Unix/Linux Basics

Thomas Girke Tutorial

Lab: Peter Morrell

Chaochih Liu

August 14, 2015

Learning Outcomes

Using powerful Linux command-line utilities.

Learn commonly used Linux commands.

Servers at the Minnesota Supercomputing Institute (MSI) run Linux

- Need to log-in into the remote Linux shell

Logging-In

Mac or LINUX

To log-in to the remote Linux shell, open the terminal and type:

```
ssh <your_username>@<host_name>
```

Where `ssh` stands for secure shell and `host_name` is the remote server's domain name (i.e. login.msi.umn.edu). You will then be asked to enter your password, simply type it and press enter.

The format you will use to log-in to MSI at the University of Minnesota is:

```
ssh x500@login.msi.umn.edu
```

For the purposes of this tutorial we will only work from Macs, if you have questions

Why GNU/Linux?

- Software costs $0
- Advanced Multitasking
- Remote tasking (“real networking”)
- Multiuser
- Easy access to programming languages, databases, open-source projects

Why GNU/Linux?

- Software freedoms
  - Free to use for any purpose
  - Free to study and modify the source code
  - Free to share
  - Free to share modified versions
- No dependence on vendors
- Better performance
- More up-to-date
- Many more reasons…

Basics

Things to keep in mind throughout this tutorial

Remember the UNIX/LINUX command line is case sensitive!

All commands in this manual are printed in gray code boxes.

The hash (pound) sign # indicates the start of comments for commands.

The notation <…> refers to variables and file names that need to be specified by the user. The symbols < and > need to be excluded.

Unix/Linux Help

One very useful resource is knowing where to get help. You don't have to memorize every single UNIX/Linux command, but you do need to know where to get help.

You can look up what any UNIX/Linux command does by using the `man <something>` command. To exit general help, press the `q` key.

For example, let's look at what the `wc` command does:

```
man wc   # manual on program 'word count' wc
```

To get a shorter version of help of `wc` type:

```
wc --help   # short help on wc
```

Online help: Google

Downloading Example Data

To download files, you must have the url of the raw format of the file. Use the command `wget` to download the files.

You can download files by copying and pasting one url at a time

```
wget https://raw.githubusercontent.com/vsbuffalo/bds-files/master/chapter-07-unix-data-
tools/contam.fastq
```

Let's download another file from a different url

```
wget https://raw.githubusercontent.com/vsbuffalo/bds-files/master/chapter-07-unix-data-
tools/contaminated.fastq
```

Now we'll download multiple files from different links with `wget`.

Orientation

To view which directory you are in use:

```
pwd     # Get full path of present working directory
```

To view the present working directory use:

```
ls      # content of present working directory
ls -l   # ls plus additional info on files and directories
ls -t   # list files in chronological order
ls -lh  # view size of files in present working directory
```

Orientation

To change directories use:

```
cd <dir_name>       # switch into specified directory
cd                  # brings you to highest level of your home directory
cd ..               # moves one directory up
cd ../../           # moves two directories up (and so on)
```

The tilde symbol ~ gets interpreted as the path to your home directory. This will work the same anywhere on the command line:

```
echo ~       # view the full (complete) path of your home
find ~       # list all your files (including everything in sub-directories)
ls ~         # list the top level files of your home directory
du -sh ~     # calculate the file sizes in your home directory
```

Exercise 1.1: Files and Directories

Here are some basic commands before we jump into the exercise:

```
mkdir <dir_name>    # creates specified directory
rmdir <dir_name>    # removes empty directory
rm <file_name>      # removes file
rm -r <dir_name>    # removes directory including its contents but asks for confirmation
rm -rf <dir_name>   # removes directory including its contents with confirmation off

cp <name> <path>    # copy file/directory to specified location (-r to include content in dir)
mv <name1> <name2>  # renames directories or files
mv <name> <path>    # moves file/directory to specified location
```

For the exercise:

Make a directory called **GitHub** with the mkdir command:

```
mkdir GitHub
```

Copy and paste

The methods will differ depending on where you are:

In a **command line** environment:

- Cut last word with Ctrl+w
- Paste with Ctrl+y

In a non-command line **desktop** environment:

- Copy with Ctrl+c
- Paste with Ctrl+v

Command line <-> **desktop** exchange:

- Copy text out of the **command line** and into the **desktop** with Shift+Ctrl+c or Apple+c
- Paste text from the **desktop** into the **command line** with Shift+Ctrl+v or

Handy Shortcuts

Anywhere in Command Line:

The up and down arrow keys scroll through command history.

