Linux Bash Shell Cheat Sheet: Basic Commands

***** Remember: man <command> = shows manual page for any command**

Navigating the file system

ls = list files in current working directory
ls -l = 'long' listing of current working directory
ls -a = list all files, including 'hidden' ones, in current working directory
ls <dirName> = list files in named directory
ls -lh = Detailed list, Human readable
ls -l *.jpg = list jpeg files only
ls -l <fileName> = list the named file only
ls -F = list current directory, displaying symbols to indicate file types, e.g. '/' means directory, '*' means executable script

pwd = print working directory

cd = with no argument means change to the user's home directory
cd <dirName> = change directory
cd / = change to root directory
cd .. = change to parent of current working directory

Remember:

- '**/**' = root directory
- '**.**' = current working directory
- '**..**' = parent of current working directory
- '**../..**' = parent of parent of current working directory, etc.
- '**~**' = user's home directory

Basic file manipulation

cp <fileName1> <fileName2> = make a copy of file in current working directory
cp <filename> <dirName> = make a copy of the file in directory <dirName>
cp -R <dirName1> <dirName2> = copy and rename a directory
cp *.txt <dirName> = copy all files with extension '.txt' in current working directory to directory called <dirName>

mv filename.txt filename2.txt = rename a file
mv file.txt <dirName> = move file into directory <dirName>
mv <dirName1> <dirName2> = rename a directory

rm <fileName1> <fileName2> <fileName3> etc. = delete file(s)
rm -i <fileName> <fileName2> <fileName3> etc. = ask for confirmation before deleting each file
rm -f <fileName> = force deletion of a file (use this with great care!)

rm -r <foldername> = delete directory AND contents (use this with great care!)

touch <fileName> = create an empty file or update last access time of an existing file

mkdir <dirName> = create new directory called <dirName> within current working directory
mkdir <dirName>/<subdirName> = create directory called <subdirName> inside <dirName>. (Directory <dirName> must already exist).
rmdir <dirName> = remove empty directory (MUCH safer than rm –r)

Wild cards in file names

'*****' Matches one or more characters in file name e.g. *.txt means all files with .txt extension

'**?**' Matches a single character in file name e.g. test0?.txt means all file names starting with 'test0' and ending in '.txt' with a single character in between. It would NOT match test099.txt.

Basic Terminal Shortcuts (Useful when editing commands)

TAB = auto completion of file or command
CTRL + A = Move cursor to start of line
CTRL + E = Move cursor to end of line
CTRL + U = Delete to left of cursor
CTRL + K = Delete to right of cursor
CTRL + W = Delete word to left cursor
CTRL + Y = Paste (after CTRL U,K or W)
CTRL + D = Logout (less, more)

Chaining commands (pipes and filters)

Place '**|**' at the end of a command to pass the output to another one
For example: **du | sort -nr | less**

Useful filters:

cat <fileName> = print out content of file

head <fileName> = print out content of file from the top (default is 10 lines)
head -n <number> <fileName> = print out first n lines of file

tail <fileName> = print out content of file from the bottom (default is 10 lines)
tail -n <#oflines> <fileName> = print out last n lines of file

wc = word count:

- **wc <fileName>** = print out number of lines, words, and characters inside file
- **wc -l <fileName>** = print out number of lines in file
- **wc -w <fileName>** = print out number of words in file
- **wc -c <fileName>** = print out number of characters in file

cut = cut a part of a file:

cut -c 2-5 names.txt = cut the characters 2 to 5 of each line
-d (delimiter) (-d & -f good for .csv files)-f (# of field to cut)

sort = sorts the content of files:
sort <fileName> = sort alphabetically
sort -r <fileName> = sort in reverse
sort -n <fileName> = sort numerically instead of alphabetically
sort -R <fileName> = sort randomly

File Permissions

chmod = modify user access/permission – simple way

u = user
g = group
o = other

d = directory (if element is a directory)
l = link (if element is a file link)
r = read (read permissions)
w = write (write permissions)
x = eXecute (only useful for scripts and programs)

'**+**' means add a permission
'**-**' means remove a permission

For example: **chmod g+w someFile.txt** = give the current group permission to modify the file someFile.txt

chown = change the ownership of a file (you can only do this if you are the initial owner of the file – you can't change ownership of someone else's files unless you have 'super user' privileges)

Searching for files

find = the best file search tool (fast):

find . -name mappings.txt= starting from the current directory, search the directory tree to find the file called mappings.txt

find ./myfiles -name "*map*" = starting from the 'myfiles' subdirectory of the current directory, search for files whose names contain the word 'map'

find ~ -name "*text" = starting from the user's home directory, search for files whose names end with the word 'text'

find ~ -size +10M = starting from the user's home directory, search for files bigger than 10 Megabytes. (File sizes can be specified as M (Megabytes), K (Kilobytes) ,G (Gigabytes) etc.)

