

# Harvard Applied Mathematics 205

## Introduction to the command line

Danyun He, Yue Sun, Catherine Ding, and Chris Rycroft

September 10, 2021

# The Linux terminal

Has a history going back many decades. First version that we might recognize is the Unix shell, developed at Bell Labs in 1971.

Through 1980's Unix was proprietary software. Free alternatives such as Linux and BSD<sup>1</sup> became widespread in the early 1990's.

Linux and BSD have the same core functionality as Unix. Collectively they are referred to as “Unix-like operating systems.” All have a very similar terminal environment.

---

<sup>1</sup>Berkeley Software Distribution

# The Linux terminal

Since 2001 MacOS is also based on a Unix-like operating system, and has the same terminal environment.

Nowadays, a wide range of servers (Unix/Linux/Mac) around the world are running the same terminal environment.

(Windows has its own terminal environment with different syntax. But you can remotely access servers from Windows just the same. In addition, Cygwin (<https://cygwin.com/>) provides Linux-like functionality on Windows.)

# The Linux terminal: a powerful and useful tool

Allows you to remotely log into to servers and control almost all aspects of them.

Remote access is done via SSH (Secure Shell), an encrypted communication protocol for connecting to another computer.

Can copy and reorganize files, install programs, run simulations, *etc.*


Since it's a minimal interface, the data requirements for running a remote session are tiny. You can run it from anywhere, e.g. your phone, or halfway around the world in a coffee shop.

# SSH software

We will be using a Linux server in the Amazon Cloud. To access it you will need SSH:

- If you have a Linux computer, then SSH is built into the terminal program.
- If you have a Mac, SSH is built into the Terminal app that comes as standard.<sup>2</sup>
- Recent versions of Windows have SSH built in. You can also use the PuTTY client.

---

<sup>2</sup>iTerm is an alternative that is recommended by some. 

# Basic terminal commands

- **cd**: changes the working directory  
e.g. `cd path/to/directory`  
`cd` or `cd ~` leads to home directory  
`cd ..` navigates up one directory
- **cp**: copy files or directories  
e.g. `cp filename new_filename`  
`cp filename directory`  
`cp -R source_directory destination_directory`  
-R recursively
- **clear**: clear the terminal screen

# Basic terminal commands

- **ls**: list directory contents  
e.g. `ls -lh`  
-h human readable  
-l shows details of files
- **mkdir**: make directories  
e.g. `mkdir directory_name`
- **mv**: move or rename files  
e.g. `mv filename new_filename`  
`mv filename destination_directory`
- **\***: wildcard to match any file
- **?**: wildcard to match any single character in a filename

# Basic terminal commands

- **open**: open files  
e.g. open filename
- **pwd**: print working directory
- **rm**: remove files or directories  
e.g. rm filename  
rm -r directory  
-r recursive  
-i interactive  
-f force



# Basic terminal commands

- **touch**: create a new blank file or update date stamp if it exists  
e.g. touch filename
- **more/less**: display contents of a file, faster load time with large files  
e.g. more filename/ less filename  
+num specify starting line number  
scroll to navigate text  
press q to exit  
advantage of less: can do forward search (/pattern) or forward search (?pattern), press n to go to next match
- **man**: read the manual page for a command e.g. man ls

# Basic terminal commands

- **cat**: concatenate and print files  
e.g. `cat file1.txt file2.txt > newfile.txt`  
`cat file1.txt > file2.txt`  
`cat file1.txt >> file2.txt`
- **echo**: display arguments  
e.g. `echo "some text"`  
`\b` backspace  
`\n` new line  
`\t` horizontal tab space  
`\v` vertical tab space
- **grep**: search text for a pattern  
e.g. `grep "pattern" filename`  
`-i` case insensitive  
`-R` recursively search all files in the current directory and the subdirs

