in sync

```
## enable
sudo systemctl enable ntp
## start running
sudo systemctl restart ntp
## check
ntpq -p
```

2.4 VPN network

If you have more than one machine, **network all devices in a private VPN**. I use `tinc`.

It is easier if you have available a machine with static IP. Otherwise, you can use dynamic DNS services to do that.

Having all your devices able to talk to each other makes everything easier. You can then, easily setup ssh, backup, nfs, file sync and much more. You can access hosts even with dynamic IP and behind firewalls and NAT.

2.5 Setup `.ssh/config`

Use the `.ssh/config` file to add all your hosts and all their IPs public or private. This way, you don't have to write the addresses or give the identity file each time. And also, this will work for many other applications that use ssh configuration.

2.6 Set `ssh` keys

```
ssh-keygen -t dsa
chmod 600 .ssh/id_dsa
scp .ssh/id_dsa.pub dude@remote.edu:
cat id_dsa.pub >> .ssh/authorized_keys
chmod 700 .ssh
```

2.7 Keep your files and config in sync

Using `unison` to sync config files, so all machines feels the same.

I select with files are synced based on each machine's role, work, personal, testing…

That doesn't count as backup. There are cases that I have messed thing up with it.

2.8 Install and configure `zsh`

```
sudo apt-get install zsh
```

Get oh-my-zsh from its repo.

github.com/robbyrussell/oh-my-zsh

Change default shell

```
chsh -s /usr/bin/zsh
```

2.8.1 change shell

```
chsh -s /usr/bin/zsh
```

3 System Backup

Your backup should be automated! human errors and reliability is a constant problem for computers. If you think, copying data by hand to a USB drive is a backup, you are wrong. **COPYING DATA BY HAND TO A USB DRIVE IS POINTLESS**. It will be lost, neglected and fail at some point, sooner than you expect.

Testing your backup should also be automated! You have to know it works, not assume it.

Constantly log and report on the backup status! You want to know if your backup has failed, now. Not three months before you need it.

I have a script that shows on `conky` the status of each backup, and counts how long before was the last successful backup, last warning and last fail. Also monitor the size of each as an indication of things going good. Color coded on `conky` so if I see

- white, everything is normal don't bother to look
- green, something is out of the ordinary but still good
- yellow, a problem has emerged, may be in danger, take action
- red, this is off, consider it not safe

3.1 Software

Backup programs, I have extensively used for years on multiple machines at the past:

- `deja-dup` too long ago to remember
- `backintime` elementary but good
- `bacula` excellent

Current program **`borg`**. Deduplication, encryption, compression, snapshots everything I need and simple. Upload the whole backup data to the cloud with **`rclone`**.

Some custom `bash` scripts on `cron` do everything, backup, prune, test, upload, report.

3.2 Current scheme

- Local backup
  - One `raid` 1 array for high frequency sort-term retention (`borg` stable version)
  - One `btrfs` raid1 array for low frequency midterm retention (`borg` testing version)
  - An external drive for occasional easy portable backup (`borg` testing version)
- Remote backup
  - One on a private remote machine (`borg` stable version)
- Cloud
  - Sync a backup copy from local to google drives (`rclone`)
  - Sync a new backup copy every month from remote to google drive (`rclone`)

This is for backing up my personal main machine. Occasionally, I have a local backup with `borg` for some of the secondary machines. These include config files or a lot of data, that can be recovered with other means but will take time and bandwidth.

4 Random stuff, tricks and tools

4.1 ddresque

Rescue failing partitions to a new disk or to an image file with `ddresque`.

4.1.1 First pass

Do a "fast" copy ignoring hard problems.

```
ddrescue -v -f -n -a 5120000 /dev/sda /dev/sdc logfile.log
```

- -n keeps `ddrescue` from "scraping", that puts a heavy load on the drive trying to rescue the most difficult parts
- -a 5120000 it will skip any area that cannot be copied at a speed of at lest 5 megabytes per second
- logfile.log a file to use, so you can start over from where you left off rather than starting from start
- -f force overwrite of outfile, needed when outfile is not a regular file, but a device or partition
- -v verbose

4.1.2 Second pass

Do a more intensive try to copy.

```
ddrescue -v -f -d -r 3 /dev/sda /dev/sdc logfile.log
```

