Computing Server Short Manual for Researcher

> (Updated until : 2019-07-16)

> Author : xyy, kchen

```
  Write programs that do one thing and do it well.
  Write programs to work together.
  Write programs to handle text streams, because that is a universal
interface.
                                        ---- Unix philosophy (Peter H. Salus)
```

Other Links

- Install V2Ray client:
  - Windows User
  - macOS User

Table of Contents

Backgrounds and Concepts

Unix

Unix is an operating system that originally developmented at the Bell Labs, by Ken Thompson, Dennis Ritchie, and others in 1970s. Ken Thompson invented the B programming language, the direct predecessor to the C programming language, Dennis Ritchie created the C programming language. Unix's genius design and continuing improvements make it popular and long lasting till today. What's more, other modern operating systems (including Mac and Windows) is significantly influenced by Unix's design and spirit also.

Long Live Unix!

[1] UNIX演义

Linux

Linux is a spiritual successor to the original Unix system. Both of them (and also macOS) are mostly POSIX-compliant [2]. POSIX [1] is a standard that maintain software compatibility between variants of Unix and other operating systems. For example, C source code should be (almost) interchangeable between (almost) POSIX-compliant systems.

When we say the system is Unix-like, we usually mean the system is mostly POSIX-compliant.

POSIX standard covers general concepts, system interfaces, shell & utilities etc. It is unlikely that you will need to read it as a beginner or even as an experienced user.

- [1] POSIX.1-2017. The Open Group Base Specifications Issue 7, 2018 edition.
- [2] POSIX, Wikipedia

GNU and Free Software (自由软件)

GNU means GNU's Not Unix. ;-)

It is the name of a software project lead by Richard Stallman (RMS). The GNU project aims to create a free operating system and a free software toolchain on top of it, by collaborative developers.

The *Free* means: the users are free to *run* the software, *share* it (copy and distribute), study it, and *modify* it.

In order to (legally) guarantee these freedom, a licence is necessary: The GNU General Public License (GNU GPL or GPL), which is a "copyleft" licence, in contrast to "copyright". In simple words, it means that you must publish your source code and licence it under GPL, if it is a derivative work from a GPL licenced code and you are distributing or selling it (with or without charge). See [2,3] for details.

Examples of Software in GNU projects:

GNU Compiler Collection (GCC), GNU C library (glibc), GNU Bash shell, GNU Core Utilities (i.e. basic Linux commands), GNOME desktop environment, GNU Octave, GNU Emacs, GIMP ("photoshop" for Linux), Gnumeric (MS Excel compatible).

Example of GPL or LGPL licenced or GPL/LGPL compatible softwares:

Linux, Firefox, Python, Android (the core part), TensorFlow, 7z, LyX, pdfTeX.

Example of NON GPL/LGPL compatible softwares

- Intel Math Kernel Library, i.e. MKL (Freeware, but not opensource, not free to share).
- Matlab
- macOS
- QQ, WeChat, Skype
- MS Word, MS Excel, Visual Studio, etc.

Ref.

- [1] The GNU Project, by Richard Stallman.
- [2] GNU General Public License.
- [3] GNU Lesser General Public License.

Linux Distributions (发行版)

Linux kernel together with a large set of user softwares (in a place called software repository). e.g. Ubuntu, Debian, Fedora.

In most cases, you will use a "Linux distributions" instead of assembling a Linux kernal and user softwares yourself.

Some Linux distributions focus on out-of-the-box (Ubuntu), some focus on bleeding edge software (Arch, Fedora), some focus on stability (Debian, CentOS).

Software Repository (软件仓库/软件源)

Also called software source.

Software repository is a website that stores and publishes a collection of free softwares that specifically tuned for the distribution. The repository is maintained by developers of the distribution and possibly the original author of the individual software. The softwares in the repository could come from everywhere, e.g. GNU projects, amateur projects, academic projects, and commercial projects.

The software repository could also contain non-free softwares, non-free in the sense that they are not GPL or LGPL compatible. e.g. mp3 decoding and encoding technology (patented, but expired 2017, now totally free), nvidia-driver (not open source).

