Talking to a Computer: Fundamentals of the Command Line

Computational Tools for the Working Biologist. Workshop 1

Yuxi (Jaden) Long

You heard about this great software tool, but you have no idea where to type all the monospaced text that appears in its tutorial. Let's solve that.

Central idea/technique: Being able to run software from source code

Practice:

- Basics of command line: terminal, man, package manager, text editor
- Running softwares: git, GitHub, make, readme
- Command line scripting*: shebang, variables, command substitution, redirection, piping, awk, grep

1 Getting started

Most softwares assume a *NIX operating system. That is, MacOS or Linux. We will do the same for this workshop. Following we will go over how to open the terminal. This is the only section where instructions differ for audience of different operating systems.

- Windows users should install the latest version of Windows Subsystem for Linux (WSL). Windows Terminal should also be nice, but takes a while to set up.
- MacOS users have a built-in terminal which can be found in the folder *Applications/Utilities* or via searching by spotlight.
- Linux users can open their terminal by Ctrl+Alt+T.

Open up the command line, type `ls -l /`, and hit `<enter>`, you should see something like this:

```
$ ls -l /
total 100
lrwxrwxrwx   1 root root     7 Sep 29  2022 bin -> usr/bin
drwxr-xr-x   4 root root  4096 Dec 26 08:29 boot
drwxrwxr-x   2 root root  4096 Mar  1  2021 cdrom
drwxr-xr-x  21 root root  6240 Dec 26 08:26 dev
drwxrwxr-x 195 root root 12288 Dec 26 08:29 etc
drwxr-xr-x   4 root root  4096 Feb  4  2022 home

<Many more lines omitted>
```

Congratulations, you just ran your first shell command!

2 Commands

Let's recap this command `ls -l /` more closely. It has three parts, separated by spaces: `ls`, `-l`, and `/`. The first part (`ls`) is the **executable**, and the parts that follow (`-l /`) are the **parameters**.

2.1 The Executable

`ls` is the executable program. It is just like an application you open on your computer, such as Google Chrome, except that this program is executed in the terminal.

Your operating system has search paths for executables. Run the command `echo $PATH` to see them: Mine looks like

```
$ echo $PATH
/home/longyuxi/.poetry/bin:/home/longyuxi/bin:/home/longyuxi/.local/bin:/usr/⌋
 ↪  local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/l⌋
 ↪  ocal/games:/snap/bin:/home/longyuxi/bin:/home/longyuxi/java/jdk-19.0.2/bi⌋
 ↪  n
 ↪  ...
```

This produces a list of folders separated by colons. When you run the `ls` command in the shell, your operating system looks at each of these folders and see if the `ls` executable is in that folder, and executes the first `ls` executable it finds. To find exactly which `ls` you just executed, invoke the command

```
$ which ls
/usr/bin/ls
```

If you get something like `ls:  aliased to ls -G`, then try

```
$ type -a ls
ls is an alias for ls -G
ls is /bin/ls
```

This tells us the path of the executable behind the `ls` command we just invoked. When executing an executable, we can also use its path (section 2.3) to execute it. In other words, the command `/usr/bin/ls -l /` is equivalent to `ls -l /`.

The audience interested in what `$PATH` is can read more about **environment variables**.

2.2 The Parameters

To find what parameters a command takes, one can usually call the command with the `--help` parameter. For example,

```
$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
```

```
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
  -a, --all                  do not ignore entries starting with .
  -A, --almost-all           do not list implied . and ..
      --author               with -l, print the author of each file
  -b, --escape               print C-style escapes for nongraphic characters
      --block-size=SIZE      with -l, scale sizes by SIZE when printing them;
                               e.g., '--block-size=M'; see SIZE format below
  -B, --ignore-backups       do not list implied entries ending with ~
  -c                         with -lt: sort by, and show, ctime (time of last
                               modification of file status information);
                               with -l: show ctime and sort by name;
                               otherwise: sort by ctime, newest first
...
```

A more standard way is to call `man <command>`, where `man` stands for “manual”. For example,

```
$ man ls
LS(1)                                                 User Commands
 ↪  LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information about the FILEs (the current directory by default).
        ↪  Sort entries alphabetically if none of -cftu-
       vSUX nor --sort is specified.

       Mandatory arguments to long options are mandatory for short options
        ↪  too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

       -b, --escape
              print C-style escapes for nongraphic characters

       --block-size=SIZE
              with -l, scale sizes by SIZE when printing them; e.g.,
               ↪  '--block-size=M'; see SIZE format below
```

```
...
```

Note that in `man <command>`, the output is displayed with `less`, which is a very convenient command-line viewer for large texts. Use the keys `j` and `k` to navigate down or up, and `q` to quit. Its most commonly used controls can be found widely on the Internet (e.g., here). and of course, you can run `man less` to view the manual of `less`.