- -d direct disk access
- -r 3 read unrecoverable sectors 3 times
- -f force overwrite of outfile, needed when outfile is not a regular file, but a device or partition
- -v verbose

4.2 `dd` and `gzip` img file

4.2.1 `dd` disk to compressed image

```

dd if=/dev/sdx | gzip > /path/to/image.gz
```

4.2.2 `dd` compressed image to disk

```

gzip -dc /path/to/image.gz | dd of=/dev/sdx
```

4.3 Polar Pro Trainer 5 and IRDA

4.3.1 Polar Pro Trainer 5

This is a 32 bit program, so we need `wine32`.

```
## install wine
sudo dpkg --add-architecture i386 && sudo apt update
sudo apt-get install wine32
## setup wine
export WINEARCH=win32
export WINEPREFIX=~/.wine32
winecfg
## run program setup
wine setup.exe
```

You can update the program with overwriting newer files.

4.3.2 Setup IRDA probe

```
sudo apt-get install irda-utils
```

For configuring ma-620 there is a bash script: ma-620_setup.sh.

For polar USB dongle setup

```

alias irda0 mcs7780
```

4.3.2.1 `/etc/modprobe.d/irda-utils.conf`

```

ENABLE="true"
DEVICE="irda0"
MAX_BAUD_RATE="9600"
```

4.3.2.2 `/etc/default/irda-utils`

4.4 OCR

4.4.1 `ocrfeeder`

- using `tesseract`

4.4.2 `gimagereader`

- easier and prettier than `ocrfeeder`
- using `tesseract`

4.4.3 `ocrmypdf`

- nice
- CLI

```
ocrmypdf -l eng+ell --rotate-pages --deskew
ocrmypdf -l eng+ell --rotate-pages --deskew --clean --remove-background
```

Usually I want to use this:

```
ocrmypdf --jpeg-quality 100 --png-quality 100 --optimize 0 -l eng+ell --rotate-pages --deske
```

Part of man page…

```
--optimize {0,1,2,3}
              Control  how PDF is optimized after processing:
              0 - do not optimize;
              1 - do safe, lossless optimizations (default);
              2 - do some lossy optimizations;
              3 - do aggressive lossy optimizations (including lossy JBIG2)
--deskew
              Deskew each page before performing OCR
--clean
              Clean  pages  from  scanning artifacts before performing OCR,
              and send the cleaned page to OCR, but do not include the
              cleaned page in the output
```

4.5 Spell check word list

I hard link all the files of the same language so

- All are updated instantly with new words
- I can remove words or edit all file together
- Use common dict for all programs

Found linked files with: `find -L ./ -samefile .vim/spell/el.utf-8.add`

```
./.config/enchant/el.dic
./.config/enchant/el_GR.dic
./.hunspell_el_GR
./.vim/bundle/vim-grammarous/misc/LanguageTool-5.5/org/languagetool/resource/el/hunspell/spe
./.vim/spell/el.utf-8.add

./.config/enchant/en.dic
./.config/enchant/en_GB.dic
./.config/enchant/en_US.dic
./.hunspell_en_US
./.vim/bundle/vim-grammarous/misc/LanguageTool-5.5/org/languagetool/resource/en/hunspell/spe
./.vim/spell/en.utf-8.add
```

- Some programs I use like `Rstudio` and `texstudio` are missing.
- `vim` doesn't add words to the correct language.

4.6 Mount physical HDD on a VirtualBox machine

You can do a fast OS installation through a virtual machine on a physical hard drive.

```
sudo VBoxManage internalcommands createrawvmdk -filename "/path/to/file.vmdk" -rawdisk /dev/
sudo chmod 777 /dev/sda
sudo chmod 777 /path/to/file.vmdk
```

4.7 Ignore lid close on laptops

Edit: `/etc/systemd/logind.conf`

```
.........
HandleLidSwitch=ignore
.........
```

```
sudo service systemd-logind restart
```

5 Software Selection Reasoning

A reminder to myself of which programs I have tried, and why I use them.

5.1 Mail client

Tested: `thunderbird`, `kontact` and `evolution`.

I have used them all for a long time with multiple accounts. I end up with `evolution`. As long as I have `seahorse` unlocked everything works fine. The other two are good programs but the synchronization of options across machines is not so straightforward.