Why Choose Linux

- No annoying advertisement, more safe

  Mostly, you will install softwares from the software repository. Softwares from official repository are verified and will be securely downloaded, thus avoid the risk of malicious softwares, and with no ads. High priority security fix will be patched in a urgent pace (within few days after been found).

- Let *you* control your computer

  No unwanted background programs running e.g. automatic software updata, force system reboot, virus scan, unexplained internet traffic or CPU usage etc.

  No hiding/joking of CPU usage, network traffic, RAM usage etc.

  Source code are available, when in doubt, or want to make some improvement, or want to learn from it, just check its source code (apt-get source *packagename*).

- Run faster

Linux has a better CPU scheduler and RAM memory management compared to Windows in general. Thus programs could run a few percent (sometimes more than 10 percent) faster after transplant to Linux. e.g., it is observed that Linux version of Mathematica is faster than its Windows version, likely Matlab behaves the same.

- Rich set of tools for advanced user
  - For administration of system

    See hardware details without installing extra softwares.

    Highly configurable and with fair documentation (man).
  - Good for batch processing

    Easy to automate complex procedures.
  - For programming

    Pre-installed gcc and python. Nice editors (vim, gedit). Powerful debug/profiling tools.
- A lot of research-oriented programs are developed under Linux, or best running under Linux. For example
  - LaTeX (run faster, much easier to install)
  - NEST Simulator (Linux only?)
  - NEURON (Linux only)
  - GNU Octave (much faster)
  - FFTW, LAPACK (much easier to compile)
  - TensorFlow (much easier to compile)
  - ssh (you can have Graphical user interface, without using vnc)
- 100% of world top 500 super computers are running Linux.

  At Jun. 2004, for the first time, more than 50% of super computers are running Linux. Then soon Linux dominate super computer list (85% at 2007), eventually reaches 100% at Nov. 2017 and remains today.

[1] Top 500 / Statistics / Development over Time

Where to Start and How to Learn Linux

鸟哥的linux私房菜. Although reads verbose, it sometime quite beginner-friendly.

To really learn Linux, you should force yourself to use it for all your works. So start by pick up a Linux distributions. e.g.

- Debian http://www.debian.org/
- Ubuntu https://www.ubuntu.com/
- Fedora https://getfedora.org/

They all come with good documentations. Just follow the documentation to start your journey.

Linux may not have the exact software you might expect in Windows or Mac, but there are softwares to achieve the same purpose.

Record useful commands/softwares in your personal notebook.

To the author's experience, there are few tasks that are impossible or too costly to be done under Linux. You might need a virtual machine or Wine [1] to accomplish these task.

```
1. Bank of China, login web interface.
2. QQ, wechat. (alternatives: use web based wechat, http://wx.qq.com/)
3. Games. (alternatives: Use Wine, or dual boot with a real Windows)
```

[1] WineHQ - Run Windows applications on Linux, BSD, Solaris and macOS. https://www.winehq.org/

Where to Get Help

> RTFM (Read The Fucking Manual) ---- said folk

- Search in Google, with keywords and name of your Linux distributions
- Most commands come with a manual

  You can open a command window and type `man` *commandname* to access its documentation. Or type `man man` to get help of `man` itself.
- Ask who knows linux

File System Organization

> "Everything is a file" --- a feature of Unix(-like) system

Linux (absolute) path starts by a slash "/", called root directory, there is *no* `C:\` or `D:\`.

The separator between directories is "/", in contrast to Windows' "\".

Relative path starts by ".". e.g.

- `./a.txt` , file `a.txt` in the current directory.
- `../a.txt` , file `a.txt` in the directory of one level up.
- `../b/a.txt` , file `a.txt` in the directory of one level up then down into `b` .

"~" is short for your home directory "/home/${USER}", "${USER}" is your user name.