Each parameter begins with a single or double dash and can be chained together. For example, if I want to list the files

- In a long format (`-l`);
- In a human-readable format (`-h`);
- Sorted by time (`-t`);
- In reverse order (`-r`),

I can run the command `ls -lhtr`, which is equivalent to `ls -l -h -t -r`.

Often there are single letter and full word versions of the same parameter, where the single letter version is prefixed with a single dash and the full word version is prefixed with a double dash. For example, `-a` and `--all` are equivalent. Finally, some parameters allow you to specify an additional value with that parameter (e.g., `-block-size=SIZE`, where you can change the `SIZE` part to a value for your purposes).

2.3 The Path

Recapitulating, we saw that `ls` takes `[OPTION]` and `[FILE]`. We have dealt with the `[OPTION]` part. For the `[FILE]` part, we can specify the path to a file or folder for `ls` to list. We can also not specify a directory here, so that `ls` lists files under the current directory.

Paths can be specified either relatively or absolutely.

- Absolute paths start with a slash (/) and specify the path from the root directory. For example, in the command we executed earlier, `ls -l /`, the path / means we are listing the root directory.
- All other paths are relative paths, which means they are specified relative to the current working directory. You can find the current working directory by running the command `pwd` (print working directory).

For example, currently I have

```
$ pwd
/home/longyuxi
```

And when I run `ls`, I see that *Downloads* folder is under this directory:

```
$ ls -l
total 461776

...
drwxr-xr-x 12 longyuxi longyuxi     61440 Dec 22 15:01  Downloads
...
```

To list my downloads folder, it is then equivalent to run either of the following commands:

- `ls -l Downloads`
- `ls -l /home/longyuxi/Downloads`

3 A few more things

Below are a few miscellaneous things before we start running softwares.

3.1 Some important commands*

I will only give a synopsis for their purpose. Read the manual for details on how to use them.

- `ls`: lists files
- `vim`/`emacs`/`nano`: text editor
- `pwd`: prints current directory
- `cd`: change directory
- `mkdir`: create empty directory
- `rm`: remove file, or directory (with `-r`)
- `top`/`htop`/`nvtop`: CPU/GPU usage monitors
- `code`: if you have VS Code installed, this opens VS Code in a specified directory
- `df`: disk space usage
- `du`: disk usage of folder
- `less`: view text file
- `echo`: print out things. Useful for writing something simple to file with redirection (section 3.4).
- `rg`: search texts for string
- `fd`: search directory for file with name

`rg` and `fd` are not bundled with your operating system and needs to be installed manually.

3.2 Package manager

The easy way to install softwares is through a package manager (we will see the hard way later). Linux distributions often come with a package manager. For example, Ubuntu has `apt`. A good package manager for MacOS is Homebrew. Now, say we want to install the funny software *cowsay*, we can look up *cowsay* on the package indices of `apt` or `brew`, and install it via (on Ubuntu) `sudo apt install cowsay` or (on MacOS) `brew install cowsay`. Then you can use `cowsay` directly in the terminal:

```
$ cowsay amogus
 ________
< amogus >
 --------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

3.3 Text editor

It is often convenient to directly edit text files from within the command line. The most newbie-friendly command-line text editor is `nano`. For example, to write something into a file called `test.txt`, one can do

1. `nano test.txt`
2. Write something
3. Exit and save the file by `<CTRL-X>` followed by `Y`.

You can check that this file is indeed present by `ls -l test.txt` and `less test.txt`.

There are more efficient terminal-based text editors, such as *vim* and *emacs*, which take a learning curve but is worth the time.

3.4 Command line scripting*

For batch processing in the command line, it is convenient to write scripts. Look up any tutorial on the Internet for details. In particular, I find the following topics to be important:

- Shebang
- Variables
- Command substitution
- Output redirection
- Piping
- `awk` and `grep`

3.5 File permissions

The first column from the `ls -l` is rather important. Read this article for details. This will become relevant when running softwares.

4 Running softwares

Let's install and run a software from beginning to end. Here, we emulate the scenario of reproducing someone else's analysis from a paper.

Suppose you want to reproduce a work on genome alignment, and the paper you found indicates that the authors ran a software called *BWA*.

4.1 Researching

Use the search engine to find information about this software. In particular,

- Is this software command-line based, or graphics-based? Unless indicated otherwise, softwares are usually command-line based. For the following tutorial, we assume that the software of interest is command-line based.
- Is this software proprietary or open-source?
- Where is the documentation?

Looking up *BWA*, we can find its website and its GitHub repository.

Every software repository should have a **README file** at its base folder, which documents necessary information about the software, like installation instructions and license information. GitHub conveniently renders this README file on the webpage. Looking more closely, we see that BWA offers an installation instruction.

4.2 Installing

It is often best to directly follow the installation instruction given by the original author. However, it can be more convenient to install via a package manager (section 3.2). Softwares are written in **source codes**, which are human readable/writable files. Source codes are compiled into binaries, which contain instructions for the machine to execute. This process is often also called **building** the software.

The installation instruction for *BWA* reads:

