Introduction to scientific computing (GBS 746): Introduction to Linux (V1.0)

Malay K Basu (malay@uab.edu)

Feb 17-18, 2014

Contents

1 Why Linux?

1. It's open-source, free as in beer.
2. Almost all large-scale computing systems use Linux.
3. Powerful shell, as you'll see.

2 Linux is not Unix

Although very similar, they are not the same. Look at SCO-Linux controversy, if you're interested in the gory details. By all means and purposes, Linux has largely replaced all types of unices. Linux was first created by Linus Torvalds. Together with GNU[1] tools, we now have GNU/Linux operating system.

3 Linux (Unix) philosophy

1. KISS - "Keep it simple, stupid".
2. One program (command), one task.
3. Chain commands one after another to do "beautiful" things.

4 Some basic features of Linux

1. Linux is case sensitive. `A` is not the same as `a`.
2. Everything is a file in Linux.
3. Text file plays a very important role in Linux. All configurations are done by editing text files.
4. Parameters to a program are passed with a `"-"` (minus or tack).
5. You can stop execution of a program by pressing `CTRL+C`.

5 Some basic commands

1. `man <commandname>` will show you help about a particular command.
2. `ls` shows a directory listing.
3. `cd` changes directory
4. `mkdir` creates dir
5. `cp` copies dir or files
6. `mv` renames or moves a file/dir.
7. `rm` removes a file.

Note: `rm -rf` is a special command that will remove everything from the current directory without prompting. If you accidentally execute this command in `"/"`, it will try to wipe out everything from your computer.

[1] Recursive acronym: GNU is not unix.

5.1 Exercise

Show a `man` on `ls`. What does `ls -l` do? What about `ls -F`?

6 File globbing and wildcard

Characters `"*"` and `"?"` has special meaning to `bash`. The former represents "one or more" characters and the latter represents only one, but any character. For e.g., `"*.fas"` represents any filename that has `".fas"` extension.

7 Redirection and piping

There are 3 channels through which a program can accept input and generate output. The input channel is called "standard in" or `STDIN`. There are two types of output channels: the normal output is called "standard out" or `STDOUT`. There is also a "standard error" output or `STDERR`.

The standard output (`STDOUT`) of a program can be "piped" into the standard input (`STDIN`) of another program. This is commonly done using a `"|"` or a "pipe" symbol. For e.g.,

```
ls *.txt | wc
```

Here the output or STDOUT of `ls` is "piped" into `STDIN` of `wc`. The final result is shown onto the terminal. In this case, it counts the number of text files in the current directory.

If you would like to redirect the output of a program to a file. This can be done in two ways:

1. `>` creates a new file. Caution: it will overwrite without any warning
2. '>>' Appends to the existing file.

For e.g.,

```
ls *.txt > ../list.txt
```

This will create a list of text files in the current directory. This list will be created on the parent directory.

If you would like pipe the STDERR of a program you need to add a 2 before the redirection sign:

```
ls *.txt 2> error.log
```

You can even do both redirections simultaneously:

```
ls *.txt >../list.txt 2>error.log
```

8 `cat`

`cat` dumps the content of the file to output.

9 more

more is a paginator. It shows an input page by page.

10 wc

wc count the lines, words, and characters. If you are just interested in the lines use wc -l.

11 head

head shows the first few lines of a file. If you are interested in specific lines use head -n <no_of_lines>.

12 tail

Like head, tail displays the last lines of the file. A very common use of tail is to skip the first few lines of the file. If you want to skip the first 2 lines of a file,

```
tail -n+3 <filename>
```

13 Directory structure

Unlike Windows Linux file systems all directories originate from a common node, called "root", indicated by "/". There is no "C:", "D:", etc.

13.1 Exercise:

Change your current directory using cd to "/". Show a directory listing of your "/".

14 Where I lay my head is home

The directory that you log into is your home. Also indicated by ~. The path is /home/Username.

14.1 Exercise

Do a directory listing of your home.

15 Login scripts

When you open a terminal in linux system, you are basically running a program or shell. The default shell is bash.

1. .bashrc - runs every time a new terminal is opened.

2. `.bash_profile` - runs every time you login.

You can put anything you want in these files. But where are they? These are hidden files. Any filename that starts with a "." is hidden. You need to type `ls -a` to view the file.

16 `alias`

You can change any command by *aliasing* it.

```
alias ls="ls -l"
```

16.1 Exercise

Change `ls` to show always formatting. That means whenever you type `ls`, it should get silently replaced by `ls -F`. You may put the command in your `.bashrc` file.

17 Path

1. `pwd` - current path
2. `"."` - current directory.
3. `".."` - parent directory

If your path starts with `"/"`, then the path is *absolute*, otherwise it is a *relative* path.

17.1 Exercise

What is the absolute path of your `home`? Using a relative path, go up all the way to root and come back again in the same directory. For e.g., if you are currently in `/a`, the answer to the latter question will be `../a`.

