Learning Linux Notes

Contents

SHORTCUTS AND COMMANDS

1	Open Terminal	Ctrl + Alt + t
2	Close Terminal	Ctrl + d OR **C** exit
3	Clear	Ctrl + l
4	Cycle through commands	Up
5	See command history See command location	**C: history** **C: which**
6	Run a numbered command from history	!4 (for example)
7	Run most recent command	!!
8	Clear history	**C: history -c; history -w**
9	Search manual (command manual with 8 sections)	**C: man -k (write what to search)**
10	Open command manual for specific section	**C: e.g., man 6 which**
11	Open command manual for section 1	**C: e.g., man which**
12	Cancel cat command	Ctrl c
13	Read content as standard input and write to standard output	**C: cat**
14	Tells you where the terminal is located	**C: tty**
15	Shows the date and time	**C: date**
16	Extract a portion of a command or text which are separated by columns you decide for the	**C: cut** **C: cut -d -f**
17	Delete files (only accepts command line arguments); remove entire directories; remove directories safely; remove empty directories	**C: rm; rm -r, rm -ri; rmdir**
18	Find out the folder the shell is working in	**C: pwd**
19	List directories and files in a directory *(-l for long listing format; -lh long list human readable; -a for hidden files)*	**C: ls**
20	Move around the file system *(cd .. to go to parent folder)*	**C: cd**
21	Speed up navigating file system	tab
22	Info about a file	**C: file**
23	Match anything regardless of length (wildcard)	**W: ***
24	Match anything but for a single space (wildcard)	**W: ?**
25	Match just one space but allows you to specify options	**W: [ ]**
26	To save time for things with patterns	**..**
28	Create files	**C: touch**
29	Create directories; create entire dire directory paths	**C: mkdir; mkdir -p**

30	Copy files; Copy directories	**C: cp; cp -r**
31	Move and rename files	**C: mv**
32	Create and edit files	**C: nano**
33	Get administrative powers to run commands	**C: sudo**
34	Locate files that match a pattern in a database; ignore case sensitive in your set pattern; pull existing files; pull files with working shorcut	**C: locate; locate -i; locate -e; locate -L**
35	Find all files and directories from working directory; set a maxdepth; choose to show files or directories; specify a size of the files; execute a command on all that was found; do -exec option with safety	**C: find; -maxdepth 2; -type fORd; -name; -size +100kOR-100k; -exec C: \; ; -ok C: \;**
36	Count the number of lines in an output	**C: wc -l**
37	Sticks files together into the standard output	**C: cat**
38	Reverse files vertically	**C: tac**
39	Reverse files horizontally	**C: rev**
40	View and scroll the commands without putting them into the output	**C: | less**
41	Cut out the first 10 lines; cut out the specified no of lines (here 3)	**C: head; head -n 3**
42	Cut out the last 10 lines; cut out specified no of lines (here 4)	**C: tail; tail -n 4**
43	Sort data alphabetically; reverse the order; sort numerically; unique results; sort tabular data (here = sort column 3 numerically reversed); sort tabular human readable, sort by month	**C: sort; -r; -n; -u; sort -k 3nr; sort -k 4h, sort -k 2Mr**
44	Search data for certain text returning lines; gives line count; will search case insensitively; will invert search and what doesn't contain what you searched	**C: grep; -c ; -i ; -v**
45	Create an archive; check archive content; extracting archive content; Compressing with gzip gzip <name of tarball> Decompressing with gzip gunzip <name of tarball> ; Compressing with bzip2 bzip2 <name of tarball> Decompressing with bzip2 bunzip2 <name of tarball> ; Creating a tarball and compressing via gzip tar -cvzf <name of tarball> <file>... Decompressing a tarball and extracting via xzip tar -xvzf <name of tarball> ; Creating a tarball and compressing via bzip2 tar -cvjf <name of tarball> <file>... Decompressing a tarball and extracting via bzip2 tar -xvjf <name of tarball> ; Creating a tarball and compressing via xzip tar -cvJf <name of tarball> <file>... Decompressing a tarball and extracting via xzip tar -xvJf <name of tarball> ; Creating a .zip archive zip <name of zipfile> <file>... Extracting a .zip archive unzip <name of zipfile>	**C: tar** tar –cvf <name of tarball> <file>... ; tar –tf <name of tarball> ; tar –xvf <name of tarball> ;
46	Create a script in nano's first line; create a bash shell script in nano's first line	**C: #!PATH ; #/usr/bin/bash**
47	Execute a bash shell script ; e.g., executing a bash shell script in the desktop	**bash PATH; bash ~/Desktop/backup.sh**
48	Allow execution from command line to a bash script (can right click, properties, permission, execute permission)	**chmod +x PATH**
49	To let the ~/bin/ to become commands	PATH="$PATH:$HOME/bin"