5.2 File browser

Tested: a bunch of the most prominent.

End up working with terminal, `caja` and `lf`, and occasional `dolphin` and `mc`. All are mapped to keyboard keys with `sxhkd`, so I start with the most appropriate, for what I have in mind.

5.3 General text editing

Nothing beats `vim`. Learn now, thank me later.

Regularly, I also use `kate` and sometimes `geany`.

6 Commands I forget

Less often used command that are very handy, but I tend to forget due to infrequent usage.

6.1 Network monitoring

`nethogs` per program network bandwidth monitor
`iftop -i any`: per connection bandwidth and volume monitor

6.2 System monitoring

`htop` general system monitoring
`bpytop` general system monitoring/manager
`glances` general system monitoring

6.3 Testing

- `systemctl enable ssh.service`
- `systemctl start ssh.service`
- `PasswordAuthentication yes`
- `sudo adduser bob`
- `sudo passwd`
- `vim /etc/hostname`
- `vim /etc/hosts`
- `visudo`
- `sudo vim /etc/lightdm/lightdm.conf`
- `sudo apt install tinc`

7 Configurations

Some configuration I use regular or have used on the past.

7.1 btrfs

I use btrfs mostly to it's flexibility. Mostly when I foresee the need to make changes on the partitions or the disks.

7.1.1 Break a mirror

The functionality of `btrfs` continues to evolve over time. The `delete` command is now an alias for the `remove` command which produces different results. Today, you would run the following commands:

```
btrfs balance start -f -sconvert=single -mconvert=single -dconvert=single <mount>
btrfs device remove <drive> <mount>
```

The first command converts all data from a mirrored setup to a single-copy setup. This effectively makes the RAID1 into a JBOD setup. The `-f` option is required to tell the filesystem to *really* reduce the resiliency of the data.

Once this completes, the second command removes the device from the JBOD. The filesystem will move any data from the removed device to the other device.

7.1.2 Rebalance to reduce unallocated space

Avoid full balance

```
 sudo btrfs balalance start -dusage=40 /
 sudo btrfs balalance start -dusage=25 /media/maindata
```

Use for new disk on raid

```
 sudo btrfs balance start /
 sudo btrfs balance start /media/maindata
```

Watch the balance progress

```
 sudo watch "btrfs filesystem usage -T /; btrfs balance status /"
 sudo watch "btrfs filesystem usage -T /media/maindata; btrfs balance status /media/maindata
```

Info on the filesystem

```
sudo btrfs device stats /dev/sda4
sudo btrfs device stats /dev/sda5
```

7.1.3 Just format two disks into raid1 with:

```
sudo mkfs.btrfs -m raid1 -d raid1 /dev/sdb /dev/sdc
```

7.1.4 Add new disk to raid1

List existing file systems

```
sudo btrfs filesystem show
```

```
Label: 'barelfs'  uuid: bd59a77e-a39c-4dad-885c-d956bc67d12b
    Total devices 1 FS bytes used 1.16TiB
    devid    1 size 1.77TiB used 1.16TiB path /dev/sdd2
```

Add disk to the filesystem

```
sudo btrfs device add -f /dev/sde2 /media/barel/
```

Start balancing to actually create the raid mirror

```
sudo btrfs balance start -dconvert=raid1 -mconvert=raid1 /media/barel/
```

7.2 zfs

I use ZFS mostly for the home folder to use native encryption

```
zpool create  -o atime=off -o encryption=on -o keyformat=passphrase poolname drive
```

Use native encryption

```
zpool create  -O atime=off -O encryption=on -O keyformat=passphrase poolname mirror drive1 d

zpool attach poolname existinghdd newhdd

zpool status poolname

sudo zpool create -f test mirror /home/athan/ZHOST/4G.img /home/athan/ZHOST/3G.img
```

7.2.1 from a pool history

```
zpool create maindata raidz1 /dev/sdb3 /dev/sda2
zfs set atime=off maindata
zfs create -o encryption=on -o keyformat=passphrase maindata/home_store
zfs set mountpoint=/home/ maindata/home_store
zpool import -c /etc/zfs/zpool.cache -aN
zfs load-key -r maindata/home_store
zfs create -o mountpoint=/nix maindata/nix_store
zfs set atime=off maindata
zpool set autotrim=on maindata
zfs set xattr=sa maindata
zpool import -c /etc/zfs/zpool.cache -aN
zfs load-key -r maindata/home_store
```

7.3 Transparent Encryption For a User's Home Folder

```
sudo systemctl enable tinc@cosmos
```

First install the packages ecryptfs-utils and rsync:

```
sudo apt-get install ecryptfs-utils rsync lsof
```

Then load the ecryptfs kernel module:

```
sudo modprobe ecryptfs
```

And make it permanent in /etc/modules-load.d/modules.conf.

