

# Linux

September 10, 2023  
Ramasamy Kandasamy

## 1. Unix shell

<code>pwd</code>	Print working directory
<code>cd</code>	Change directory.
<code>./</code> , <code>../</code>	Relative paths to current and parent dir.
<code>mkdir</code>	Make directory
<code>mkdir -p</code>	Make directory and necessary parent dir.
<code>touch</code>	Creates text file.
<code>ls</code>	Options: <code>-l</code> al
<code>du -h dir</code>	Gives size of all directories in <i>dir</i>
<code>du -sh dir</code>	Gives size of <i>dir</i> .
<code>df -h</code>	Gives information about disk usage.
<code>tree</code>	List in tree form. eg: <code>tree dir</code>
<code>cp</code>	Copy files.
<code>mv</code>	To move files and rename files.
<code>rm</code>	Remove files permanently.
<code>rm -i</code>	Remove files interactively.
<code>rm -r, rm -R</code>	Remove files recursively. Use to delete folders.
<code>rm -f</code>	Force delete.
<code>chmod 777</code>	r-4,w-2,x-1. User, group,all.
<code>chmod xyz</code>	Eg <code>chmod u+w</code> . x = u : user, g : group, a : all. y = + : add, - : remove. z = r : read, w : write, x : execute.
<code>~</code>	Root directory.
<code>?,*, [A-Z]</code>	Wild cards.
<code>{}</code>	Expands combinatorially. Eg: <code>\$ mkdir mm10-{chr1,chr2,chr3}</code> Eg: <code>echo "...\$(...)"</code> Eg: <code>mkdir results-\$(date +%F)</code>
<code>alias x = "..."</code>	Store new commands. But in shell startup file eg <code>~/.profile</code> or <code>~/.bashrc</code> and it is temporary. Eg: <code>\$ today = "date + %F"</code> .
<code>jobs</code>	List all jobs. Use id in <code>[]</code> to bg,fg,kill.
<code>top</code>	Lists all ongoing processes. Better than <code>jobs</code> .
<code>fg,bg,kill</code>	Bring job to foreground, background, kill a job. Pause a job before bg.
<code>ctrl</code> + <code>z</code>	Pause a running job.
<code>ctrl</code> + <code>c</code>	Kill a running job.
<code>echo \$?</code>	Exit status,=0 when a program exits without an error.
<code>Cmd1 ; Cmd2</code>	Run Cmd2 irrespective of exit status of Cmd1.
<code>Cmd1    Cmd2</code>	Execute Prog2 only if Prog1 has failed (non-zero exit status).
<code>Cmd1 &amp;&amp; Cmd2</code>	Execute Prog2 only if Prog1 has succeeded (zero exit status).
<code>true, false</code>	Assign exit status as 0 or 1 respectively.
<code>basename</code>	Removes folder name from path and optionally suffix.
<code>-s</code>	Remove suffix. eg: <code>basename -s .fastq &lt;path&gt;</code>
<code>&amp;</code>	Run in background. eg: <code>nohup prog1 &amp;</code>