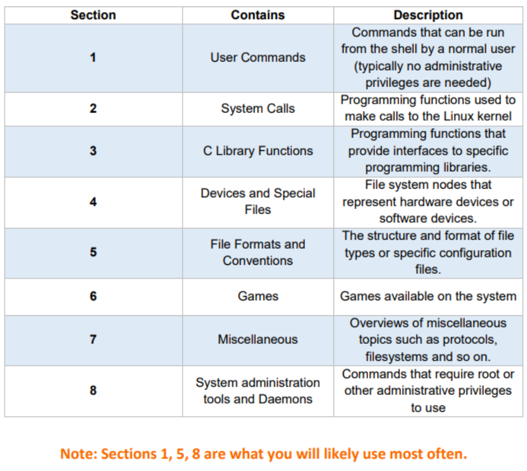
50	Automate and run commands	**C: crontab -e**
51	Run command every (you decide) e.g., every 15 mins in first crontab column	*/15
52	Do two commands after one is finished (without piping) ; example	&& ; **C: make && sudo make install**

TERMINAL

Terminal vs Command vs Shell

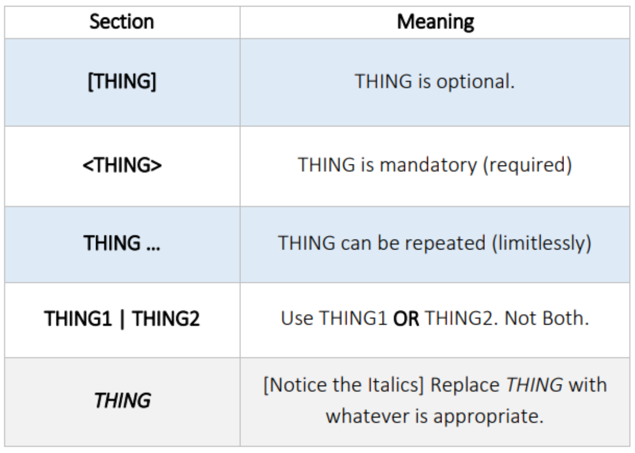
- Terminal is a window into the shell
- Commands are texts in the terminal
- Shells interprets the command (e.g., Bash)

Manual page structure and breakdown of reading it



Section	Contains	Description
1	User Commands	Commands that can be run from the shell by a normal user (typically no administrative privileges are needed)
2	System Calls	Programming functions used to make calls to the Linux kernel
3	C Library Functions	Programming functions that provide interfaces to specific programming libraries.
4	Devices and Special Files	File system nodes that represent hardware devices or software devices.
5	File Formats and Conventions	The structure and format of file types or specific configuration files.
6	Games	Games available on the system
7	Miscellaneous	Overviews of miscellaneous topics such as protocols, filesystems and so on.
8	System administration tools and Daemons	Commands that require root or other administrative privileges to use

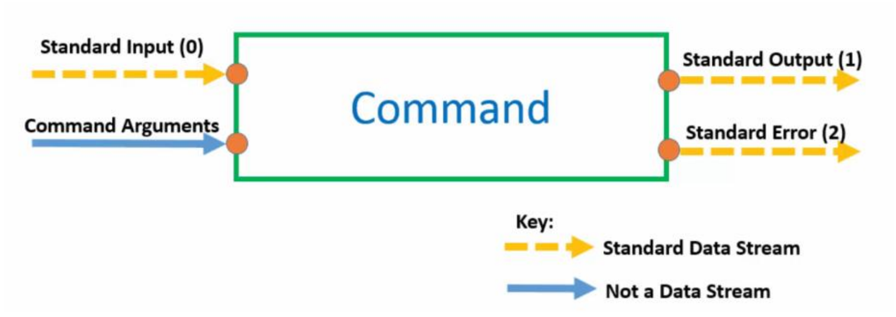
Note: Sections 1, 5, 8 are what you will likely use most often.



