CB2-101: Introduction to Linux

Malay K Basu (malay@uab.edu)

Nov 2-3, 2017

Contents

Why Linux?

For a complete history of Linux look at the WikiPedia article.

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.

Linux is not Unix

All unix like operating systems have one thing in common, they are POSIX compatible. POSIX stands for **Portable Operating System Inerface**, a set of C libraries. Another freely (as in beer) available unix like operating system is BSD. Mac OSX is a BSD variant with open-source kernel called, Darwin.

Although very similar (all of them are POSIX compatible), Linux and Unix 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 kernel was first created by Linus Torvalds. Together with GNU[1] tools, we now have GNU/Linux operating system.

GNU software stack

In 1983, Richard Stallman at MIT created Free Software Foundation (FSF). The dream was to create a "free" (as in freedom) operating system. By 1990, he had almost everything except the kernel. This software stack is called GNU. Linux provided that.

In a sense, what is more important for us is the GNU software stack. You can get it on BSD (another Unix like operating system) variants, such as OSX or even on Windows through CygWin project.

GPL: GNU general public license

Almost all software on GNU/Linux are licensed under this licensing system, including the Linux kernel. The full text of the license can be found at https://www.gnu.org/copyleft/gpl.html. The license allows you modify and redistribute the software as you please (free as in freedom). But, if you have modified the software, the modified version has to be redistributed under the original license. That means the freedom that the license gave you, you should pass the same freedom to others. "Treat others the way you would like to be treated" – the golden rule of morality, the core of the rule of *Dharma*! This license was a game changer!

Linux-Unix-GNU philosophy

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

Some basic features of GNU/Linux

1. It 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`.

[1] Recursive acronym: GNU is not unix.

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.

Exercise

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

Finding help

A typical Linux system has several levels of help. Some of these may not be available for specific software. These are detailed below:

`-h` or `--help`

Most well-designed program will have an in-built help. If it is there then most probably this can be accessed by passing `-h` or sometimes `--help` option to the software. Try this one first.

`man` pages

Almost all program has a `manual` page. That can be accessed using

```
man <program name>
```

`info` pages

Many software in Linux have more easy to read help page. This can be accessed using

```
info <program name>
```

If you run `info` without any option, it will show the entire `info` tree. You can then select by keyboard which portion of the `info` tree you would like to read.

`apropos`

Sometimes you do not remember the exact command name, but you would like to do a specific operation. For e.g., suppose you have forgotten the command `wc`. You can find all commands that has the word `line` in the manual page by using

```
apropos line
```

You'll see that `wc` is one of the listed commands. You'll also notice that `apropos` return each command with a number in the parenthesis. `wc` is listed as `wc(1)`. The `1` in the parenthesis indicates that it is a runnable command. If you have something else, that means it is not a runnable command.

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.

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
```

`cat`

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

`more`

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

wc

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

head

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

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>
```

Directory structure

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

Exercise:

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

Where I lay my head is `home`

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

Exercise

Do a directory listing of your `home`.

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.

alias

You can change any command by *aliasing* it.

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

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.

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.

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`.

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`.

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>`.

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.

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.

Exercise

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

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
```

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`.

Exercise

`HMMER` is a software for sequence analysis using profile hidden Markov model. Download `HMMER` source code from http://selab.janelia.org/software/hmmer3/3.1b1/hmmer-3.1b1.tar.gz, then install the software in your machine.

Working with structured text file

`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
```

`sort`

Sort the line of a input in alphabetical order.

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

`uniq`

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

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

`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.

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/
```

grep

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

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 `">"`.

A `"-v" option to`grep‘ will print lines that are not matching. To find only the sequence lines we can do the following.

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

tr

This commands *translates* one or more character in a string to another.

```
echo 'uab' | tr '[a-z]' '[A-Z]'
```

This command will acutally convert the word `uab` to `UAB`.

One very commong use of `tr` is to delete the line terminating character. This is an invisible character represented as `"\n"` that are present after each line. For example to join all lines in the file, you may do this:

```
cat myfile.txt | tr -d "\n"
```

find

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

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 {} . \;
```

Looping in bash shell

Sometimes it is useful to loop through a set of files. For e.g., if we would like change the filename of all the files with `".faa"` extension to `".fas"` we will do this:

```
for i in `ls *.faa`
do
    base=`basename $i .faa`
    mv $i ${base}.fas
done
```

rsync

`rsync` is a synchronization software. The first time you use `rsync` it will try to copy everything to the target directory. Next time onwards it will only copy the changed file to the target.

```
rsync -avP source_dir/ target_dir/
```

`rsync` can also synchronize two directories over network. If you have `ssh` account on a remote machine,

```
rsync -avP source_dir/ user@password:/target_dir/
```

unison

`unison` will make two directories identical. It performs a bidirectional synchronization.

```
unison -auto source_dir/ target_dir/
```

Like `rsync` `unison` also works over network.

```
unison -auto source_dir/ ssh://user@password//target_dir
```

