A pedantic guide to get yourself up and running with Python virtual (conda) environments.

Created	@October 5, 2022 9:51 PM
Publishable	☑
Property	

This is written for mainstream linux OSes. If you're using something edgy, YMMV. If you're on windows, I can do no better than to suggest opening up this page.

Index

0. A super opinionated set-up for ✨new machines ✨

Did you just get access to a new server? Maybe your cat peed on your laptop and you had to get a new hard disk [0] ? Do you like to be told how to behave and have no problem submitting to authority?

Then this section is for you!

[OPTIONAL] Setting some directories up

1. You want two directories on your system. One to store 'permanent' stuff, one which is primarily used for 'work'. Let's start there. (**DO NOT COPY THE PRECEEDING `$` in the commands below)**

```
$ cd # to make sure you get to your home direcory from wherever you may be
$ mkdir perm
$ mkdir perm/z # just trust me on this (https://github.com/rupa/z)
$ mkdir work # where you'll be cloning your git repos and doing nasty things with them
$ mkdir temp # where you'll be downloading stuff temporarily and forgetting to delete them till the end of this civlization.
$ tree .
.
├── perm
│   └── z
├── public
│   └── README.txt
├── temp
└── work

5 directories, 1 file
$
```

[OPTIONAL] Install z (faster jumping around).

See z in action.

1. Assuming you created a folder for it in step 1, follow along blindly. If not, change the paths as needed.

```
$ wget https://raw.githubusercontent.com/rupa/z/master/z.sh -O ~/perm/z.sh
```

Further install instructions are here (around the time you're setting up your bash configurations)

[OPTIONAL] Now let's give your bash (primary way of interacting with the system) a few superpowers.

> 💡 If you're using z.sh or some other fancy shmancy shell, I'm not sure how many of these commands will work. Probably all of them. If you don't know what you're doing,
> a. Why are you using zsh again, you fancy pants and
> b. maybe you should skip this part? 🤷

If you're still here, you should copy paste this following to your .bashrc file. That is, the file accessed by `~/.bashrc`. Use your favorite text editor 🤦 (unless it's emacs. Again, if you're already using emacs you probably don't need this, go back to the cave where you came from). If the file doesn't exist (it probably wont on Grid5000 servers), create it.

```
# append to the history file, don't overwrite it
shopt -s histappend

# make less more friendly for non-text input files, see lesspipe(1)
[ -x /usr/bin/lesspipe ] && eval "$(SHELL=/bin/sh lesspipe)"

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
```

Next, setting up z.sh (see this block). Copy paste this into your `.bashrc` wherever (at the end should be fine).

```
# Enabling z sh
. ~/perm/z/z.sh  # Replace this dir with wherever you chose to download the shell file if you changed the dir
export _Z_DATA=$HOME/perm/z/z.data # z maintains a list of directories visited. It clutters the homedir which is not something I generally
```

Next, some stuff corresponding to specific NLP Libraries. Again, this is super optional.

> 💡 If you have storage access on Grid5000, you probably want to move these ones there. The last one specially can get storage intensive down the line.

```
export NLTK_DATA=~/perm/nltk/nltk_data
export CORENLP_HOME=~/perm/corenlp
export HF_HOME=~/perm/huggingface
```

[OPTIONAL] ALIAS TIME!