## 2. Unix data tools

<code>echo</code>	Process and print whatever follows.
<code>echo -e</code>	enable backslash escapes like <code>\\</code> , <code>\t</code> , <code>\n</code>
<code>cat</code>	Takes standard input or input from file and gives standard output.
<code>cat -n</code>	Output with line numbers.
<code>&gt;</code> , <code>&gt;&gt;</code>	Write and append, respectively, standard output to a file.
<code>2&gt;</code> , <code>2&gt;&gt;</code>	Write and append standard error to a file.
<code>2&gt;&amp;1</code>	Redirects std.err to std.out.
<code>&lt;</code>	Take input.
<code> </code>	Pipe
<code>tee</code>	Eg: <code>prog1 in.txt   tee intermediate.txt   prog &gt; result.txt</code>
<code>/dev/null</code>	Fake file, black box.
<code>head -n x</code>	Print first x lines.Default: 10 lines.
<code>tail -n y</code>	Print last y lines.
<code>wc</code>	Word count. Outputs number of words, lines and characters.
<code>wc -l</code>	Outputs only number of lines.
<code>tr</code>	Translate. Eg: <code>tr ':' '\t'</code> .
<code>less</code>	Pager. Commonly used commands:
<code>Space</code>	Next page.
<code>b</code>	Previous page.
<code>g</code>	First line.
<code>G</code>	Last line.
<code>j</code>	Down (One line at a time).
<code>k</code>	Up (One line at a time).
<code>/&lt;pattern&gt;</code>	Search down for a pattern.
<code>?&lt;pattern&gt;</code>	Search up for pattern.
<code>n</code>	Repeat last search downward.
<code>N</code>	Repeat last search upward.
<code>cut</code>	To extract specific columns.
<code>-f x</code>	Extract columns x.
<code>-f x-z</code>	Extract range of columns.
<code>-f w,x-z</code>	Extract w and x-z. Cut cannot reorder column.
<code>-d</code>	Specify delimiter eg: <code>-d</code> ", ". Default delimiter is tab.
<code>column -t</code>	To visualize columns of data. Usually data is piped to <code>column -t</code> .
<code>-s</code>	Specify delimiter using <code>-s</code> ", ". Default: tab.
<code>grep</code>	Use as <code>grep "&lt;pattern&gt;" file</code> . Quotation around the pattern is not necessary but it is safe. If the pattern contains quote then use single quotes eg: <code>grep '..."..."'</code> .
<code>-i</code>	Case insensitive.
<code>-E</code>	To use regular expressions in grep.
<code>^</code>	Look for pattern in the beginning of line. Eg: <code>"^#"</code>
<code>-w</code>	Matches the entire word surrounded by space.
<code>-v</code>	Returns only lines that do not match the pattern.
<code>-o</code>	Return the exact matching pattern.
<code>-c</code>	Count how many lines match a pattern.
<code>-B1</code>	Print one line of context before the matching line.
<code>-A2</code>	Print two lines of context after the matching line.
<code>-C</code>	Context before and after the matching line.(Doesn't work?)

<code>sort</code>	Sorts alphanumerically by line.
<code>-ka,b</code>	Sorts w.r.t to columns a to b.
<code>-k2,2n</code>	Treats columns 2 as numeric and sorts w.r.t to columns 2.
<code>-t</code>	Specify delimiter eg: <code>-t</code> ", ". Default = tab.
<code>-s</code>	Stable sort. Do not reorder lines in file if the sort rank is equal.
<code>-c</code>	Check if the file is already sorted.
<code>-r</code>	Reverse sort.
<code>-V</code>	Understands numbers inside string. Eg <code>chr22</code> .
<code>-S</code>	Specify memory to be used. Eg: <code>-S 2G # Use 2 GB</code> , <code>-S 50% # Use 50% of memory</code> .
<code>--parallel</code>	to use parallel processing.
<code>uniq</code>	Usually used along with sort as : <code>sort ...  uniq</code> .
<code>-i</code>	Case insensitive.
<code>-c</code>	Count occurrences next to the unique lines.
<code>-d</code>	Return line with duplicates.
<code>join</code>	Combine data based on a common column. Eg: <code>join -1 a -2 b file1 file2</code> . a and b represent two columns common to file1 and file2.
<code>-a</code>	If some elements of common column are missing from one file. Use this flag to show all elements of common column from superset file.

Checkout [hexdump](#)

## 3. Advanced

<code>(...;...) ...</code>	Subshell: Both commands separated by a semi-colon are processed independently and piped in parallel to next step.
<code>mkfifo</code>	Create a named pipe. Eg: <code>mkfifo fqin</code> . Treat named pipe like any other file. But the input and output is piped. While using named pipe nothing is written on the disk.
<code>&lt;(...)</code>	Process substitution, like anonymous named pipe. Eg: <code>program --in1 &lt;(...) --in2 &lt;(...)</code>
<code>&gt;(...)</code>	Write output to anonymous named pipe. Eg: <code>program --out1 &gt;(...) --out2 &gt;(...)</code> .
Directory depth	To trim the path shown in terminal: Add the following to <code>.bashrc</code> . <code>PROMPT_DIRTRIM=1</code> 1 indicates a depth of 1.