Typing `history` will show all the commands you have used recently.

Auto Completion:

Typing the beginning of something followed by TAB will complete the program_path/file_name.

```
<something-incomplete> TAB
```

Taking control over the cursor (the pointer on the command line):

Finding Files in Directories and Applications

Here are some important commands to know:

```
find -name "*pattern*"             # searches for *pattern* in and below current directory
find ~/GitHub -name "*fastq*"       # finds file named *fastq* in specified directory
find ~/GitHub -iname "*FaSTq*"      # Same as above, but case insensitive
```

Here are some additional useful arguments that would be good to know exist:

- -user <user name>
- -group <group name>
- -ctime <number of days ago changed>

```
find ~ -type f -mtime -2        # finds all files you have modified in the last two days
locate <pattern>                # finds files and directories that are written into update file
```

Finding Things in Files

Grep

grep searches **within files** whereas find searches **directories**

```
grep pattern file           # provides lines in 'file' where pattern 'appears'
                         # if pattern is shell function use single quotes: '>'

grep -H pattern             # -H prints out file name in front of pattern

grep 'pattern' file | wc    # pipes lines with pattern into word count 'wc'
                         # wc arguments:
                         #   -c: show only bytes
                         #   -w: show only words
                         #   -l: show only lines
                         # help on regular expressions:
                         #   $ man 7 regex
                         #   man perlre
```

Permissions and Ownership

List directories and files

```
ls -al  # shows something like this for each file/dir:
        #   drwxrwxrwx
        # d: directory
        # rwx: read, write, execute
        # first triplet: user permissions (u)
        # second triplet: group permissions (g)
        # third triplet: world permissions (o)
```

Assign write and execute permissions to user and group

```
chmod ug+rx my_file
```

To remove all permissions from all three user groups

Exercise 1.2: Useful Unix Commands

Commands that will be used in the exercise:

```
du -sh               # displays disk space usage of current directory
du -sh *             # displays disk space usage of individual files/directories
du -s * | sort -nr  # shows disk space used by different files/directories sorted by size
```

STDIN, STDOUT, and STDERR

By default, UNIX commands read from **standard input (STDIN)** and send their output to **standard out (STDOUT)**.

Example: STDIN and STDOUT

This will take the input of a program and output it to be written to a file:

```
ls -l              # lists (outputs) files in long listing format

ls -l > ls-l.txt    # takes output from above and inputs it to a file called 'ls-l.txt'
```

Note: When creating new files, it is easiest to avoid having spaces between names.
Use _ or - as spaces instead.

Example: STDERR

Redirections

Before we jump into redirections, I would like to review **wildcards**.

- Wildcards are denoted by * and it is used to specify many files (I'll discuss more details about this later and show an example).
- A few examples of formats are:
  - <beginning-of-filename>*
  - *<end-of-filename>
  - *<middle-of-filename>*

Now, we'll go back to redirections.

The following commands are redirections:

```
ls > file              # stores ls output into specified file
command < my_file      # uses file after '<' as STDIN
command >> my_file     # appends output of one command to file (will not overwrite file with same
```

Piping

Piping is another form of redirects.

It is a way to chain commands together

- Can take the STDOUT of one command and send it to STDIN of another
- Denoted by | symbol

Example:

```
find $(pwd) -name "name_of_file" | sort
```

Globbing

Globbing is denoted by * and is a wildcard.

It finds all that matches the surrounding text.

Example:

```
*.fastq         # matches all files that end with .fastq

WBDC_*          # matches all files taht start with WBDC_

*_2015_08-05*   # matches all files that have the following date in the file name
```

Command Substitutions

The process of using the output of one command as the argument for another.