Linux directories are arranged by their category (e.g. program, setting, library or document), not by software name or company name (like in Windows).

```
/home       user data all go here
/boot       linux kernal and configurations about booting.
/bin        essential system programs (commands)
/lib        system library
/etc        system configurations. mostly in plain text.
/usr/bin    general software executable
/usr/lib    general software library
/usr/share  software documentation and (default) settings
/var/log    system logs
/dev        hardware devices (shown as files)
/proc       information about currently running programs
/sys        settings of hardware and linux kernel
```

The files of a software will be spread in /usr/bin, /usr/lib, /usr/include, /usr/share etc., according to the catalog of each file. Unlike Windows software, mostly in C:\Program Files. In order to manage the spreading files (and software dependency), most Linux systems come with a "Package Manager". In Debian (also Ubuntu), it is APT (command `apt-get`).

Every file has access permission, can be shown by command `ls -l` , can be modified by command `chmod` .

"root" user has the absolute right to do every thing, including deleting the system itself.

To see the full path to the command, use command `which` . e.g. `which python` .

One Day in Linux

> Note: The dollar sign ($) at the begining of the code scripts below indicates the user account type you are logged in currently. DO NOT include it when you test those commond on your own command line environment.

> Note: Virtual username, directory name, server name and IP are used here for illustration. Please replace them with your own settings during your implementations.

Assume Bill is going to work on our computing server(bill@zcpu), and your local machine is also running Linux (or Unix-like), (bill@laptop).

Login

```
$ ssh -p 22 bill@202.121.1.1
```

where 202.121.1.1 is the server IP address, and 22 is the port number.

Operation of File Content

Bill wants to run a test C program. So he opens a text editor.

```
$ nano
```

```
#include <stdio.h>
int main() {
        printf("Hello?\n");
  return 0;
}
```

save and exit

```
C^X
```

name it abc.c

Tips for `nano` usage

Nano is a user-friendly text editor for terminal users. Just treat it as the usual text editor in Windows or Linux GUI, though without mouse operation support. Most commonly used shortcuts are listed on the statue bar at the bottom of the window. You can easily get detailed documantations with `Ctrl+h` (denote as `C^h`).

> Note: Of source, you may find other text editor, like vi/vim, which is one of the most powerful editor in Unix-like OS, though it might not be friendly to freshmen.

Compile and execute files

Now to compile it

```
$ gcc abc.c -o abc
```

And run it

```
$ ./abc
```

Hello?

OK, works.

Operation of Files (Part 1)

Now Bill wants to upload his own files to the server. *Run the following command on your laptop.*

```
$ scp -P 22 /home/bill/work bill@202.121.1.1:/home/bill/
```

> Note: Unlike `ssh` command, the augment option here is CAPITAL P, `-P` .

> Note: If multiple files to be uploaded, do it once for all.

```
$ scp -P 22 /home/bill/work1 /home/bill/work2 /home/bill/work3 bill@202.121.1.1:/home
```

> Note: If you are using Windows OS, you may use WinSCP to manage your files on server, which provides a nice GUI and is more user-friendly. More details. (https://winscp.net/eng/index.php)

Now Bill finds that constantly typing the address and port is anoiying. So let's alias it. *Run the follow commands on your laptop.*

```
$ mkdir -p ~/.ssh
$ cd ~/.ssh
$ nano config
```

add following lines into ssh config file

```
Host zcpu
        HostName 202.121.1.1
        Port 22
        User bill
```

Now Bill can use upload files in a much simpler way.

```
$ scp /home/bill/work zcpu:/home/bill/
```

> Note: for multi-file version,

```
$ scp /home/bill/work1 /home/bill/work2 /home/bill/work3 zcpu:/home/bill/
```

Similarly for download data from the server.

```
$ scp zcpu:/home/bill/data /home/bill/
```

Now Bill can login in like this.

```
$ ssh zcpu
```

Operation of Files (Part 2)

After uploading his own file, Bill wants to create a new folder to store his files.

```
$ mkdir -p foo
```

Move file to the folder `foo`.

```
$ mv work foo/
```

> Note: There is no explicit rename command in Linux. Use `mv` to rename existing files.

```
$ mv work work1  #      rename work as work1
```

Now Bill wants to move to `foo` folder and create a copy of `work`.

```
$ cd foo
$ cp work work_cp        #      create a copy of work as work_cp
```

Besides, Bill wants to know the information of files in current folder.

```
$ ls -l
total 16
-rw-r--r--  1 bill  bill  1 Jan  1 12:00 work
-rw-r--r--  1 bill  bill  1 Jan  1 12:00 work_cp
```

After a while, Bill wants to delete work_cp.

```
$ rm work_cp
```

Inspecting system

When using the computating server, Bill wants to inspect the state of operating system as well as programs and the available resources.