## 4. Working with remote machines

### 4.1. SSH

- Usage  
`$ ssh host`  
`$ password:`
- Examples of host  
192.162.82.120  
bioclust.myuniversity.edu  
darwin@192.162.82.120

darwin@bio.univ.edu

#### Options

-v verbose. Verbosity can be increased by: -vv or -vvv.  
-p port. Eg: ssh -p 5043 cdarwin@bio.univ.edu  
Default port is 22

- **Using alias:** To use alias create the file ~/.ssh/config and store server as info as below. Host bio\_serv

```
HostName 190.512.171.29
User cdarwin
Port 50434
```

Also applies for Rsync and scp

- **SSH keys:** SSH key to connect without password. Eg:  
\$ ssh-keygen -b 2048

This command request the following:

- File to save the key. By default this is:  
/Users/username/.ssh/id\_rsa NOTE: This file is the private key.
- Passphrase:  
Not necessary but good to use.

Private key: ~/.ssh/id\_rsa

Public key: ~/.ssh/id\_rsa.pub

\$ chmod 400 id\_rsa # restrict access to private key

\$ ssh-add

## 4.2. Establishing a server

Use "Open SSH":

<https://help.ubuntu.com/lts/serverguide/openssh-server.html>

My IP address: hostname -I

List of logins to the server: sudo less /var/log/auth.log

## 4.3. nohup

nohup runs program un-interrupted.

The execution continues even when the terminal is closed or connection to remote machine is lost.

- Just add nohup just before the command.  
Eg: nohup prog1
- Usually nohup is in the background.  
Eg: nohup prog1 &

Unfinished: Tmux.

## 5. Networking

wget url	Download file from http or ftp.
-- accept, -A "..."	Only download files matching this criteria. Eg: "**.fastq"
-- reject, -R	Similar to above
--no-directory, -nd	Don't download directory structure.Only files.
--recursive, -r	Don't move above parent directory. This is important to avoid downloading unnecessary data.
--no-parent, -np	Don't move above parent directory. This is important to avoid downloading unnecessary data.
-O	Output filename.
-e robots=off	To not want wget to follow 'robot.txt'. See: <a href="#">This answer</a>

Other options: -limit-rate, -user=user, -ask-password

curl url > file	Redirect output to file.
curl -O <file>	download to file.
-L,--location	Download ultimate page and not the redirect page.

Curl can also download form SFTP and SCP. Also checkout RCurl and pycurl.

rsync	Usage: rsync source destination.
-r	Recursive to copy directories. Book doesn't use this. But I had to use this when I use rsync with pendirve.
-a	Enable archive mode.
-z	Enable file transfer compression.
-v	Make progress verbose.
-e ssh	If one of the directory is in remote host then have to use this option.Eg: \$ rsync -e ssh ./dir/ url:/home/...

Trailing slash in the source in rsync is meaningful. Eg rsync ./dir/ copies the contents of dir whereas rsync ./dir copies the entire directory. Rsync is use to synchronize directories but if you want to just copy one file then scp is enough. eg :

\$ scp file url:/home/...

Checkout Aspera Connect, ncbi sra-toolkit

shasum	Calculate checksum using SHA-1. Can be used to find checksum of many files and store the result in a text file. Eg: shasum *.fa > chksm.sha
-c	Validate the files. Eg: shasum -c chksm.sha.
sum	Checksum program used by Ensemble.
diff -u	Outputs a diff file that shows difference between two files. Eg: diff -u file1 file2

## 6. File compression

tar	Tape archive
-cf	To make tar file form a directory tar -cf dir.tar /dir. -c alone does not work for me (Why?), so use -cf
-tf	View contents of an archive.
-tvf	View contents, verbose.
-xf	extract.
zip -r	Compress. zip -r file.zip /dir
unzip -l	View contents. unzip -l file.zip
unzip	Decompress. unzip file.zip
gzip	Eg: gzip filename. gzip can only compress a file and not a directory. To compress a directory first make a .tar file and then compress that.
gunzip	To unzip .gz files.
-c	Output to standard output. Eg: gzip -c file1 > file.gz. Eg: gzip -c file2 >> file.gz. and gunzip -c.
bzip2	Works like gzip. Higher compression, but slow. File extension .bz2

Checkout compress/uncompress. Also: zgrep, zcat, zless, zdiff etc.