18 Environment variables

An environment variable is defined in bash as follows:

```
export foo=bar
```

After this `bar` can be referred to as `$foo` anywhere within the shell. You may want to define an environment variable in `.bashrc` or `.bash_profile`. You can list all the environment variables already define for you by running `env`.

19 PATH env variable

To run a command (or program), the location of the program has to be in a particular environment variable called PATH. You can add to the existing $PATH by adding to it like this:

```
export PATH=$PATH:/some_dir/of/my/choice
```

You may add line like this in your .bashrc or .bash_profile. You can also run a program by calling it by absolute path. To run a command from the current directory call it by its relative path like this

```
./my_awesome_program
```

You can find the absolute path of a command by using which <commandname>.

19.1 Exercise

Using which find the location of ls. Change the directory to the parent directory of ls.Overwrite the current path by adding empty string to it. Now again do a which on ls. What happens? Run ls using a relative path.

20 File permission

There are three kind of permissions: read, write, and execute. A file need to have executable permission in Linux to run. You can change the permission of a file that you own by the command chmod. You can check the permission of a file by ls -l. chmod is run like this:

```
chmod a+wrx filename
```

a means all users. + grants permission. w,r,x denotes write, read, and execute permissions.

20.1 Exercise

Change the permission of a file that you own to executable by everyone.

21 Compression and archiving

The two most common compression programs that you'll encounter are gzip and bzip2. Both have identical syntaxes. Traditionally they take .gz and .bz2 extensions, respectively. They generally operate on one file at a time. If you just type gzip file, the compressed file will be automatically named file.gz and original file will be replaced. Interesting thing about these programs are that they can be used by piping. For e.g.,

```
echo "Hello there." | gzip -9 -c >hello.gz
```

You can use compressed files without uncompressing it. zcat can be used for gzip and bzcat is for bzip2.

Corresponding uncompressing programs are gunzip and bunzip2.

To compress many files (including directories) you need to use tar first then use gzip/bzip2.

```
tar -c Downloads/ | gzip -c >downloads.tar.gz
or,
tar -cvzf downloads.tar.gz Downloads/
```

If you get a file with `.tar.gz` or `.tar.bz2` extensions. Unzip them like this:

```
tar -xvzf myfile.tar.gz
tar -xvjf myfile.tar.bz2
```

22 Program compilation

There is a standard way to compile a `C` program in Linux. Almost all the software are distributed as `.tar.gz` files. These are source codes. You should compile and install the software like this:

```
tar -xvzf foo.tar.gz
cd foo
./configure
make
make install
```

The last install step may require you switch to `root`.

22.1 Exercise

`TopHat` is splice junction aligner for RNASeq data. Download `TopHat` source code from http://tophat.cbcb.umd.edu/downloads/tophat-2.0.10.tar.gz, then install the software in your machine.

23 Working with structured text file

23.1 `cut`

`cut` extracts the columns from a text file. The default field separator is `tab`. To extract 2nd column for a `tab-delimited text file`,

```
cat foo.txt | cut -f 2
```

23.2 `sort`

Sort the line of a input in alphabetical order.

```
cat foo.txt | sort
```

23.3 `uniq`

Removes duplicate lines if the are consecutive. You have to use `sort` to use `uniq` correctly.

```
cat foot.txt | sort | uniq
```

24 wget

`wget` is a swiss-army-knife of web downloader. You can use it to download `http` or `ftp` files. The basic use is very simple.

```
wget <URL>
```

But once you know how to use `wget` correctly, you can use it to do pretty sophisticated stuff. For e.g., to download all the PDF files for a URL.

```
wget -A.pdf <URL>
```

```
or, if the URL is ftp
```

```
wget ftp://myurl.net/*.pdf
```

We will be using `wget` to download files from `ftp` sites like NCBI.

24.1 Exercise

If you go to the *Drosophila melanogaster* genome `ftp` site (ftp://ftp.ncbi.nlm.nih.gov/genomes/Drosophila_melanogaster/RELEASE_5_48/) on NCBI, you will find that files are scattered in 5 directories, each corresponding to one chromosome:

```
CHR_2 CHR_3 CHR_4 CHR_Un CHR_X
```

Each of these directories contain many files. The protein sequences are in `.faa` files. To download all the protein `fasta` files in the current directory, use

```
wget ftp://ftp.ncbi.nlm.nih.gov/genomes/Drosophila_melanogaster/RELEASE_5_48/\
CHR_{{2..4},Un,X}/*.faa
```

This download works, because there only few subdirectories. If there are too many directories, then we should use,

```
wget -r -A.faa ftp://ftp.ncbi.nlm.nih.gov/genomes/Drosophila_melanogaster/RELEASE_5_48/
```

