Hacker Tools

August 13, 2024

Ramasamy Kandasamy

1. Command Line Tools

Files & Directories

`cd`	Change directory.
`pwd`	Print working directory
`ls`	Options: `-Ralh`
`tree`	List in tree form. eg: `tree` *dir*
`touch`	Creates text file.
`mkdir`	Make directory
`mkdir -p`	Make directory and necessary parent dir.
`cp`	Copy files.
`mv`	To move files and rename files.
`rm`	Remove files permanently.
`rm -i`	Remove files interactively.
`rm -r, rm -R`	Remove files recursively. Use to delete folders.
`rm -f`	Force delete.
`basename`	Removes folder name from path and optionally suffix.
`-s`	Remove suffix. eg: `basename -s .fastq` <*path*>
`~`	Home directory, aka `$HOME`.
`./ , ../`	Relative paths to current and parent dir.
`/dev/null`	Fake file, black box.
`chmod 777`	r-4,w-2,x-1. User, group,all.
`chmod xyz`	Eg `chmod u+w`. x = `u` : user, `g` : group, `a` : all. y = `+` : add, `-` : remove. z = `r` : read, `w` : write, `x` : execute.
`du -h` *dir*	Gives size of all directories in *dir*
`du -sh` *dir*	Gives size of *dir*.
`df -h`	Gives information about disk usage.

File compression

`tar`	Tape archive
`-cf`	To make tar file form a directory `tar -cf dir.tar dir`.
`-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`

TODO: chown, chgrp. `compress/uncompress`. Also: `zgrep`, `zcat`, `zless`, `zdiff` rsync
hexdump, checksums, diff (in text processing?).

Process Execution

`Cmd1 ; Cmd2`	Run Cmd2 irrespective of exit status of Cmd1.
`Cmd1 || Cmd2`	Execute `Prog2` only if `Prog1` has failed (non-zero exit status).
`Cmd1 && Cmd2`	Execute `Prog2` only if `Prog1` has succeded (zero exit status).
`(...;...)|...`	Subshell: Both commands separated by a semi-colon are processed independently and piped in parallel to next step.
`<(...)`	Process substitution, like anonymous named pipe. Eg: `program --in1 <(...) --in2 <(...)` .
`>(...)`	Write output to anonymous named pipe. Eg: `program --out1 >(...) --out2 >(...)`.
`mkfifo`	Create a named pipe. Eg: `mkfifo fqin`. Treat named pipe like any other file. But the input and output is piped. While using named pipe nothing is written on the disk.
`nohup`	Run a program without interruption.
`&`	Run in background. eg: `nohup prog1 &`

Process mangement

`jobs`	List all jobs. Use id in [] to bg,fg,kill.
`fg`	Bring a job to foreground.
`bf`	Resume a suspended process in the background.
ctrl + z	Pause a running job.
ctrl + c	Kill a running job.
`kill`	End a job.
`echo $?`	Exit status,=0 when a program exits without an error.
`htop`	User friendly tool to view running processes and resource utilization.

Etc

`?,*, [A-Z]`	Wild cards.
{}	Expands combinatorially. Eg: `$ mkdir mm10-{chr1,chr2,chr3}`
$()	Eg: `echo "...$(...)..."` Eg: `mkdir results-$(date +%F)` Eg: `$ today = "date + %F".`

Terminal customization

`alias x ="..."`	Store new commands. But in shell startup file eg `~/.profile` or `~/.bashrc` and it is temporary.
Directory depth	To trim the path shown in terminal: `PROMPT_DIRTRIM=1` 1 indicates a depth of 1.

2. System Tools

df -h View usage of all the mounted disk.