The user whose home directory you want to encrypt MUST NOT be logged in. For example, you can be logged as root in a tty.

Then run as root:

```
ecryptfs-migrate-home -u <username>
```

When this is done the user must login BEFORE rebooting the computer.

Assisted Encrypted Swap Partition

To encrypt the swap partition too:

```
sudo apt-get install cryptsetup
sudo ecryptfs-setup-swap
```

7.4 home encryption with `cryptsetup`

This is viable and similar to using zfs native encryption on a dataset.

- user data are encrypted
- user profile accessible over ssh
- can crypt/decrypt and mount over ssh
- other services can ran while encrypted
- Install repos on partitions
- use a common partition as `/home/`
- encrypt partition with cryptsetup
- create fs in the encrypted side
- mounted over root `/home/`
- have some config keys and scripts on both `$HOME`
- ask to mount cryptsetup when logging in
- should work also without decrypt/mount for access
- can even auto login a guest user or `$USER`
- may link original `/home` to another location for access

7.5 Encrypted Home directories + SSH Key Authentication

Howtos & bash encrypted-home-dir ecryptfs ssh

This is similar to other other approaches for home encryption. The setup of ssh keys can be also done to the others.

There is an interesting it-makes-sense-when-you-think-about-it issue with Encrypted Home directories and SSH key authentication I've recently discovered in Ubuntu (it will affect any distro though). Since Encrypted home directories aren't decrypted until the login is successful, and your SSH keys are stored in your home directory, the first SSH connection you make will require a password. If you have password authentication turned off, you'll have big issues.

I found a question on Super User which explains solutions to this issue. My solution follows closely on this answer, although I have added in a symbolic link to make it easier to manage.

1. Create .ssh folder in /home for the keys to be stored

```
sudo mkdir /home/.ssh
```

2. Move existing authorized_keys file into .ssh dir as username

```
sudo mv ~/.ssh/authorized_keys /home/.ssh/username
```

3. Create symbolic link to authorized_keys file in user .ssh dir

```
ln -s /home/.ssh/username ~/.ssh/authorized_keys
```

4. Update sshd config file to set the new path for the authorized_keys file

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

```
# Change the AuthorizedKeysFile line to:
 AuthorizedKeysFile /home/.ssh/%u
```

5. Reboot the computer

```
sudo shutdown -r now
```

6. Login to your server and you should be presented with a minimal un-decrypted home directory... You will need to create and edit a .profile file in there to get ecryptfs to mount your home directory.

```
sudo vim ~/.profile
```

```
Add these lines:
 ecryptfs-mount-private
 cd /home/username
```

7. Log out/Restart, and go back in again. You should be promoted for your password after SSH key auth, and then be presented with your decrypted home directory.

You should now be able to login using SSH keys every time, no matter if your home dir is decrypted or not :-)

8 General usage notes

8.1 Monitor programs usage

```
apt install acct       # Install the package
accton on              # Enable accounting to /var/log/account/pacct
```

Then you can view the commands executed

```
lastcomm               # All commands
lastcomm -u userid     # Processes executed by {userid}
```

Summarizes accounting information under Linux

```
sudo sa
```

Popularity contest

```
sudo apt install popularity-contest
sudo popcon-largest-unused
```

8.2 git unusual usage

8.2.1 List all files in a repo

```
git log --pretty=format: --name-only --diff-filter=A | sort -u
```

8.2.2 Remove a file from the repo

Will delete history and existing files

```
git filter-branch -f --prune-empty --tag-name-filter cat --index-filter "git rm -rf --cachec
```