## 7. Awk

Format: awk pattern {action} input1.txt input2.txt

awk -f file.awk input.txt.

Record = row. Column = fields.

-F	Input field separator. Eg: awk -F"," input.txt. Defaule field separator = tab.
-f	Take input from file. Eg: awk -f file.awk input.txt.
(...) && (...)	Use logical operators. See below.
\$n /.../	Use regular expression between slashes.
/.../,.../	Specify range. Works only with regex (with double slash) .
BEGIN{...}	Eg: awk 'BEGIN{...} ... {...} END{...}'
END{...}	

Awk operations: +,-,\*,/,%,^.

a ... b. Replace "...": ==, !=, <, >, <=, >=, ~, !~, &&, ||, !a

Field separators: FS,RS,ORS.

Awk variables: NF, NR (Record number accumulates between files.), FNR(Resets record number after every files.).

Example awk script file

```
awk -f script.awk plasmids.tsv
BEGIN{FS="\t";ORS="\t";x=0}
/[Cc]re/{
x+=1;
print x,$1,$2}
END{print "There are " x "plasmids with Cre"}
```

Checkout BioAwk.

Checkout control flow.

8. Sed

sed 's/target/replacement/flag'

- e to Chain commands.Eg: sed -e 's://~' -e 's/-/~'.
- E Use extended POSIX.
- g Global flag. Usually sed replaces only the first occurrence in a sentence. Use global flag to replace all occurrences.
- i To make the search case insensitive.

9. Regular Expression

Single character meta characters

- . Match any single character.
- [ ] Match any single character between []. Eg: [at] match "a" or "t".
- [^] Match any single charcter except on between [].
- [0-9] Any number between 0 and 9. Eg: 0-3a-cz] equals [123abcz].
- (...) Grouping. eg: (AT)+ or (GLY ) {2,}.

Quantifiers

- ? Match preceding character zero or one time.
- \* Match zero or more time.
- + Match one or more time.
- {n} Match n times.
- {n,} Match atleast n times.
- {a,b} Match atleast a times, atmost b times.

Anchors

- ^ Match the start of a line.
- \$ Match end of a line.
- \< Match beginning of word.
- \> Match at the end of word.
- \b Match either beginning or end of word.
- \B Match any character not at the beginning or end.

Character class

[:alnum:], [:digit:], [:alpha:], [:upper:], [:lower:], [:blank:], [:space:], [:punct:] and [:print:].

Use backslash as escape character.

- \s white space character. What it includes depends on the flavour of regex.
- \d Add digits.
- \w Word character, matches [A-Za-z0-9\_]

| as OR logical operator: (GLY|GLN). "one and/or two" is equal to "(one and)| (or two)".

"one (and/or) two" is "one and two" or "one or two".

Back references: () : Memorizes the match for regular expression within parenthesis. Use \n to recall nth match.

10. Vim

Motion Usage: <num> <motion>

- h l One character left or right.
- j k One line up or down.
- w b One word forward or backwarks.
- e Similar to w but keeps the cursor at the end of the word.
- 0 Cursor to the begining of the sentence.
- \$ Moves cursor to the end of the sentences.
- G End of the file.
- gg First line.
- H Top of screen.
- M Middle of screen.
- L Botom of screen.
- <num>G Go to line <num>.
- [Ctrl] + f One screen forward.
- [Ctrl] + b One screen backward.
- [Ctrl] + G View position in the file.
- [Ctrl] + O Go to where you came from .
- [Ctrl] + I Opposite of [Ctrl] + [O]
- % Go to the corresponding opening or closing parenthesis.

Operators

- i INSERT mode
- a append, goes to insert mode
- A append from the end of the line.
- v visual selection, selection is stored in clipboard
- o open a line below
- O open a line above
- [Esc] Go to command mode
- d delete and also cut, ≡ [Ctrl] + [X]
- dd delete whole sentence
- x delete character under the cursor
- r replace the character under the cursor
- R replace until [Esc]
- c change: works equivalent to d followed by i
- y yank, copy
- P paste
- u undo most recent edit
- U undo all the changes in the line
- [Ctrl] + [R] Redo

Copy, paste, bookmark

- :xmy Move line x below line y.
- :x,ymz Moves lines between and including x and y below line z.
- :xty Copy line x below line y.
- :x,ytz Copy lines between and including x and y below line z.
- ma Set bookmark at current line. a ∈ [a-z].
- 'a Jump to bookmark a.
- :'a,'bco'c Copy lines between and including bookmarks a and b below bookmark c.
- :'a,'bco'z Copy lines between and including bookmarks a and b below line z.