These things should be put into a new file (or appended to, if it already exists) called `.bash_aliases`

```
alias l='ls -lah'
alias la='ls -lAh'
alias ls='ls -G'
alias lsa='ls -lah'
alias cp='cp -iv'                           # Preferred 'cp' implementation
alias mv='mv -iv'                           # Preferred 'mv' implementation
alias mkdir='mkdir -pv'                     # Preferred 'mkdir' implementation
alias ll='ls -FGlAhp'                       # Preferred 'ls' implementation
alias less='less -FSRXc'                    # Preferred 'less' implementation
cd() { builtin cd "$@"; ls; }               # Always list directory contents upon 'cd'
alias cd..='cd ../'                         # Go back 1 directory level (for fast typers)
alias ..='cd ../'                           # Go back 1 directory level
alias ...='cd ../../'                       # Go back 2 directory levels
alias ....='cd ../../../'                       # Go back 2 directory levels
alias .3='cd ../../../'                     # Go back 3 directory levels
alias .4='cd ../../../../'                  # Go back 4 directory levels
alias .5='cd ../../../../../'               # Go back 5 directory levels
alias .6='cd ../../../../../../'            # Go back 6 directory levels
alias ~="cd ~"                              # ~:            Go Home
alias which='type -all'                     # which:        Find executables
alias path='echo -e ${PATH//:/\\n}'         # path:         Echo all executable Paths
alias show_options='shopt'                  # Show_options: display bash options settings
alias fix_stty='stty sane'                  # fix_stty:     Restore terminal settings when screwed up
alias cic='set completion-ignore-case On'   # cic:          Make tab-completion case-insensitive
alias loc="find . -name '*.py' | xargs wc -l"

#   lr:  Full Recursive Directory Listing
#   ------------------------------------------
alias lr='ls -R | grep ":$" | sed -e '\''s/:$//'\'' -e '\''s/[^-][^\/]*\//--/g'\'' -e '\''s/^/   /'\'' -e '\''s/-/|/'\'' | less'e

#   extract:  Extract most know archives with one command
#   ---------------------------------------------------------
extract () {
    if [ -f $1 ] ; then
      case $1 in
        *.tar.bz2)   tar xjf $1     ;;
        *.tar.gz)    tar xzf $1     ;;
        *.bz2)       bunzip2 $1     ;;
        *.rar)       unrar e $1     ;;
        *.gz)        gunzip $1      ;;
        *.tar)       tar xf $1      ;;
        *.tbz2)      tar xjf $1     ;;
        *.tgz)       tar xzf $1     ;;
        *.zip)       unzip $1       ;;
        *.Z)         uncompress $1  ;;
        *.7z)        7z x $1        ;;
        *)     echo "'$1' cannot be extracted via extract()" ;;
         esac
     else
         echo "'$1' is not a valid file"
     fi
}

# enable color support of ls and also add handy aliases
if [ -x /usr/bin/dircolors ]; then
    test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -b)"
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'

    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
    alias egrep='egrep --color=auto'
fi
```

1. What are Conda virtual environments and why do we need them?

Python is a tricky thing to get set up correctly. Most operating systems come with a ready-made Python interpreter, typically

installed in `usr/bin/python`. Your terminal defaults to this when you execute a command `python`. Example →

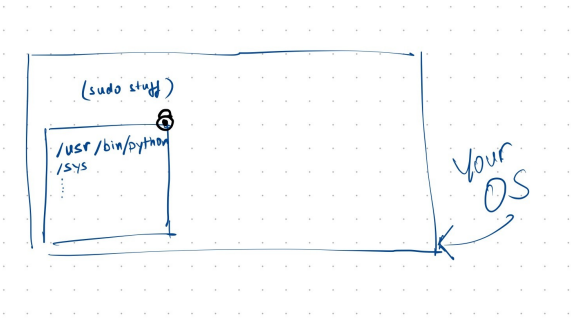
```
priyansh@priyansh-ele:~$ which python
python is /usr/bin/python
priyansh@priyansh-ele:~$
```

We don't want to use this interpreter.

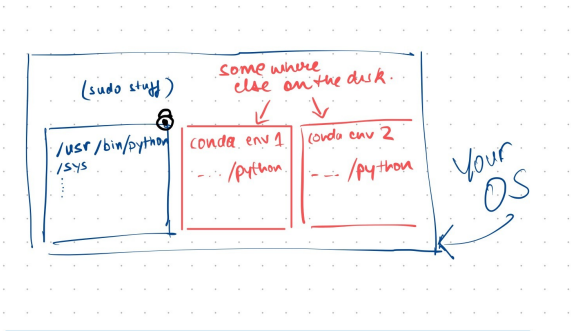
Why?

1. Installing libraries requires admin access.
   That is, to install something, you need to type `sudo pip install my-torch`, not just `pip install my-torch`.
   a. With sudo access given to a command which essentially runs arbitrary scripts, the rock bottom is really really deep. A malicious [1] [2] package (from official package repository - PyPi) can really mess with your system. So when we make virtual environments, the potential of damage reduces slightly.
2. Python libraries are associated with interpreters. So, if there is only one `python` there is only one version of the library. Imagine you're working on a project where you require a specific version of scikit-learn [3]. Imagine you're working on another project where you need another version of it. There's no way of handling this situation.

So what are we trying to do here?



This is your (probaly) current situation.



We want to turn it into this.

In this way, we can have as many 'virtual' python set ups as we want. They will not be stored in /sys or /usr so you can fiddle with them as you please (without admin access). You can install multiple versions of python libraries. Heck you can install multiple versions of Python if you so please.

Anyway, so how do we do it?

2. Installation

1. Get the miniconda installer from this page. Most likely you're looking for the `Miniconda3 Linux 64-bit` version.
   PS: if you're doing this on a GUI-less machine (like a server), you can just wget the link. That is:

   ```
   wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -P temp/
   chmod +x ./temp/Miniconda3-latest-Linux-x86_64.sh
   ```

   This will download the shell file to install conda environments and make it executable.
2. Next, execute the file.