Example:

```
find $(pwd) -name "*.sam"
```

- pwd command outputs full path to the current working directory (**p**ath to **w**orking **d**irectory)
- find command looks through pwd for filename(s) specified with double quotes

Sometimes you will see older syntax for command substitution that looks like this:

```
find $(pwd) -name "*.sam"
```

Exercise 1.3: Using find, chmod, and grep commands

Make sure you are in the directory ~/GitHub (this path may be different depending on where you created your GitHub directory earlier on).

```
#   Use 'find' command to locate all .fasta files
#   'pwd' means point working directory
#   'sort' sorts text line by line in alphabetical order
find $(pwd) -name "*.fasta" | sort
```

Check how many files are listed:

```
#   Use 'wc' command
#   '-l' option searches line by line
#   Use the up arrow to scroll through your history
#   so you don't have to re-type the entire command
find $(pwd) -name "*.fasta" | sort | wc -l
```

Exercise 1.3: Using find, chmod, and grep commands

Again, make sure you are in the ~/GitHub directory.

```
#   Use the 'find' command to locate all .fasta and .fastq files
find $(pwd) -name "*.fast*" | sort
```

Check how many files were found:

```
#   Use the 'wc' command to see how many files were found
find $(pwd) -name "*.fast*" | sort | wc -l
```

There should be 5 files with either .fasta or .fastq in their filenames.

Exercise 1.3: Using find, chmod, and grep commands

View permissions of the files downloaded:

```
#   Use the 'ls' command to do this
ls -l
```

Now we will change the permissions for the file contam.fastq. Let's assign read ®
and write (w) permissions for the user (u) and group (g).

```
chmod ug+rw contam.fastq
```

You will rarely want to assign permissions to world (o) but this depends on your specific project. World permissions give anyone either read®, write (w), and/or execute (x) permissions to your files

Exercise 1.3: Using find, chmod, and grep commands

We will now practice using the grep command.

```
#   Download the file we are using to ~/GitHub
wget https://raw.githubusercontent.com/vsbuffalo/bds-files/master/chapter-07-unix-data-tools/lengths.txt
```

View the beginning of the file called lengths.txt:

```
#   Use the 'head' command for this
head lengths.txt
```

Now, pull all chr1 lengths from the file:

Process Management

There are quite a few process management commands, one of the more useful ones is Ctrl+c to kill the process that is currently running in the foreground.

Here are a few more:

```
top     # view top consumers of memory and CPU
who     # Shows who is logged into the system
ps      # Shows processes running by user

ps aux | grep <user_name>   # Shows all processes of one user
ps ax --tree                # Shows the child-parent hierarchy of all processes
ps -o %t -p <pid>           # Shows how long a particular process was running

kill <process-ID>           # Kills a specific process
kill -9 <process-ID>        # NOTICE: "kill -9" is a very violent approach
                            # It does not give the process any time to perform cleanup procedures

renice -n <priority_value>  # Changes the priority value, which ranges from 1-19,
```

Non-Graphical Text Editors

These are typically used on the command line to make small edits to scripts. Below are some of the non-graphical text editors that can be used:

Vi and Vim

- Non-graphical (terminal-based) editor
- Vi is guaranteed to be available on any system
- Vim is the improved version of vi

Pico

- Simple terminal-based editor available on most version of Unix
- Uses keystroke commands but they are listed in logical fashion at bottom of screen

Nano

- A simple terminal-based editor which is default on modern Debian systems

Graphical Text Editors

These are often used when writing large chunks of code and can be used on a GUI (Graphical User Interface). Below are a few options for graphical text editors:

Sublime2

- Graphical editor
- Customizable
- Cross Platform
- OS X, Windows, Linux
- Free unlimited trial period
- Can also purchase license

Visual Studio Code

- Graphical editor
- Cross Platform
- OS X, Windows, Linux
- F

Vim Manual

For this tutorial, we will focus on learning Vim. Vim has a larger learning curve but is a very powerful tool.