Search and replace

- :/REGEX Find regular expression.
- n next search target
- N Previous search target
- :s/target/replace Similar to sed. Replaces target only in the current sentence and only once.
- :s/target/replace/g Replaces at all instance in the current sentence.
- :%s/target/replace/g Replaces through the entire file.
- :%s/target/replace/gc Ask for confirmation at each instance.

Save, write and Exit

- :q quit
- :q! quit without saving
- :w save the current file
- :wq or :x save and quite
- :w file write to file.
- :xyw file write lines between and including lines x and y to file.
- :! Execute shell command. Eg: :!pwd

:set

Usage: :set option. Eg: :set ic

- ic Case-insensititve search
- hls Highlight search
- number Show line number

To turnoff the option use no. Eg :set noic to turnoff ic

Etc

[Ctrl] + [D] for command completion.

[Tab] for filename completion.

For further setting: ~/.vimrc

Help:

[F1]

:help

:help w

:help user-manual

Default settings: Set default settings in ~/.vimrc

Create this file if it does not exist.

Example .vimrc file:

syntax on  
colorscheme desert  
set number  
set hls

# 11. Shell scripting

## Modifying PATH

Add a directory to path: Append one of the following files.

~/.profile or ~/.bash\_profile

with the following line:

PATH=\$PATH:<directory>

Eg: PATH=\$PATH:\$HOME/scripts

## Header

#!/bin/bash	Shebang
set -e	Terminates script if there is non-zero exit status.
set -o pipefail	If a program in the pipe fails the entire pipe returns non-zero exit status.
set -u	Terminates for undefined variables.

## Variables

sample="CNTRL"	Assignment, no space around "="
echo \$sample	
echo \${sample}_aln	Use curly braces while concatenating a variable with additional text.
mkdir "\${sample}_aln"	Quoting variables prevents commands from interpreting spaces and special variables.
echo \${#sample}	Length of the variable sample

## Command-line arguments

\$0 Script name  
\$1 First argument  
\$n n<sup>th</sup> argument.  
\$# Number of arguments not including \$0.

Example:

```
#!/bin/bash
echo "script name: $0"
echo "first arg: $1"
echo "second arg: $2"
echo "There are $# input arguments"
```

## 11.1. Conditionals

Format

```
if [ <conditon-statement> ]
then
if-statements
elif
then
elif-statements
else
else-statements
fi
```

Example:

```
if [ $# -lt 3 ]
then
echo "There are less than 3 arguments"
fi
```

In bash 0 is true/success, anything else is false/failure

String and integer comparison

-z str	str is null string.
str1 == str2	str1 and str2 are identical.
str1 != str2	
int1 -eq int2	int1 and int2 are equal.
int1 -ne int2	
int1 -lt int2	
int1 -gt int2	
int1 -le int2	
int1 -ge int2	
-o	Logical OR.
-a	Logical AND.

if conditional can also be used to depend on exit status. Eg:

```
if grep "pattern" file1.txt > /dev/null && grep "pattern"
file2.txt > /dev/null/
then
echo "found pattern in file1.txt and file2.txt"
fi
```

```
if ! grep "pattern" file1.txt > /dev/null
then
echo "pattern not found in file1.txt"
fi
```

## Testing files and dirs

List of test expressions.

-d dir	dir is a directory
-f file	file is a file.
-e file	file exists.
-h lind	link is a link.
-r file	file is readable.
-w file	file is writable.
-x file	file is executable.

Example

```
test -d dir ; echo $?

test -d dir1 -o -d dir2; echo $?
```

Exit status would be 0 if the directory dir exists.