8.2.3 Worktrees

8.2.3.1 Create new worktree

```
git worktree add -b test_build ./test
```

8.2.3.2 Remove worktree folder

```
git worktree remove ./test
```

8.2.4 Remove branch

```
git branch --delete test_build
```

8.2.5 Remove remote branch

```
git push origin --delete test_build
```

8.2.6 List deleted files from repo

```
git log --diff-filter=D --summary
git log --diff-filter=D --summary | grep "delete"
```

8.2.7 Move from git to git

http://gbayer.com/development/moving-files-from-one-git-repository-to-another-preserving-history/

8.2.7.1 Moving Files from one Git Repository to Another, Preserving History If you use multiple git repositories, it's only a matter of time until you'll want to refactor some files from one project to another. Today at Pulse we reached the point where it was time to split up a very large repository that was starting to be used for too many different sub-projects.

After reading some suggested approaches, I spent more time than I would have liked fighting with Git to actually make it happen. In the hopes of helping someone else avoid the same trouble, here's the solution that ended up working best. The solution is primarily based on ebneter's excellent question on Stack Overflow.

Another solution is Linus Torvald's "The coolest merge, EVER!" Unfortunately, his approach seems to require more manual fiddling than I would like and results in a repository with two roots. I don't completely understand the implications of this, so I opted for something more like a standard merge.

8.2.7.2 **Goal**:

- Move directory 1 from Git repository A to Git repository B.

8.2.7.3 **Constraints**:

- Git repository A contains other directories that we don't want to move.
- We'd like to

Preserve the Git commit history for the directory we are moving.

8.2.7.4 Get files ready for the move: Make a copy of repository A so you can mess with it without worrying about mistakes too much. It's also a good idea to delete the link to the original repository to avoid accidentally making any remote changes (line 3). Line 4 is the critical step here. It goes through your history and files, removing anything that is not in directory 1. The result is the contents of directory 1 spewed out into to the base of repository A. You probably want to import these files into repository B within a directory, so move them into one now (lines 5/6). Commit your changes and we're ready to merge these files into the new repository.

```
git clone <git repository A url>
cd <git repository A directory>
git remote rm origin
git filter-branch --subdirectory-filter <directory 1> -- --all
mkdir <directory 1>
mv * <directory 1>
git add .
git commit
```

8.2.7.5 Merge files into new repository: Make a copy of repository B if you don't have one already. On line 3, you'll create a remote connection to repository A as a branch in repository B. Then simply pull from this branch (containing only the directory you want to move) into repository B. The pull copies both files and history. Note: You can use a merge instead of a pull, but pull worked better for me. Finally, you probably want to clean up a bit by removing the remote connection to repository A. Commit and you're all set.

```
git clone <git repository B url>
cd <git repository B directory>
git remote add repo-A-branch <git repository A directory>
git pull repo-A-branch master --allow-unrelated-histories
git remote rm repo-A-branch
```

8.3 Install 'pyenv' on Ubuntu and Debian

8.3.1 Install dependencies

```
sudo apt-get install -y make build-essential libssl-dev zlib1g-dev libbz2-dev libreadline-de
```

8.3.2 Install pyenv

```
curl -L https://raw.githubusercontent.com/pyenv/pyenv-installer/master/bin/pyenv-installer |
```

8.3.3 Validate installation

The command below should print the installed version:

```
pyenv -v
```

which in my case printed:

```
pyenv 2.1.0
```

If it doesn't, then add the following lines at the end of your:

- `.bashrc` if you use bash
- `.zshrc` if you use zsh

```
export PYENV\_ROOT="$HOME/.pyenv"
export PATH="$PYENV\_ROOT/bin:$PATH"
if command -v pyenv 1>/dev/null 2>&1; then
 eval "$(pyenv init --path)"
fi
eval "$(pyenv virtualenv-init -)"
```

Then reload your shell and it should work.

8.3.4 Installing Python versions

Run the command below to list the available Python versions to be installed:

```
pyenv install --list
```

Which should yield something like:

Available versions:

```
  2.1.3
  2.2.3
  ...
  3.10.0
  3.10-dev
  3.11.0a1
  3.11-dev
```

Then, pick a version of your choice and install it running the following command:

```
pyenv install 3.13.2
```

Finally, set the installed version as the default local (to this terminal session) or global (entire system):

```
pyenv local 3.13.2
```

8.3.5 Install modules

After that and maybe after a install `pip` works

```
pip install ephem
```