Basics:

```
vim my_file_name    # open/create file with vim
```

Once you are in Vim, here are some of the most important commands:

```
i       # insert mode - allows you to edit text
ESC     # the escape key allows you to exit insert mode
:       # the colon key starts command mode at the bottom of the screen
```

Modifier Keys to Control Vim

Here are additional keys you should become familiar with in Vim:

```
:w      # save command (if you are in editing mode you have to hit ESC first!)
:q      # quit file, don't save
:q!     # exits WITHOUT saving any changes you have made
:wq     # save and quit
R       # replace mode
r       # replace only one character under cursor
q:      # history of commands (from NORMAL MODE!)
        # to re-execute one of the commands, select and hit enter
:w new_filename     # saves into new file
:#      # colon followed by a line number, jumps to specified line
```

Additional Vim Commands

Below are additional Vim commands that will make your life a lot easier if you keep them somewhere you can reference. It is not necessary to memorize these.

```
:s/txt1/txt2    # replace 'txt1' with 'txt2' on the current line
:%s/txt1/txt2   # replace 'txt1' with 'txt2' in the whole file
u               # undo last change
<CTRL>+r        # redo last undo
dd              # delete current line the cursor is on
$               # jump to the end of current line
^               # jump to the beginning of current line
/txt            # find all instances of 'txt' in file
<shift>+g       # jump to the bottom of the page
gg              # jump to the top of the page
```

Vim Help

Online help:

For help on the web, Google will find answers to most questions on **vi** and **vim**. There is also an Animated Vim Tutorial. There are also plenty of Vim commands cheat sheets available online.

Help from Command Line

```
vimtutor    # open vim tutorial from shell
```

Help in Vim

```
:help           # opens help within vim, hit :q to get back to your file
:help <topic>   # opens help on specified topic
: <up down keys>  # like in shell, you get recent commands
```

More Vim

- Moving around in files
- Line Wrapping and line numbers
- Spell checking & dictionary
- Enabling syntax highlighting
- Deleting things
- Search in files
- Replacements with regular expression support
- Printing and inserting files
- Convert text file to HTML format
- Shell commands in vim
- Use Vim as Table Editor

To learn more about vim, reference the Thomas Girke tutorial.

Useful Shell Commands

Below are some useful commands to know exist.

```
cat <file1> <file2> <cat.out>       # concatenate files into output file 'cat.out'
```

```
cmp <file1> <file2>         # tells whether two files are identical
diff <file1> <file2>        # finds differences between two files
diff -y <file1> <file2>     # shows differences between two files side by side
```

```
head -<number> <file>       # prints first n lines of a file
```

```
sort <file>                 # sorts single file, many files and can merge (-m) them,
                            # -b ignores leading white space
```

```
grep                        # searches within a given input file and
```

Exercise 1.4: Text Viewing

Now we will use less to view entire contam.fastq file. The less command will show you the entire file and you can scroll up and down through the file with your mouse or arrow keys. To quit out of less hit q on the keyboard.

```
less contam.fastq
```

Next, we will use more to view entire contam.fastq file . The more command allows you to scroll through the file in one direction using the space bar. To quit out of more hit q on the keyboard.

```
more contam.fastq
```

Try using cat to concatenate file and print contam.fastq file content to standard output

Loops

A way of executing a command or a series of commands on multiple cases multiple times.

Saves time

- Iterations over a range of data
- Running a command endlessly to monitor the status of a job
- Performing an operation on hundreds of files

Reduce chance of human error

- Let the computer take care of all steps

Fast

- Takes considerably less time than running commands manually

Examples on for-loops from Thomas Girke

Very simple loop to print the first line of all your .fasta files

```
for file in *.fasta; do head -1 $file; done
```

Renames many files *.old to *.new. To test things first, replace `do mv` with `do echo mv`.

```
for i in *.input; do mv $i ${i/\.old/\.new}; done
for i in *\ *; do mv "$i" "${i// /_}"; done         # Replaces spaces in files by underscores
```

Run an application in loops on many input files.

```
for i in *.input; do ./application $i; done
```

[illegible] fastacmd [illegible] BLAST [illegible] *input file [illegible]

Exercise 1.5: For-loops

A class of loop structures.

Used for iterations through data/files to perform the same operation multiple times on a set.

```
#   Here is an example of a for-loop written by Paul Hoffman, another member of the Morrell Lab
#   This for-loop renames all FASTQ files in a directory with the extension ".fq"
#   It creates a variable that is just the name of the file without the extension (basename)
#   And uses that as the base for the rename procedure
#   This happens for each FASTQ file iteratively
for file in `ls *.fastq`
do
    f=`basename $file .fastq`
    mv $file $f.fq
done
```

Check the directory contents with ls and you should see all the files that originally

Exercise 1.6: While-loops

Another class of loop structures.