# Basic terminal commands

- **du/df**: estimate file space usage e.g. `du/df -option`
  - a write all files
  - c total usage
  - h human readable
  - df display more information
- **scp/rsync**: copy files/directories between local system and remote system
  - e.g. `scp/rsync options source destination`
  - `scp/rsync username@remotehost:filename /local/directory`
  - `scp/rsync -r username@remotehost:/remote/dir/ /local/dir/`
  - v verbose
  - z compress
  - h human readable

# Basic terminal commands

- **vim** editor

- press i to enter the insert mode and insert text

- press to esc to command mode

- :w write

- :q quit

- :set number

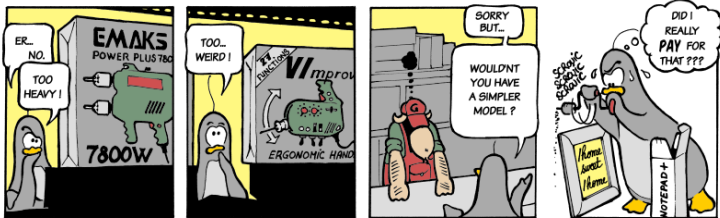
- other text editors: **nano**, **vi**, **emacs**

- **top**, **htop**: monitor the server's process in real time

# Vim versus Emacs

Vim and Emacs are two editors for the Linux terminal that have been around for a long time

Emacs is viewed comprehensive and powerful, whereas Vi is viewed as lightweight and highly customizable



# Batch processing

## ■ **for:** loop over multiple instances

```
altair:~% for a in d e f
for> echo $a
d
e
f
altair:~% for a in hello hi hola
for> do
for> echo $a

for> done
hello
bonjour
hi
hi
hola
hola
```

# Batch processing

## ■ **sed**: the Stream EDitor

```
altair:~% echo "I like polar bears" >foo
altair:~% sed 's/polar bear/panda/g' <foo
I like pandas
altair:~% sed -e 's/like/love/g' -e 's/I/We/g' <foo
We love polar bears
altair:~% sed -e 's/[lp]/-/g' <foo
I -ike -o-ar bears
```

# Exercise 1

1. Create a directory called `tmp` and change into it
2. Use terminal commands to make empty files called `x1`, `y1`, `z1`, `x2`, `y2`, `z2`, `x3`, `y3`, and `z3`—how can this be speeded up?



## Exercise 2(a)

1. Copy the code from `/shared/codes/ex2` into your home directory.
2. Change the prime numbers to “#” characters and the non-prime numbers to “.” characters.

## Exercise 2(b)

1. The file `/shared/codes/ex2b` contains a Caesar cipher with a shift of 5. Decode it.

## Exercise 2(c)

1. The file `/shared/codes/ex2c` contains a Caesar cipher. Decode it.

# Permissions

Files have three types of owners, with associated permissions.

- owner(u): the person who created the file
- group(g): a user-group can contain multiple users assigned with same permissions
- other(o): other users who can access the file but not belong to owner or group

You will see "-rwxrw-r-", the first "-" implies file type, then the following shows permissions given to owner, group, and the other. r (read), w(write), x(execute).

# Permissions

- **chmod**: modify file permissions  
e.g. `chmod u=rwx filename`  
= assign specified permissions  
+ add specified permissions  
- remove specified permissions
- **chown, chgrp**: change group ownership of a file or directory
- **sudo**: run with administrator privileges  
e.g. `sudo apt install`

# Programming

- **make**: compile a large program with makefile
- **git**  
e.g. git clone, git status, git add, git commit, git push, git checkout, git pull, git fetch, git reset, git reset
- **g++**: compilation  
e.g. g++ -o test test.c
- **python**: run a python program  
e.g. python test.py

# Programming

- **screen**: use multiple shell sessions from a single ssh session  
e.g. screen, start screen  
screen -S file  
screen -ls  
screen -d, or ctrl+A+d detach the screen  
screen -r screen\_id reattach the screen  
ctrl+A+n switch screens  
exit quit screen session
- **tmux**: similar functions as screen
- **time**: estimate the execution time