Example:

```
if ! test -d $1
then
mkdir $1
fi
```

Above script is equivalent to the following.

```
if [ ! -d $1 ]
then
mkdir $1
fi
```

## 11.2. Arrays and For loop

### Manual creation

```
$ sample_names=(zmaysA zmaysB zmaysC)
$ echo ${sample_names[0]}
zmaysA
$ echo ${sample_names[@]}
zmaysA zmaysB zmaysC
$ echo ${#sample_names[@]}
3
$ echo ${!sample_names[@]}
0 1 2
```

### Array creation using command substitution

```
samples=$(cut -f3 samples.tsv)
file_names=$(ls)
```

### Array of number sequence

```
seq 0 0.1 1 # seq start step end
s=$(seq 0 0.1 1)
```

\${arr[i]}	(i-1) <sup>th</sup> element of array.
\${arr[@]}	All the elements of arr.
\${#sample_names[@]}	Length of arr.
\${!sample_names[@]}	Returns an array containing the index of elements in arr.

## 11.3. For loop

```
for name in ${file_names[@]}
do
process.sh $name
done
```

```
for name in ${file_names[@]}; do
process.sh $name
done
```

```
for name in ${file_names[@]}; do; process.sh $name; done
for i in $(seq start step_size end);
do
process.sh $i
done
```

## 11.4. Find, exec and xargs

<b>find</b>	Usage: find <folder> -name "<pattern>". Eg: find . -name foo.sh.
<b>-name &lt;pattern&gt;</b>	Find <pattern> using same special characters as bash (*,?, [...])
<b>-iname</b>	Identical to -name but case-insensitive.
<b>-empty</b>	Matches empty files and folders.
<b>-type &lt;x&gt;</b>	Matches types x (f - file, d - directory, l - links).
<b>-size &lt;size&gt;</b>	Matches <size>. Eg: +50M ; Files larger than 50 MB Eg: -50M; Files smaller than 50 MB
<b>-regex</b>	Match regular expression. Use -E for extended POSIX.
<b>-iregex</b>	Case-insensitive.
<b>-print0</b>	separate results with null-byte and not new line. <b>Explain!</b>
<b>expr -and expr</b>	Logical AND.
<b>expr -or expr</b>	Logical OR.
<b>-not expr</b>	Logical NOT. Alternate: "!" expr
<b>(expr)</b>	Group a set of expressions.
<b>-exec</b>	Example: find . -name *.c -exec <prog1> {} \;. Execute <prog1> on all the found files. {} represents the found files. Mind the space between {} and \;

Examples using **expr**:

```
expr 2 + 3 # Space is required for expr
a=$(expr 2 + 3)
expr $x + 1
```

**expr** is similar to **let**, but only evaluate and not assign value to a variable.

### Arithmetic operations:

+, -, /, %	
*	Multiplication operator for <b>let</b>
/*	Multiplication operator for <b>expr</b>
<b>var++</b>	increment var by 1 used only in <b>let</b>
<b>var--</b>	increment var by 1 used only in <b>let</b>

### xargs

**xargs** takes input passed to it and uses as argument for another program.

Examples:

```
find . -name *.fastq | xargs rm # Uses all the inputs as arguments for rm
find . -name *.fastq | xargs -n 1 rm # Uses one input at a time after rm.
find . -name *.fastq | xargs -n 1 echo "rm -i" > delete-temp.sh
find . -name *.c | xargs basename -s ".c" | xargs -I{} gcc -o {}.o {}.c
```

**xargs** flags

<b>-n &lt;num&gt;</b>	Process num input(s) at time.
<b>-I{}</b>	Uses {} as a placeholder for input.
<b>-P &lt;num&gt;</b>	Parallel processing: use <num> processes.