First of all, about the current size of home directory.

```
$ du -sh $HOME
```

Total usage of file system and available disk space.

```
$ df -ht ext4
```

Specifications of the server's CPU

```
$ lscpu
```

Information and current state about server's GPU (only available on GPU server)

```
$ nvidia-smi
```

Most importantly, Bill wants to inspect his programs as well as the resource available on the server.

```
$ top
```

```
top - 23:27:57 up 25 days,  6:30, 15 users,  load average: 18.23, 15.50, 8.92
Tasks: 826 total,   2 running, 518 sleeping,   0 stopped,   0 zombie
%Cpu(s): 25.6 us,  6.8 sy,  0.0 ni, 67.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 26358241+total,  1502220 free, 18863252+used, 73447664 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 73098776 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
19899 mike      20   0 9640456 1.072g 248604 S  2002  0.4 186:07.75 MATLAB
19272 john      20   0  0.171t 0.171t   3340 R  99.7 69.6  72:10.56 t3
20426 mike      20   0   49704   4872   3436 S   1.6  0.0   0:08.43 top
19555 john      20   0   49732   4996   3484 S   1.3  0.0   0:36.48 top
21401 bill      20   0   49740   4816   3448 R   1.3  0.0   0:00.50 top
```

```
55645 mike      20   0   49836   5176   3500 S   1.3  0.0  56:45.55 top
    9 root      20   0       0      0      0 I   0.3  0.0  14:03.25 rcu_sched
 4708 gdm       20   0  591400  27924  17284 S   0.3  0.0  31:56.19 gsd-color
 4828 kernoops  20   0   56936   2364   1912 S   0.3  0.0   0:56.01 kerneloops
    1 root      20   0  226096   9896   6668 S   0.0  0.0   0:33.61 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.32 kthreadd
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:0H
    5 root      20   0       0      0      0 I   0.0  0.0   1:30.69 kworker/u128:0
    7 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 mm_percpu_wq
    8 root      20   0       0      0      0 S   0.0  0.0   0:01.32 ksoftirqd/0
```

The first two lines illustrate the total up time, the number of users, the number of tasks, the number of running tasks of the server. The third line shows the detailed CPU usage, with `us` for user usage, `sy` for system usage, `wa` for waiting usage. (**NOTE**: if `wa` percent is high, which CPU is waiting for I/O operations, it might indicate the I/O bottle neck of your program.) Next two lines shows the detailed RAM(memory) and swap usage.

The rest of table lists all the detailed information about all processes. By default, they are sorted in the descent order of CPU usage. Pay attention of the `RES` column, which is the actual memory usage of processes.

Tips for top

- press `e` to change units of each process (None->k->m->g->t->p: B->KB->MB->GB->TB->PB)
- press `E` to change units of total resources (None->k->m->g->t->p: B->KB->MB->GB->TB->PB)
- press `k` and the PID of target process to kill the target process

  e.g. to kill the bill's top process, whose PID is 21401

  ```
  $ k
  $ 21401
  $
  ```

  It is possible that the program refuse to exit (e.g. MATLAB), in which case you can send the program a "signal 9".

  ```
  $ k
  $ 21401
  $ 9
  ```

- use `>` and `<` to change the sort column of processes (default sort column is %CPU)

> Note: Another way to check the status of specific command.

```
$ ps -e |grep command |less
```

Bill opened 10 python processes by accident. So he wants to kill all of them together.

```
$ jobs
[1]    suspended (tty output)  python
[2]    suspended (tty output)  python
[3]    suspended (tty output)  python
[4]    suspended (tty output)  python
[5]    suspended (tty output)  python
[6]    suspended (tty output)  python
[7]    suspended (tty output)  python
[8]    suspended (tty output)  python
[9]  - suspended (tty output)  python
[10]  + suspended (tty output)  python
```

```
$ pkill python
```