Section	Meaning
[THING]	THING is optional.
<THING>	THING is mandatory (required)
THING ...	THING can be repeated (limitlessly)
THING1 | THING2	Use THING1 **OR** THING2. Not Both.
THING	[Notice the Italics] Replace *THING* with whatever is appropriate.

- Synopsis: shows how to use the command, example: which [-a] filename …
  - which = command name
  - [ ] Square brackets = Anything in here is optional
  - -a = An option
  - filename = This is the commands input known as command line arguments
  - … = Means it takes more than one input which is a ‘filename’ for this command
- More rules
  - < > = anything in angle brackets is mandatory
  - E.g., [-a | -f] = This means you have to take one or the other, in this case if you want to use an option you can’t use both

Command input and output



- Standard output: When a command is run it is outputted through standard output
    - By default, standard output is connected to terminal
    - This output can be redirected into another commands input via piping
- Standard output: When a command fails it is outputted in standard error
    - By default, this output is also connected to terminal
- Standard input:
    - By default, connected to the keyboard
    - E.g., cat command reads of standard input
- Command line arguments: Example cal -A 1 -B 1 12 2017
    - Here there are arguments
        - 1 is the input command line argument for the -A option
        - 1 is the input command line argument for the -B option
        - 12 2017 are the input command line arguments for the overall cal command
    - All data streams can flow but not Command Arguments only associate with the option they are with
- Not all commands accept standard input but all accept command line arguments – can check via man pages

Redirection

- Will use cat command because reads from standard input and writes to standard output
    - cat

Redirection of standard input using 1> or > (or 1>> and >>)

- To redirect the standard output (1) of the cat command, use > like so:
    - cat 1> output.txt
      Linux is amazing
        - Here we have redirected to output.txt
        - 1> = To signify redirection to the 1 data stream; standard output (1)
        - Linux is amazing = Will now be outputted to Output.txt instead of terminal
    - cat > output.txt
      Linux is amazing
        - > = This achieves the same thing as >1 since Standard Output is the most often redirection
- To avoid truncation use 1>> when adding to a file to append to it

Redirection of standard error using 2> (or 2>>)

- To redirect the standard error (2) of the cat command, use 2> or 2>>
    - cat -k bla 2>> output.txt
        - The -k option is not in the cat command so this will result in an error usually in the terminal but now redirected to output.txt

<u>Redirection of standard input using 0< or <</u>

- To redirect the standard input (0) of the cat command, use 2> or 2>>
    - cat 0< input.txt 1>> output.txt >>error.txt
        - Given that input.txt has something, this will be read and then append to output.txt and any errors will append to into error.txt

Can redirect to another terminal by finding the location of the other terminal using the tty.

Can place redirection anywhere after the command, e.g., between options, before options or after options

Piping; using | , the tee command, and the xargs command

- Will use the date and cut command (15 and 16 in the list)

<u>Using the cut command to read out of date.txt which</u>

- Say there is a file called date.txt that has the output of the date command redirected into there with the text: Mon 4 Jul 16:11:08 BST 2022
- Command: cut < date.txt --delimiter " " --fields 1,5
  Result: Mon 16.11:08
    - --delimiter or -d = will let you decide what the columns which are denoted by field numbers are separated by, the " " shows that this is a space
    - --fields or -f = lets you decide the field you wish to cut and thus extract from the commands input, in this case fields 1 and 5

<u>Piping: Using | and the cut command to directly cut from the output of the date command</u>

- Command: date | cut -d " " -f 1,5
  Result: Mon 16.11:08
    - Now there is no need to output the date and then use that input in the cut, this is piping
- Command: date | cut -d " " -f 1,5 > today.txt
    - This outputs the overall command into the txt file
- Command: date >today.txt | cut -d " " -f 1,5

    - This won’t work as the standard output stream only goes to one place and redirection makes it go to the txt file before being piped into the cut command; solution = tee command

<u>Tee command: Snapshot of standard output of a command and still piping into input of another command</u>