## 11.5. Arithmetics

### let

Examples using **let**:

```
let x=1 #No space within expression
let x=x*2
let x++
let "x = x + 1" # Space OK within quotation.
```



## 12. Git

Setup git with the following commands:

```
$ git config --global user.name "Ramasamy Kandasamy"
```

```
$ git config --global user.email ".....@gmail.com"
```

Next command tells git to use color to indicate changes.

```
$ git config --global color.ui true
```

To change default text editor:

```
$ git config --global core.editor gedit
```

These commands create a .gitconfig file in home directory. Use \$ cat

```
~/.gitconfig to get current information.
```

---

Git command structure: git <subcommand>

```
git init Initialize git repository in a directory.
```

```
git clone To clone a git repository.
```

Eg:

```
$ git clone https://github.com/user/sth.git
```

```
$ git clone https://github.com/user/sth.git dir_name
```

```
$ git clone https://user@bitbucket.org/user/sth.git
```

---

Git consists of untracked files, tracked file, files staged for commit, and files committed to the repository.

git status	Gives three categories of files: untracked, tracked files that have been modified, files staged for commit.
------------	---

git add	Start tracking a file or stage a file for commit.
---------	---

-f	To stage a file not tracked, i.e. a file in .gitignore.
----	---

git commit	Commits all staged files to repository. --amend
------------	---

-a	This options tells git to automatically stage all modified tracked files in this commit.
----	--

-m "..."	Message is mandatory. If there is no message, git opens text editor to input message. Default text editor can be specified in git-config.
----------	---

git diff	Shows difference between current version and staged version. If there are no staged version, shows difference between last commit and current versions.
----------	---

-- staged	To see difference between staged version and last commit.
-----------	---

git reset	Unstage a file. Without a file name all staged files get unstaged.
-----------	--

git log	List all commits, commit message SHA-1 checksum etc. Options: --pretty=oneline, --abbrev-commit, --graph, --branches, -n2 : to view only latest two commits.
---------	--

git rm	Use these commands to rename or delete files.
--------	---

git mv	Using rm and mv will confuse git.
--------	-----------------------------------

.gitignore	Used to avoid certain files, fastq files for example, from being listed in untracked section of git status. Eg: \$ echo "*.fa" >> .gitignore.
------------	---

git ls-tree	List contents of tree object.
-------------	-------------------------------

Use to list all files in the latest commit.

Eg: git ls-tree -r master --name-only

---

To add a remote repository.

```
$ git remote add origin git@github.com:username/project.git
```

```
$ git remote add origin user@bitbucket....
```

git remote -v	Shows remote repository that connected to local repository.
git remote rm	Remove remote repository. Eg: git remote rm origin
git push	Use git push origin master to push main branch to origin (remote repository)
git pull	git pull origin master: similar to above.

**Resolving merge conflicts:** First git pull from remote repo. git status shows files with merge conflict. Open the file and resolve the conflict using guidelines provided.

Unfinished: Github SSH

git checkout	Restores file from HEAD. To restore a file from a specific commit. Use the commit SHA-1 ID. Eg git checkout 08ccd3b -- README.md
--------------	--

git stash	To temporarily store the changes and go back to HEAD.
-----------	---

git stash pop to restore changes stored in git stash.

git diff	git diff id1 id2 file to compare different version using SHA-1 ID.
----------	--

git diff HEAD~3 HEAD~4 : w.r.t to last commit.

git commit	To edit message in last commit.
------------	---------------------------------

--amend	Can also be used to modify files in previous commit, but I don't know how.
---------	--

---

git branch	Creates a new branch. It also lists all branches and indicate the branch that is used currently.
------------	--

-d	To delete a branch.
----	---------------------

-m	Rename a branch. Eg:
----	----------------------

git branch -m new-branch # Renames current branch.

git branch -m old-branch new-branch.

--all	To view hidden branches including remote repositories. For eg, /remote/origin/master is usually hidden. This functions like an actual branch but one cannot develop in this remote branch.
-------	--

git checkout	To jump between branches. Use branch name that you want to jump to.
--------------	---

git merge	To merge two branches go to the branch you want to merge to and use git merge <other branch>. Merge conflict can be resolved as described earlier. In fact the earlier merge conflict was between a local branch and a remote branch.
-----------	---

git push	New branch from local can be synchronized with remote using: git push origin branchname.
----------	--

git fetch	Used to synchronize my remote branch with remote repository. Eg: git fetch origin. To incorporate this to local branch use git merge.
-----------	---

**NOTE:** git pull is nothing but git fetch followed by git merge.

git checkout -b new-methods origin/new-methods

This command simultaneously creates and swithces a new branch using -b option. This local branch will push and pull to this specific remote branch.

git remote prune origin : To prune a stale branch in /remote branch.