Internet

Bill wants to download a file from http://website.com/files/file.zip over HTTP. *Do not try this address since it's invalid.*

Firstly, he needs to check whether the internet connection is avaiable. *(Optional)*

```
$ ping website.com
```

Then, Bill can simply use `wget` command to download the file.

```
$ wget http://website.com/files/file.zip
```

TMUX

Occasionally, Bill would suffer from the broken pipe issue due to the poor network connection. In case of losting working data, Bill decides to use tmux on server, as windows manager. *`tmux` can prevent your current program from being interrupted due to the broken network connection. You can retrieve your working program when you relogin the server.*

Start a new session

```
$ tmux
$ tmux new  #   both works
```

Detach from session

```
$ Ctrl-a  d
```

i.e. press the key Ctrl and the key A simultaneously, release both, then press the key D.

List existing sessions

```
$ tmux ls
```

Attach to last session

```
$ tmux attach
```

Exit and close current session

```
$ exit
```

Create new window

```
$ Ctrl-a  c
```

Close current window

```
$ Ctrl-a  &
```

Next window

```
$ Ctrl-a  n
```

Switch/select window by number

```
$ Ctrl-a  0 ... 9
```

More instructions : https://tmuxcheatsheet.com

History operations

Bill suddenly forgets to the gcc command to compile c++ code. So he goes to history command to look up his previous command.

```
$ history
```

However, there are too much commands' histories to be contained in a single screen. So Bill uses history command together with less command in a pipe line format.

```
$ history | less
```

Now Bill can just use up and down button to browse all history contents one by one. Besides, to find the gcc command, simply, Bill uses the keywords searching function.

```
/g++
```

Tips:

- Use `n` to search the next occurrence, `Shift-n` to search the previous occurrence.
- Type `-i` to ignore (or not ignore) case in searches.

Programming tips

GCC and Linking Principle

Bill writes a C++ project which contains several files, including main.cpp, functions.cpp, functions.h. Now he need to compile those header file and source files into a excutable program.

```
$ g++ -o exc.out functions.cpp main.cpp
```

In the compiling command above, one option is explicitly specified.

`-o` output file

There are other useful options, including

- `-l` Link on Objective-C and Objective-C++ program, e.g., `-lm` `-lfftw3`
- `-I` Add the directory dir to the list of directories to be searched for header files during preprocessing.

- `-L` Add directory dir to the list of directories to be searched for -l
- `-O` , `-O2` Compile with optimization (-Os, -O, -O2, ...)

Sanitizer

Sometimes, there are some potential undefined behaviors or events of memory leaking in Bill's program. Thus, in order to enable run-time analysis for those dangrous events, Bill uses the follow compiling options.

```
-fsanitize=undefined
-fsanitize=leak
```

Compile and Install External Softwares

Now Bill needs to compile and install external softwares from source files. Here is one of the commonly used procedures.

```
$ cd /path/of/source/                          # move to the directory of source fil
$ mkdir -p build                               # create build directory
$ cd build                                     # move to build directory
$ ../configure --prefix=/path/to/install/      # config compilation with prefix opti
$ make                                         # make target files
$ make -j 32                                   # or make in parallel with 32 cores
$ make install                                 # copy the built files to the target
```

Profiling

Bill's program suffers a low performance. Bill analyzes its performance using perf.

First of all, program needs to be compiled with specified option '-g'.

```
$ g++ -o exc.out functions.cpp main.cpp -g
```

Run perf to record the performance. Use the program as usual during profiling.

```
$ perf record ./exc.out
```

Call perf report to view the performance report.

```
$ perf report
```

More details : https://perf.wiki.kernel.org/index.php/Tutorial

PATH, and Other Environmental Variables

Environmental variables are named values which can be accessed by running programs. It is used as a mechanism to pass parameters to subprocess (program that is started by another program), thus affect the subprocess' behaviour. For example, Shell, as a subprocess of OS, uses `PATH` to determine where to find runnable commands. When starts a program from a Shell, all the environmental variables in that shell will be copied and passed to the program just started, the program is then allow to read and modify its own copy of environmental variables.