But the command recreates all subdirectories. Finally this should replicate the first command,

```
wget -r -nH --cut-dirs=4 -A.faa ftp://ftp.ncbi.nlm.nih.gov/\
genomes/Drosophila_melanogaster/RELEASE_5_48/
```

In last two commands we download every `".faa"` files, because each directory is *D. melanogaster* represents a chromosome. But look at the *D. pseudoobscura* site (ftp://ftp.ncbi.nlm.nih.gov/genomes/Drosophila_pseudoobscura/). The directories are,

```
CHR_2 CHR_3 CHR_Un mapview
```

Clearly `mapview` is *not* a genome directory. In this case we should download `".faa"` files only from directories that matches `CHR_*`.

```
wget -r -nH --cut-dirs=4 -A.faa -I genomes/Drosophila_pseudoobscura/CHR_* \
ftp://ftp.ncbi.nlm.nih.gov/genomes/Drosophila_pseudoobscura/
```

25 grep

`grep` search for a pattern in file or input.

25.1 Exercise

In the last step we downloaded a bunch of `.faa` files for *Drosophila* proteome. These files are in `FASTA` format; each sequence starts with a `">"` sign. We can grep for these lines to to print these lines only,

```
cat *.faa | grep \>
```

Note that we had to put a `"\"` character in front of `">"`. This is called a shell *escape*. What will happen if we do not escape `">"`.

26 find

`find` searches files recursively going into a directory hierarchy.

26.1 Exercise

In the `wget` examples we downloaded `.faa` files using this command,

```
wget -r -A.faa \
    ftp://ftp.ncbi.nlm.nih.gov/genomes/Drosophila_melanogaster/RELEASE_5_48/
```

This recreated all the intermediate directory and `.faa` files are deeply nested in a directory hierarchy. We can find all the `.faa` files like this,

```
find ftp.ncbi.nlm.nih.gov/ -name "*.faa" -print
```

This prints the list of files:

```
ftp.ncbi.nlm.nih.gov/genomes/Drosophila_melanogaster/RELEASE_5_48/CHR_3/NT_033777.faa
ftp.ncbi.nlm.nih.gov/genomes/Drosophila_melanogaster/RELEASE_5_48/CHR_3/NT_037436.faa
ftp.ncbi.nlm.nih.gov/genomes/Drosophila_melanogaster/RELEASE_5_48/CHR_4/NC_004353.faa
ftp.ncbi.nlm.nih.gov/genomes/Drosophila_melanogaster/RELEASE_5_48/CHR_2/NT_033779.faa
ftp.ncbi.nlm.nih.gov/genomes/Drosophila_melanogaster/RELEASE_5_48/CHR_2/NT_033778.faa
ftp.ncbi.nlm.nih.gov/genomes/Drosophila_melanogaster/RELEASE_5_48/CHR_X/NC_004354.faa
```

Interesting thing is that we can not only find files but execute arbitrary command on each of these files. For e.g., we can pull all the `".faa"` files in the current directory like this:

```
find ftp.ncbi.nlm.nih.gov/ -name "*.faa" -exec cp {} . \;
```

27 Final problem set

27.1 Problem 1

PFAM is a database of domains. It also provides pre-calculated domains for all proteomes. The current version can be found here ftp://ftp.sanger.ac.uk/pub/databases/Pfam/releases/Pfam27.0/proteomes/. Each file is a proteome identified by its taxonomic ID. Human has the ID 9606. Each of these files is tab-delimited and the 6th column is the domain ID. Download the human proteome file using `wget`. After downloading write just a single line of `bash` to find how many domain types (unique domains) are there in human genome. You may use as many commands, chained in pipes, as you wish.

27.2 Problem 2

On NCBI FTP site all the bacterial genomes are present in the directory ftp://ftp.ncbi.nlm.nih.gov/genomes/Bacteria/. There are hundreds of genomes in that directory. Using a single `wget` command download proteomes corresponding to all the *Yersinia_pestis* strains. The proteomes should be downloaded in such a way that each `".faa"` files are inside separate directory. A `ls` should print something like this:

```
Yersinia_pestis_A1122_uid158119/
Yersinia_pestis_Angola_uid58485/
Yersinia_pestis_Antiqua_uid58607/
Yersinia_pestis_biovar_Medievalis_Harbin_35_uid158537/
Yersinia_pestis_biovar_Microtus_91001_uid58037/
Yersinia_pestis_CO92_uid57621/
Yersinia_pestis_D106004_uid158071/
Yersinia_pestis_D182038_uid158073/
Yersinia_pestis_KIM_10_uid57875/
Yersinia_pestis_Nepal516_uid58609/
Yersinia_pestis_Pestoides_F_uid58619/
Yersinia_pestis_Z176003_uid47317/
```

27.3 Problem 3

Starting from last directory write a single `bash` command line to count the total number of proteins in all the *Yersinia pestis* strains together. You may chain as many commands as you wish.