   ```
   ./temp/Miniconda3-latest-Linux-x86_64.sh
   ```

1. You will then read the license (press spacebar to scroll faster)
2. `Do you accept the license terms? [yes|no]` ... I mean, you probably want to agree so `yes`.
3. Specify the location where you want this installed. **Highly recommend that you put in `~/perm/conda`.**

```
Do you accept the license terms? [yes|no]
[no] >>> yes

Miniconda3 will now be installed into this location:
/home/ptrivedi/miniconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/home/ptrivedi/miniconda3] >>> ~/perm/conda
```

4. Do you want to run conda init? **Highly recommended that you put in `yes` !!!!**

And you should be good to go. Its a good idea to check at this point if something went wrong or not. For that, try activating conda base by refreshing your bash config. `source ~/.bashrc`

There should now be a prefix `(base)` on your prompt. That is, it should look like:

3. New Environments

We can now start making virtual python environments. This part's simple.

1. Decide which **python version** you want to install (as of @October 5, 2022), its a good idea to stick to version 3.8. Decide also a name for your environment. I usually go with `main`.

```
(base) ptrivedi@fgrenoble:~$ conda create -n main python=3.8
```

2. Activate your new environment and **verify** if things are alright. (the prefix should reflect the new virtual environment name)

```
(base) ptrivedi@fgrenoble:~$ conda activate main
(main) ptrivedi@fgrenoble:~$
```

   Now let's see what interpreter do we refer to, when we invoke `python`:

```
(main) ptrivedi@fgrenoble:~$ which python
/home/ptrivedi/perm/conda/envs/main/bin/python
(main) ptrivedi@fgrenoble:~$ which pip
/home/ptrivedi/perm/conda/envs/main/bin/pip
(main) ptrivedi@fgrenoble:~$ which python3
/home/ptrivedi/perm/conda/envs/main/bin/python3
(main) ptrivedi@fgrenoble:~$ which pip3
/home/ptrivedi/perm/conda/envs/main/bin/pip3
```

   That's it! This tell us that unlike previously, now this particular file (in my case: `/home/ptrivedi/perm/conda/envs/main/bin/python`) is used when we call `python`.

> 💡 You can see here that `python` and `python3` are **different commands but point to the same environment.** Same for `pip` and `pip3`. We will stick to invoking them by simply `python` and `pip` respectively.

> 💡 **AFTER THIS POINT, ALWAYS ASSUME WE'RE INSIDE AN ACTIVATED CONDA ENVIRONMENT UNLESS EXPLICITLY STATED OTHERWISE.**
> That is, your bash is prefixed with `(<your environment name>)`)

3. Let's install a few essentials

```
$ pip install numpy scikit-learn scipy ipython jupyter tqdm termcolor
```

Verify here that your ipython & jupyter is installed properly.

```
(main) ptrivedi@fgrenoble:~$ which ipython
/home/ptrivedi/perm/conda/envs/main/bin/ipython
(main) ptrivedi@fgrenoble:~$ which jupyter
/home/ptrivedi/perm/conda/envs/main/bin/jupyter
(main) ptrivedi@fgrenoble:~$ ipython
Python 3.8.13 (default, Mar 28 2022, 11:38:47)
Type 'copyright', 'credits' or 'license' for more information
IPython 8.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

If your outputs look similar to mine, we're good to go.

4. [OPTIONAL] Let's make getting to our conda environment easy.

In your `~/.bash_aliases` file, append

```
alias d="conda activate main" # or whatever your 'default' conda environment is called
alias py='d && ipython'
alias jpy='d && jupyter notebook --no-browser'
```

And now you can get to a nice crisp ipython interpreter inside your new steaming hot conda environment with one click, right after log in, i.e. `py`

---

voilà

🤭

References

[0] Good on the cat, sustaining our multibillionaire tech overlords by making you burn a bigger hole in your pocket./

[1] https://arstechnica.com/information-technology/2021/11/malware-downloaded-from-pypi-41000-times-was-surprisingly-stealthy/

[2] https://jfrog.com/blog/malicious-pypi-packages-stealing-credit-cards-injecting-code/

[3] A very real situation I'm facing due to a very minor change in the library - https://stackoverflow.com/questions/62390517/no-module-named-sklearn-utils-linear-assignment

[4] CUDA is a programming language or an API which enables CPU code to talk to the GPU. You don't need CUDA if your system does not have a GPU. Read more - https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/