In most Shell (e.g. Bash), prepending a `$` sign to the variable name means substitute its value in situ, thus `$PATH` prints the value of `PATH`.

`PATH` is an environmental variable in Linux and other Unix-like operating systems that tells the shell which directories to search for executable files (i.e., ready-to-run programs) in response to commands issued by a user.

Several ways to view the contents of PATH.

```
$ env | grep PATH
$ echo $PATH
```

```
/opt/dell/srvadmin/bin:/opt/dell/srvadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbi
```

Look at the contents of PATH variable, which are bunches of colon-seperated directories. Now Bill wants to add the directory /usr/sbin to the PATH.

```
$ export PATH=$PATH:/usr/sbin
```

Now, view the contents of PATH again.

```
$ echo $PATH
```

```
/opt/dell/srvadmin/bin:/opt/dell/srvadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbi
```

Note that, an addition to a user's PATH variable can be made permanent by adding it to that user's .bash_profile file. .bash_profile is a hidden file in each user's home directory that defines any specific environmental variables and startup programs for that user.

Remarks: The same mechanism exists for other types of files that some program is going to search for by name. Here are a few typical PATH-like variables.

- `PATH` : executables (e.g. /home/username/bin:/usr/local/bin:/usr/bin:/bin).
- `MANPATH` : manual pages (e.g. /usr/local/man:/usr/man).
- `LD_LIBRARY_PATH` : native code libraries (on Linux, in addition to the value of this variable, the lookup path typically contains /usr/local/lib, /usr/lib, /lib and a few others). The name LD comes from dynamic loader, the system component that loads libraries into dynamically linked executables.
- `PYTHONPATH` : Python libraries (e.g. /usr/local/lib/python:/usr/lib/python:/usr/lib/python2.7).

Python Tutorial on HPC-Server

> If you want to use Python on your own laptop, you may simply use PyPI or Anaconda to build your Python distribution. If you are more comfortable with IDE(Integrated Development Environment), Spyder, PyCharm, Visual Studio are recommended here.

Running Python

We've installed essential Python distributions for you. You can simply run python(for Python 2.7), python3(for Python 3.6), python3.7(for Python 3.7).

```
python3
```

```
Python 3.6.8 (default, Aug 20 2019, 17:12:48)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
print("Hello world")
```

Exit

```
exit()
```

Execute Python Script

Create a Python file, `test.py`

```
echo "print('Hello world')" > test.py
```

> Note: You can write your own Python file with your text editor.

Run Python file, test.py, in command line directly.

```
python3 test.py
```

Coding Python in IPython

IPython is a powerful interactive Python shell, provides a rich toolkit to help you make the most out of using Python interactively.

> Refer from (https://ipython.readthedocs.io/en/stable/interactive/python-ipython-diff.html)

1. Accessing help

- Using question mark for requesting general IPython help.

```
In [1]: ?

IPython -- An enhanced Interactive Python
=========================================

IPython offers a combination of convenient shell features, special commands
and a history mechanism for both input (command history) and output (results
caching, similar to Mathematica). It is intended to be a fully compatible
replacement for the standard Python interpreter, while offering vastly
improved functionality and flexibility.

At your system command line, type 'ipython -h' to see the command line
options available. This document only describes interactive features.

MAIN FEATURES
-------------
...
```

- A single question mark before, or after an object available in current namespace will show help relative to this object:

```
In [2]: object?
Docstring: The most base type
Type:      type
```

- A double question mark will try to pull out more information about the object, and if possible display the python source code of this object.

```
In[1]: import collections
In[2]: collections.Counter??

Init signature: collections.Counter(*args, **kwds)
Source:
class Counter(dict):
                '''Dict subclass for counting hashable items.  Sometimes called a bag
                or multiset.  Elements are stored as dictionary keys and their counts
                are stored as dictionary values.

                >>> c = Counter('abcdeabcdabcaba')  # count elements from a string

                >>> c.most_common(3)                # three most common elements
                [('a', 5), ('b', 4), ('c', 3)]
                >>> sorted(c)                       # list all unique elements
                ['a', 'b', 'c', 'd', 'e']
                >>> ''.join(sorted(c.elements()))   # list elements with repetitions
                'aaaaabbbbcccdde'
                ...
```