Used for:

- Conditional (i.e. yes/no, true/false statements) looping
- Running a command given something that is *either* true or false

```
#   Here is an example of a while-loop written by Paul Hoffman, another member of the Morrell Lab
#   This loop adds one to a number until the nubmer is 10 or greater
#   We start by defining a variable as the number 1
#   and as long as the variable is less than 10, we add one to the variable
num=1

while [ $num -lt 10 ]
do
    num=$[ $num + 1 ]
    echo $num
done
```

Exercise 1.7: If/then Statements

Different lines of code can be executed depending on whether or not something tests true or not.

```
#   Here is an example of an if/then statement written by Paul Hoffman, another member of the Morrell Lab
#   The condition is whether or not SAMTools is installed
#   If SAMTools is installed, 'THEN' indexes the fasta file
#   'ELSE' tells us that SAMTools is not installed
if `command -v samtools > /dev/null 2> /dev/null`
then
    echo "SAMTools is installed"
    samtools faidx reference.fasta
else
    echo "SAMTools is not installed"
fi
```

Secure Copy Between Machines

General Syntax:

```
scp source target       # Use form 'userid@machine_name' if your local and remote user ids are different
                        # If they are the same, you can use only 'machine_name'
```

Examples:

```
scp user@remote_host:file.name  # Copies file from server to local machine (type from local
                                # machine prompt). The '.' copies to pwd, you can specify
                                # here any directory, use wildcards (*) to copy many files.

scp file.name user@remote_host:~/dir/newfile.name   # Copies file from local machine to server

scp -r user@remote_host:directory/ ~/dir    # Copies entire directory from server to local machine
```

Archives

Single package containing multiple files within themselves.

They make distributing packs of files easy

- For example, distributing software

There are many kinds that are often called packages

- Most common is tarball

Tarballs

A form of archives

- Created by tar utility

Easy to create and nearly universal

Standard file format: .tar

Compressed file formats:

- Gzip: .tgz
- Bzip2: .tbz

Compressed Files

Compressed files:

- Take up less space
- Are unviewable while compressed
- Different commands are used to decompress/compress different compression formats
- 3 different compression utilities used: gzip, zip and bzip2

Common Compression Formats

gzip	zip	bzip
.gz	.zip	.bz2
.tgz	.tar.zip	.tar.bz2
.tar.gz		.tbz
		.tbz2

How to Decompress Files

Decompress gzipped files:

- Use gzip -d to decompress gzipped files
- Use tar -xvzf to decompress gzipped tarball

Decompress bzip2 files:

- Use bzip2 -d to decompress bzip2 files
- Use tar -xvjf to decompress bzipped tarball

Decompress zipped files:

- Use unzip to decompress zipped files
- Use unzip then tar -xvf to decompress a zipped tarball

PATH

List of directories that the shell will search when you type a command.

Where software is installed.

```
echo $PATH      # view the PATH variable

export PATH=$PATH:<path to directory>   # You can edit PATH variable
                                  # Edit is not permanent
                                  # unless you put it in
                                  # your .bash_profile
```

Exercise 1.8: Practicing Unix/Linux Commands

```
#   In the directory ~/GitHub
#   Use 'find' command to locate all '.fasta' and '.fq' filenames
find $(pwd) -name "*.f*" | sed
```

Exercise 1.8: Practicing Unix/Linux Commands

Use ls to check your directory is there. Now go into the directory.

```
cd Itasca
```

Use ls again if you want to check the files in the directory (There should be none so far). Now we will move the .fasta and .fq files to ~/Itasca. First we will go into the directory our files are located in.

```
cd ~/GitHub
```

Then we will use globbing to move all the files at once instead of one at a time. We will use the mv command followed by the files we want to move. Then we will specify the directory we want to move the files into.

```
mv *.f* ~/Itasca
```

Resources

Thomas Girke's Linux tutorial

The answer key to today's *exercises* can be viewed and downloaded from my Gist repository.