## 13. Markdown

Text formatting:

- *\*italics\**
- **\*\*bold\*\***
- **\*\*\* bold italics \*\*\***
- \_\_underline\_\_
- \*underline italics\*
- **\_\_\*\*underline bold\*\*\_\_**
- **\_\_\*\*\*underline bold italics\*\*\*\_\_**
- ~~~~strikethrough~~~~
- Text coloring:  
<span style="color:blue"> blue text </span>

Heading, lists and links

- Itemized list: \* item 1 or + item 1 or - item 1
- Ordered list: Eg:
  1. red
  2. blue
  4. green # Here output automatically numbers it to 3
- Use # for Headers.  
# Header level 1  
## Header level 2  
Markdown supports upto 6 levels.
- <http://website.com/link>
- [link text](http://website.com/link)
- Insert figure  
![alt text](path/to/figure.png/)

Inserting code

- ``inline code``, Use backticks.
- Code block with tilde:  
~~~ Language (Optional used by pandoc to )  
code block  
code block  
~~~~~
- Codeblock with three backticks:  
```Language (Optional used by pandoc to )  
code block  
code block  
```

## 14. Pandoc

- **Markdown to HTML (simple version)**  
\$ pandoc -f markdown -t html README.md -o README.html
- **md to word**  
\$ pandoc -s README.md -o README.docx
- **Standalone:** -s. Necessary for syntax highlighting.  
To get list of languages: --list-highlight-languages

- **Box/shading for code:** Use `--highlight-style`. Eg:  
`--highlight-style tango` # Good for light shade.  
`--highlight-style breezedark` # Good for dark shade.  
`--list-highlight-style` # List of highlight themes.

## 15. Uncategorized

### Terminal shortcuts

<code>ctrl</code>	+	<code>W</code>	Delete from cursor to beginning of word.
<code>ctrl</code>	+	<code>U</code>	Delete from current cursor to start of line.
<code>ctrl</code>	+	<code>A</code>	Move cursor to beginning of line.
<code>ctrl</code>	+	<code>E</code>	Move cursor to end of line.
<code>ctrl</code>	+	<code>L</code>	Clear the screen.
<code>alt</code>	+	<code>F</code>	Move forward by word.
<code>alt</code>	+	<code>B</code>	Move backward by word.

## 16. WSL and windows CMD

### 16.1. Execute command prompt commands from WSL.

- Notepad:  
`notepad.exe`  
`notepad.exe temp.txt`
- File explorer:  
`explorer.exe`  
`explorer.exe .`
- Execute command prompt commands in WSL.  
`cmd.exe` *command-line-commands* Eg: Opening a windows program  
`cmd.exe /C start program_name file_name`  
Eg:  
`cmd.exe /C start SumatraPDF.exe`  
`mementopython3-english.pdf`

### 16.2. Open from command prompt

- Websites using edge or chrome.  
Edge: `start microsoft-edge`  
Edge: `start microsoft-edge:http://www.google.co.in/`
- MS-office apps.
- Other applications.

## 17. Using GUI in WSL

### 17.1. Installing XFCE

Under construction

Ref:

<https://www.youtube.com/watch?v=nKCe9UE-quA>

<https://www.shogan.co.uk/how-tos/wsl2-gui-x-server-using-vcxsrv/>

### 17.2. Running XFCE

**Open XLaunch app**

The following is just to open a windows with simple settings.

1. Double-click and open XLaunch app. You will see a dialog box for display settings.
2. Choose "One large window" and choose "-1" for Display Number. Click "Next".
3. Choose "Start no client". Click "Next".
4. Check "Clipboard", "Primary Selection", and "Native opengl". Click "Next".
5. Save the configuration if you want, or just click "Finish" to start the window.

**Launch xfce in WSL**

Execute the command `xfce4-session`. Ignore the warnings.

## 18. Incomplete:

NOTE: This cheatsheet does not include Bioconductor and GRanges. Ver2 has them. But I will split it to a different cheat-sheet, "Bioconductor and R"

- arithmetics in bash
- pandoc
- markdown syntax
- install packages
- make
- tabix
- SQL