Perl

Created by Larry Wall[2], Perl has been the language of choice in bioinformatics for a long time. Perl is always there in all GNU/Linux machine. It is an extremely powerful general purpose programming language. We will use a little bit of Perl to write what is known as perl "one-liners".

```
perl -e 'print "Hello world\n"'
```

This prints the text `"Hello world"` on the screen. Generally, if you would like loop through all the lines given by STDIN, you would write a program like this:

```
while (<STDIN>) {
    print;
}
```

But, using a one liner with `-n` option will put everything automatically within a `while` loop,

```
perl -ne 'print'
```

If you're extracting data from a tab-delimited text file, you can use `-a` that autosplits the line into an array, called `@F`. In Perl, an array is denoted by an `@` in front of a variable and the individual element can be accessed as `$F[0], $F[1],...`, etc.

Exercise

If you have downloaded the human PFAM file form the Problem 1, the column 12 has the score. We would like to extract the lines that have score >2000.

```
zcat 9606.tsv.gz | tail -n+4 | perl -nae 'print if $F[11] >2000'
```

Note that although we are interested in column 12, we actually print `$F[11]`, because arrays in Perl starts with `0`.

[2]Most quotable software guru. Take a look at his quips. He is famous for his witty and intelligent state-of-the onion talks. Listen to his particularly brilliant 1998 talk at http://www.perl.com/pub/1998/08/show/onion.html.

To give you a more sophisticated example, let's extract all the GIs from the *Drosophila* `.faa` files of chromosome 2. There are two files for chromosome 2: NT_033778.faa and NT_033779.faa. If you do a `head` on the first file, it prints,

```
>gi|116007464|ref|NP_001036428.1| CG17683, isoform A [Drosophila melanogaster]
MSRLSRALQLTDIDDFITPSQICIKPVQIDKARSKTGAKIKIKGDGCFEESESGNLKLNKVDISLQDCLA
CSGCITSAEEVLITQQSREELLKVLQENSKNKASEDWDNVRTIVFTLATQPILSLAYRYQIGVEDAARHL
NGYFRSLGADYVLSTKVADDIALLECRQEFVDRYRENENLTMLSSSCPGWVCYAEKTHGNFLLPYVSTTR
SPQQIMGVLVKQILADKMNVPASRIYHVTVMPCYDKKLEASREDFFSKANNSRDVDCVITSVEVEQLLSE
AQQPLSQYDLLDLDWPWSNVRPEFMVWAHEKTLSGGYAEHIFKYAAKHIFNEDLKTELEFKQLKNRDFRE
IILKQNGKTVLKFAIANGFRNIQNLVQKLKREKVSNYHFVEVMACPSGCINGGAQIRPTTGQHVRELTRK
LEELYQNLPRSEPENSLTKHIYNDFLDGFQSDKSYDVLHTRYHDVVSELSISLNINW
>gi|116007468|ref|NP_001036430.1| CG17683, isoform C [Drosophila melanogaster]
MSRLSRALQLTDIDDFITPSQPVQIDKARSKTGAKIKIKGDGCFEESESGNLKLNKVDISLQDCLACSGC
```

The GI's are just after the `">"` sign.

```
cat *.faa | perl -ne 'print $1, "\n" if ($_ =~ /^>(gi\|\d+)\|/)'
```

Now extract the accession number yourself.

Problem set

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.ebi.ac.uk/pub/databases/Pfam/current_release/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.

Problem 2

On NCBI FTP site all the bacterial genomes are present in the directory ftp://ftp.ncbi.nlm.nih.gov/genomes/archive/old_refseq/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/
```

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.

Problem 4

E. coli MG1655 is the standard referece strain of *E. coli*. The protein FASTA file for this strain can be downloaded from ftp://ftp.ncbi.nlm.nih.gov/genomes/archive/old_refseq/Bacteria/Escherichia_coli_K_12_substr___MG1655_uid57779/NC_000913.faa. (a) Using just `bash` commands can you find out what is the average length of protein in this strain? You may use as many commands as you may wish. (b) In the second part of the problem, write your commmands in a bash script such a way that given any fasta file as the option to the script, it can print out the average length of protein in the file.

Appendix

How to resize VirtualBox disk image

The `.vdi` file must be resizable for this to work. First resize the VirtualBox disk image to the required size in megabytes:

```
$ VBoxManage modifyhd cb2-fedora.vdi --resize 10240
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
```

Once you have resized the image, login to your VM, switch to root account and then do this.

```
# First create a new partition with with the free space
$ fdisk /dev/sda # Create a new partition of type 8e

# Create a new physical volume
$ pvcreate /dev/sda3

# Add the new volume to volume group
$ vgextend fedora /dev/sda3

# Increase the size of the logical volume to occupy all the free space
$ lvextend -l +100%FREE /dev/fedora/root

# Resize the filesystem
resize2fs /dev/fedora/root
```