- If you are looking for an object, the use of wildcards `*` in conjunction with question mark will allow you to search current namespace for object with matching names:

```
In [1]: *int*?
FloatingPointError
int
print
```

2. Shell Assignment When doing interactive computing it is common to need to access the underlying shell. This is doable through the use of the exclamation mark `!` (or bang). For instance,

```
In[1]: !pwd
/User/home/
```

Create an isolated Python environment

> Refer from (https://virtualenv.pypa.io/en/latest/userguide/) `virtualenv` is a tool to create such an isolated Python environment. In this case, it isolate its own installation directories, which is convenient and safe for version management of packages.

Create a new virtual environment

There is one basic command in `virtualenv`.

```
$ virtualenv ENV
```

Here, `ENV` is a directory in which to place the new virtual environment.

> Options: 1. `-p PYTHON_EXE, --python=PYTHON_EXE` : The Python interpreter to use, e.g., `--python=python3.7` will use the python3.7 interpreter to create the new environment. You may choose other interpreters installed in the system. 2. `--system-site-packages` : If you build with it, your virtual environment will inherit packages from your global site-package directories. If you want isolation from the global system and manage your own packages, do not use this flag.

Activate script

In a newly created virtualenv there will also be a activate shell script. On Posix systems, this resides in `/ENV/bin/`, so you can run

```
source /path/to/ENV/bin/activate
```

to activate the environment.

To deactivate your environment, just run:

```
deactivate
```

Removing an Environment

Removing a virtual environment is simply done by deactivating it and deleting the environment folder with all its contents.

```
rm -r /path/to/ENV
```

Manage your own Python environment with PyPI

> Refer from (https://pip.pypa.io/en/stable/quickstart/) Once you activate a virtual environment, you can use `pip` to manage your packages.

1. Install a package from PyPI:

```
pip install SomePackage
```

2. Show what files were installed for SomePackage:

```
pip show --files SomePackage
```

3. List what packages are outdated:

```
pip list --outdated
```

3. Upgrade a package:

```
pip install --upgrade SomePackage
```

4. Uninstall a package:

```
pip uninstall SomePackage
```

5. More help for specific pip command

```
pip <command> --help
```

Get Started with Anaconda

Anaconda is a highly integrated Python package manager and environment manager. Get Conda Cheat Sheet Here.

Prerequisites (already satisfied)

```
apt-get install libgl1-mesa-glx libegl1-mesa libxrandr2 libxrandr2 libxss1 libxcursor
```

Installation

For x86 systems,

1. Using `wget` to download the Anaconda install for Linux.

```
wget -c https://repo.anaconda.com/archive/Anaconda3-2019.10-Linux-x86_64.sh -O ~/
```

> Note: `Anaconda3-2019.10-Linux-x86_64.sh` is used here as an example. You might find the latest verion or other historical verion frome the Link above.

2. (Optional, but recommended) Verify data integrity with SHA-256. Check your own hashes with official released ones here.

```
sha256sum ~/Anaconda3-2019.10-Linux-x86_64.sh
```

3. Run the silent installation of Anaconda for Linux.

```
bash ~/Anaconda3-2019.10-Linux-x86_64.sh -b -p $HOME/anaconda
```

The installer will not prompt you for anything, including setup of your shell to activate conda. To add this activation in your current shell session:

```
eval "$($HOME/anaconda/bin/conda shell.bash hook)"
```

With this activated shell, you can then install conda's shell functions for easier access in the future:

```
# initiate conda
conda init
```

If you prefer that conda's base environment not be activated on startup of shell, set the `auto_activate_base` parameter to `false`:

```
conda config --set auto_activate_base false
```

One more thing: Create your own Python environment

```
# create a new conda environment named as tf-env, with python version 3.7
conda create -n tf-env python=3.7
# activate tf-env
source activate tf-env
# install 必要python库
conda install numpy scipy matplotlib
# 安装tensorflow-gpu 2(默认版本)
pip install tensorflow-gpu
```

```
# 若想安装tensorflow-gpu 1.15，请勿同时安装两个版本，容易出错。
pip install tensorflow-gpu==1.15
```