3. Text processing

`echo`	Process and print whatever follows.
`echo -e`	enable backslash escapes like `\\, \t, \n`
`cat`	Takes standard input or input from file and gives standard output.
`cat -n`	Output with line numbers.
`>, >>`	Write and append, respectively, standard output to a file.
`2>, 2>>`	Write and append standard error to a file.
`2>&1`	Redirects std.err to std.out.
`<`	Take input.
`|`	Pipe
`tee`	Eg: `prog1 in.txt | tee intermediate.txt | prog > result.txt`
`head -n x`	Print first x lines.Default: 10 lines.
`tail -n y`	Print last y lines.
`wc`	Word count. Outputs number of words, lines and characters.
`wc -l`	Outputs only number of lines.
`tr`	Translate. Eg: `tr ':' '\t'`.
`less`	Pager. Commonly used commands:
Space	Next page.
b	Previous page.
g	First line.
G	Last line.
j	Down (One line at a time).
k	Up (One line at a time).
`/<pattern>`	Search down for a pattern.
`?<pattern>`	Search up for pattern.
n	Repeat last search downward.
N	Repeat last search upward.
`cut`	To extract specific columns.
`-f x`	Extract columns x.
`-f x-z`	Extract range of columns.
`-f w,x-z`	Extract w and x-z. Cut cannot reorder column.
`-d`	Specify delimiter eg: `-d","`. Default delimiter is tab.
`column -t`	To visualize columns of data. Usually data is piped to `column -t`.
`-s`	Specify delimiter using `-s","`. Default: tab.

`grep`	Use as `grep "<pattern>" file`. Quotation around the patter is not necessary but it is safe. If the pattern contains quote then use single quotes eg: `grep '..."..".'`.
`-i`	Case insensitive.
`-E`	To use regular expressions in grep.
`^`	Look for pattern in the beginning of line. Eg: `"^#"`
`-w`	Matches the entire word surrounded by space.
`-v`	Returns only lines that do not match the pattern.
`-o`	Return the exact matching pattern.
`-c`	Count how many lines match a pattern.
`-B1`	Print one line of context before the matching line.
`-A2`	Print two lines of context after the matching line.
`-C`	Context before and after the matching line.(Doesn't work?)

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

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-interupted.
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`	
`--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: This answer

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 wherease `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. 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{...} END{...}`	Eg: `awk 'BEGIN{...} ... {...} END{...}'`

Awk operations: `+,-,*,/,%,^`.
`a ... b`. Replace "..." : `==,!=,<,>,<=,>=,~,!~,&&,||,!a`
Field separators: FS,RS,OFS, 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";OFS="\t";x=0}
/[Cc]re/{
x+=1;
print x,$1,$2}
END{print "There are " x "plasmids with Cre"}
```

Checkout BioAwk.
Checkout control flow.

7. 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 occurences.
`i`	To make the search case insensitive.

8. 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.

9. 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	Simillar 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	Simillar 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
```

10. 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^{th} 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"
```

10.1. 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
```

10.2. 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)th 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.

10.3. 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
```

10.4. Find, exec and xargs

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

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

`-n <num>`	Process `num` input(s) at time.
`-I{}`	Uses {} as a placeholder for input.
`-P <num>`	Parallel processing: use `<num>` processes.

10.5. 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.
```

Examples using `expr`:

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

`expr` is simillar to `let`, but only evaluate and not assign value to a variable.
Arithmetic operations:
+, -, /, %

*	Multiplication operator for `let`
/*	Multiplication operator for `expr`
`var++`	increment var by 1 used only in `let`
`var--`	increment var by 1 used only in `let`

11. 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`: simillar 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 guidlines provided.
Unfinished: Github SSH

`git checkout -- file`	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 --amend`	To edit message in last commit. 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 remote repository. Eg: `git fetch origin`. To incoporate 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.

12. 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
  ‘‘‘
  ```

13. 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.

14. Uncategorized

Terminal shortcuts

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

15. WSL and windows CMD

15.1. 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` *`commmand-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`

15.2. 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.

16. Using GUI in WSL

16.1. 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/

16.2. Running XFCE

Open XLaunch app

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

1. Doulble-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.

17. Incomplete:

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

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