```
git clone https://github.com/lh3/bwa.git
cd bwa; make
./bwa index ref.fa
./bwa mem ref.fa read-se.fq.gz | gzip -3 > aln-se.sam.gz
./bwa mem ref.fa read1.fq read2.fq | gzip -3 > aln-pe.sam.gz
```

The first two lines explain how to download and build the software, and the last three lines show an example use case (see section 3.4 for what | and > in these commands mean). Let's focus on the first two lines.

`git clone` downloads ("clones") this repository. First, install git via your package manager. Then navigate to a directory of your choice in the terminal with `cd`, and execute this command. After the command is done, we can see the bwa folder in current directory.

```
$ ls -l
total 4
drwxrwxr-x 5 longyuxi longyuxi 4096 Dec 26 13:35 bwa
```

Next, execute `cd bwa; make` will build `bwa`. The output binary can then be located in the base folder of this directory.

```
$ ./bwa

Program: bwa (alignment via Burrows-Wheeler transformation)
Version: 0.7.17-r1198-dirty
Contact: Heng Li <hli@ds.dfci.harvard.edu>

Usage:   bwa <command> [options]

Command: index         index sequences in the FASTA format
         mem           BWA-MEM algorithm
         fastmap       identify super-maximal exact matches
         pemerge       merge overlapping paired ends (EXPERIMENTAL)
         aln            gapped/ungapped alignment
         samse          generate alignment (single ended)
         sampe          generate alignment (paired ended)
         bwasw          BWA-SW for long queries (DEPRECATED)

         shm            manage indices in shared memory
         fa2pac         convert FASTA to PAC format
         pac2bwt        generate BWT from PAC
         pac2bwtgen     alternative algorithm for generating BWT
         bwtupdate      update .bwt to the new format
         bwt2sa         generate SA from BWT and Occ

Note: To use BWA, you need to first index the genome with `bwa index'.
      There are three alignment algorithms in BWA: `mem', `bwasw', and
      `aln/samse/sampe'. If you are not sure which to use, try `bwa mem'
      first. Please `man ./bwa.1' for the manual.
```

We can also run each of the commands to the documentation of that particular command. For example,

```
./bwa index

Usage:   bwa index [options] <in.fasta>

Options: -a STR    BWT construction algorithm: bwtsw, is or rb2 [auto]
         -p STR    prefix of the index [same as fasta name]
         -b INT    block size for the bwtsw algorithm (effective with -a
          ↪   bwtsw) [10000000]
         -6        index files named as <in.fasta>.64.* instead of
          ↪   <in.fasta>.*

Warning: `-a bwtsw' does not work for short genomes, while `-a is' and
         `-a div' do not work not for long genomes.
```

Great, we just built BWA! And we also see its documentation on how to run it.

4.3 A deeper look at git and make*

Git is a version control system. There are plenty of blogs online detailing it (example).

GitHub is an online service hosting Git repositories.

make is a build system, and it takes a `Makefile`. There are also plenty of blogs on it (example).

Underlying the `make` command executed in our installation of BWA is *gcc*, the GNU compiler for C code, which compiled the source code of BWA written in the C programming language to an executable.

5 Assignment

We need a few softwares for the genomics problem in the next workshop. Hone the skills you learned in this workshop by installing them.

1. BWA (which you just installed)
2. Picard tools
3. fastp
4. GATK
5. SAMtools
6. Tablet

To check that these softwares are indeed installed, try running them (e.g. `./bwa` from before). They should say something interesting, rather than "command not found".