Activate and Deactivate

Use `conda` command for activating and deactivating anaconda environment.

```
conda activate          # activate
conda deactivate        # deactivate
```

Uninstalling Anaconda:

Simply remove your entire Anaconda directory.

```
rm -rf ~/anaconda
```

Removing Anaconda path from `.bashrc`

You may wish to check the `.bashrc` file in your home directory for lines such as:

```
export PATH="/home/bill/anaconda/bin:$PATH"
```

> Note: Replace `/home/bill/anaconda/` with your actual path.

Conda commands for package management

```
conda list                                          # List installed packages
conda install package # Install package in current environment
conda update package    # Update package in current environment
conda remove package    # Delete package in current environment

conda list -n  your_env                                          # List installed pack
conda install -n your_env package       # Install package in your_env environment
conda update -n your_env package        # Update package in your_env environment
conda remove -n  your_env package       # Delete package in your_env environment
```

Conda commands for environment management

```
conda env list
conda create -n your_env python=x.x             # Create an environment with python v
conda activate your_env                                                  # Act
```

```
conda deactivate
conda remove -n your_env -- all                          # Delete your_env
```

Access Jupyter Notebook

Reference: Slurm远程登录Jupyter Notebook, 分享脚本远程登陆 Jupyter Notebook

As those on your own laptop, you can run `jupyter notebook` on any high performance computing server. The only difference might be the lack of desktop environment so that you may have trouble accessing jupyter notebook through termianl window. Thanks to the power SSH potocol, we can run `jupyter` in no-brower mode, and redirect the remote port to your local port. Then, we can access remote `jupter` notebooks with your local browser.

1. Run `jupyter notebook` on computing server.

```
# Run jupyter notebook in no-browser mode;
# Specify port 8880(if occupied, try other idle ones);
jupyter notebook --no-browser --port=8880
```

2. In your local laptop, open a new terminal and run:

```
# -N: Do not execute a remote command.  This is useful for just forwarding ports.
# -L: Specifies that connections to the given TCP port or Unix socket
#     on the local (client) host are to be forwarded to the given host
#     and port, or Unix socket, on the remote side.
#     First localhost:8880 is for your local port, and the second localhost:8880
#     is for the remote ones.
# Replace your_name with your own username on the computing server
#     with IP address=cluster_url
ssh -N -L localhost:8880:localhost:8880 your_name@cluster_url
```

3. Open the browser on your own laptop, access Jupyter notebook with the url below:

```
# Replace the token with the one appears in the log for jupyter command
#   on your computing server
"http://127.0.0.1:8880/?token=260544c76ee3eeca****************d8523886dde4656"
```

> See also: Remote accessing Jupyter Notebook with SLURM

MATLAB Tutorial on HPC-Server

Recommended Softwares

(on your own laptop with Unix-like operating system)

Meld

Meld is a visual diff and merge tool targeted at developers. Meld helps you compare files, directories, and version controlled projects. It provides two- and three-way comparison of both files and directories, and has support for many popular version control systems.

Meld helps you review code changes and understand patches. It might even help you to figure out what is going on in that merge you keep avoiding.

More details : http://meldmerge.org

LyX

LyX is a document processor that encourages an approach to writing based on the structure of your documents and not simply their appearance.

LyX combines the power and flexibility of TeX/LaTeX with the ease of use of a graphical interface. This results in world-class support for creation of mathematical content (via a fully integrated equation editor) and structured documents like academic articles, theses, and books. In addition, staples of scientific authoring such as reference list and index creation come standard. But you can also use LyX to create a letter or a novel or a theatre play or film script. A broad array of ready, well-designed document layouts are built in.

LyX is released under a Free Software/Open Source license, runs on Linux/Unix, Windows, and Mac OS X, and is available in several languages.

More details : https://www.lyx.org/Home

No packages published

Contributors 2

NeoNeuron Kai Chen

bewantbe count