find /var/log -name "syslog" -type d = starting from directory /var/log, find only other files that are directories called ‘syslog’

find . -name “*test*” -type f = starting from current working directory, find only ordinary files whose names contain the word ‘test’

Searching for data within files

grep <someText> <fileName> = search for specified text in a file
-i = case insensitive
-I = exclude binary files
-v = search for lines that DON’T contain the specified text
For example:
grep -i -v fred names.txt = search the file names.txt case insensitively and return lines that do NOT contain the word ‘fred’
If the text you are searching for contains spaces or special characters, use quotes to protect it
For example:
grep “apples oranges” fruit = search for lines containing ‘apples oranges’ separated by a space in the file called fruit

grep with regular expressions:

grep -E ^<text> <fileName> = show lines that start with <text>

grep -E <0-4> <fileName> = show lines that contain numbers 0-4

grep -E <a-zA-Z> <fileName> = retrieve all lines with alphabetical letters

Job and process control

Viewing processes:

ps = Static process list

ps -ef = show all processes with full listing of process details

ps -u = show processes owned by current user

top = Dynamic process list – shows processes in descending order of computer resource usage (cpu, memory, etc.).
While in top:

- q to exit
- h to show the help
- k to kill a process

who = who is logged on and what they are doing

To run a job in the background: add a '&' at the end of a command
For example: **cp bigMovieFile.mp4 myfile.mp4 &**

nohup (nohangup): ignores the HUP signal so that the process will still run if the terminal is closed
For example: **nohup cp bigMovieFile.mp4 myfile.mp4**

jobs = show jobs running in the background

fg = put a background process to foreground (use %jobnum as argument if more than one process)
For example: **fg %2**

bg = put a foreground process into background

Stopping, suspending and restarting processes:

CTRL + C to stop current foreground process

CTRL + Z to suspend running process (use **fg** to resume it in foreground, **bg** to resume it in background)

kill = kill a process (You need the PID # of the process from **ps** command)

kill -9 <PID> = terminate process immediately

Shell (environment) variables

set = use without arguments to see what variables are set in current bash shell

It is customary for variable names to be written in upper case

To assign a variable: **VARIABLE = value**
For example: **SECRET_IDENTITY = Dracula**

Use colons to separate values in lists
For example: **PATH = /usr/local/bin:/usr/bin:/bin**

echo $VARIABLE = see value of a variable
For example: echo $SECRET_IDENTITY

export VARIABLE = make variable available for use in child processes
For example: export SECRET_IDENTITY

Export variables in user's **~/.bashrc** file to make available in all child processes:
For example:
export SECRET_IDENTITY=Dracula
export BACKUP_DIR=$HOME/backup

Use **alias** command to make short cuts for things you don't want to type over and over again (also useful in .bashrc file)
For example:
alias backup=/bin/zarble -v --nostir -R 20000 $HOME $BACKUP_DIR
Also handy for correcting common typos! For example:
alias mroe=more

Flow redirection

Redirect the standard output of commands:

'**>**' at the end of a command to redirect the result to a file (instead of screen)
For example: **ps -ejH > process.txt**

'**>>**' to append the output to the end of an existing file

Redirect the standard error:

'**2>**' at the end of the command to redirect error messages to a file
For example: **cut -d , -f 1 file.csv > file 2> errors.log**

'**2>&1**' to redirect error messages and standard output into the same file

Secure shell

For interacting securely with remote machines – always prompts for a password

ssh remoteUser@remoteHost = login to machine called remoteHost with username remoteUser
(To log out again type **exit**)

To copy between machines use **scp**. You must always provide the path to the file on the remote machine, not just the file name.

scp localFile remoteUser@remoteHost:~/remoteFile = secure copy file from local machine to user's home directory on remote machine

scp remoteUser@remoteHost:/home/user1/remoteFile localFile = secure copy file from remote machine to local machine

scp –r localDir remoteUser@remoteHost:/home/user1/remoteDir = recursively copy a directory and its contents from the local machine to the remote machine

Other methods for moving data between machines **(see man pages)**

- FTP
- SFTP
- rsync
- wget
- curl

Xargs **(feed output from one command to input of another which doesn't expect stdin)**

The "xargs"command runs the same command on all files specified in the input. Often used with "find" output, e.g.:
find . -name '*.nc' | xargschmodu=rwx (Changes permissions on all .ncfiles)

Shell scripting

Bash shell scripts must contain:
#!/bin/bash
As their first line and must have execute ('x') permission to run them.

Iteratate (loop) over multiple files, for example:
for file in *.pdb; do zip $file.zip $file; done

Other control structures are available

- if test –e myfile; then ...; fi
- case ...; esac
- while ...

(Use **man bash** to read about them).

For complicated control flows better to switch to a programming language, e.g. Python.

1. *Tux the penguin (Linux logo) as originally drawn as raster image by Larry Ewing in 1996.*