- Command: date | tee -a fulldate.txt | cut -d “ “ -f 1,5
  Result: Mon 16.11:08 AND a fulldate.txt file with text = Mon 4 Jul 16:11:05 BST 2022
    - Here we got a snapshot of the date command into the fulldate.txt file and still used the cut the command
    - -a= To append to the fulldate.txt if there was already something there
    - Notice that > was not used since this will result in all the output being redirected before it can reach the next command, thus the tee command is not redirection but rather a snapshot

<u>Xargs command: Converting piped data into Command Line Arguments for commands that only accept Command Line Arguments</u>

- Command: echo “hello”
  Result: hello
    - The command line argument hello is echoed, not the input
- Command: date | echo
  Result:
    - Echo only reads its own arguments and not input; thus the date is not echoed even though it was inputted to the echo command
- Command: date | xargs echo
  Result: Mon 4 Jul 16:11:05 BST 2022
    - So if you pipe into the xargs command, then echo will receive the date as though it was its own command line argument and echo it back

Aliases

<u>Creating and Alias</u>

- Create a new text file in the home directory and save as: .bash_aliases
    - The . makes it and a hidden file
- In here now, you can save any long command and label it how you wish

- Command: date | tee /home/maaz/fulldate.txt | cut -d " " -f 1 | tee /home/maaz/shortdate.txt | xargs hello
- To give it a label and thus make it an alias:
  Alias get dates='above command'
  - "getdates" is now the alias here
  - Notice the need for ' ' for the command, can also use " " if you want

<u>Using an Alias as a Command in the Terminal</u>

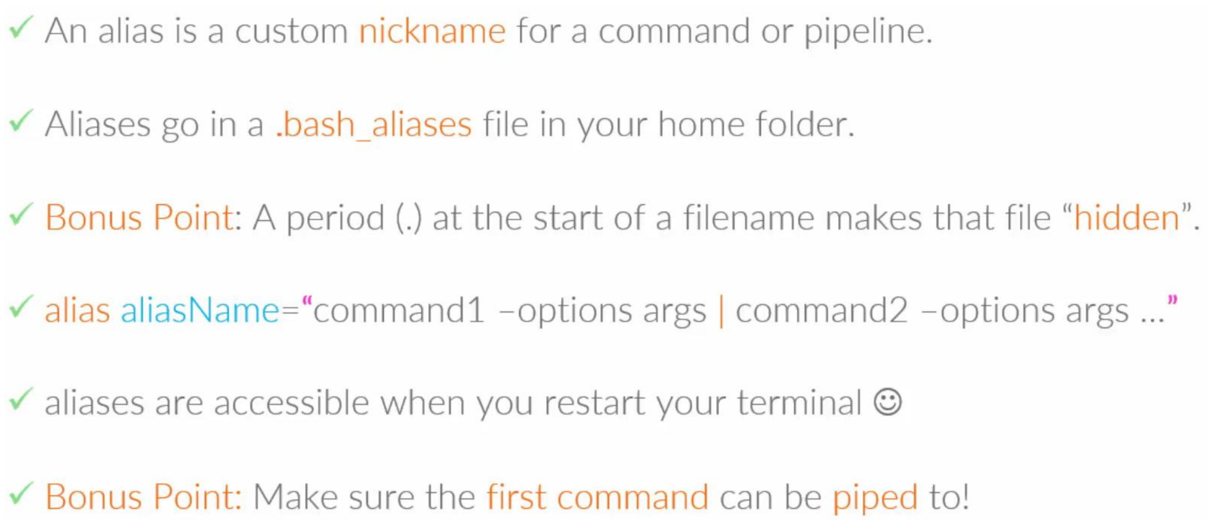
- Now you can use the alias as a command in the terminal
  - You must save the .bash_aliases file first and then reboot the terminal
- Command: getdates
  Result: hello Mon AND a fulldate.txt & shortdate.txt file created in hone directory

<u>Piping into and Alias</u>

- Alias: alias calharam="xargs cal -A 1 -B 1 | tee -a /home/maaz/thing.txt"
  - You must save the .bash_aliases file first and then reboot the terminal

  Command: echo "12 2017" | calharam

  Result: The calendar of one month before and after in the terminal and saved into the thing.txt file. This can be done again and will be appended due to the -a in the tee command within the alias.


✓ An alias is a custom nickname for a command or pipeline.

✓ Aliases go in a .bash_aliases file in your home folder.

✓ Bonus Point: A period (.) at the start of a filename makes that file "hidden".

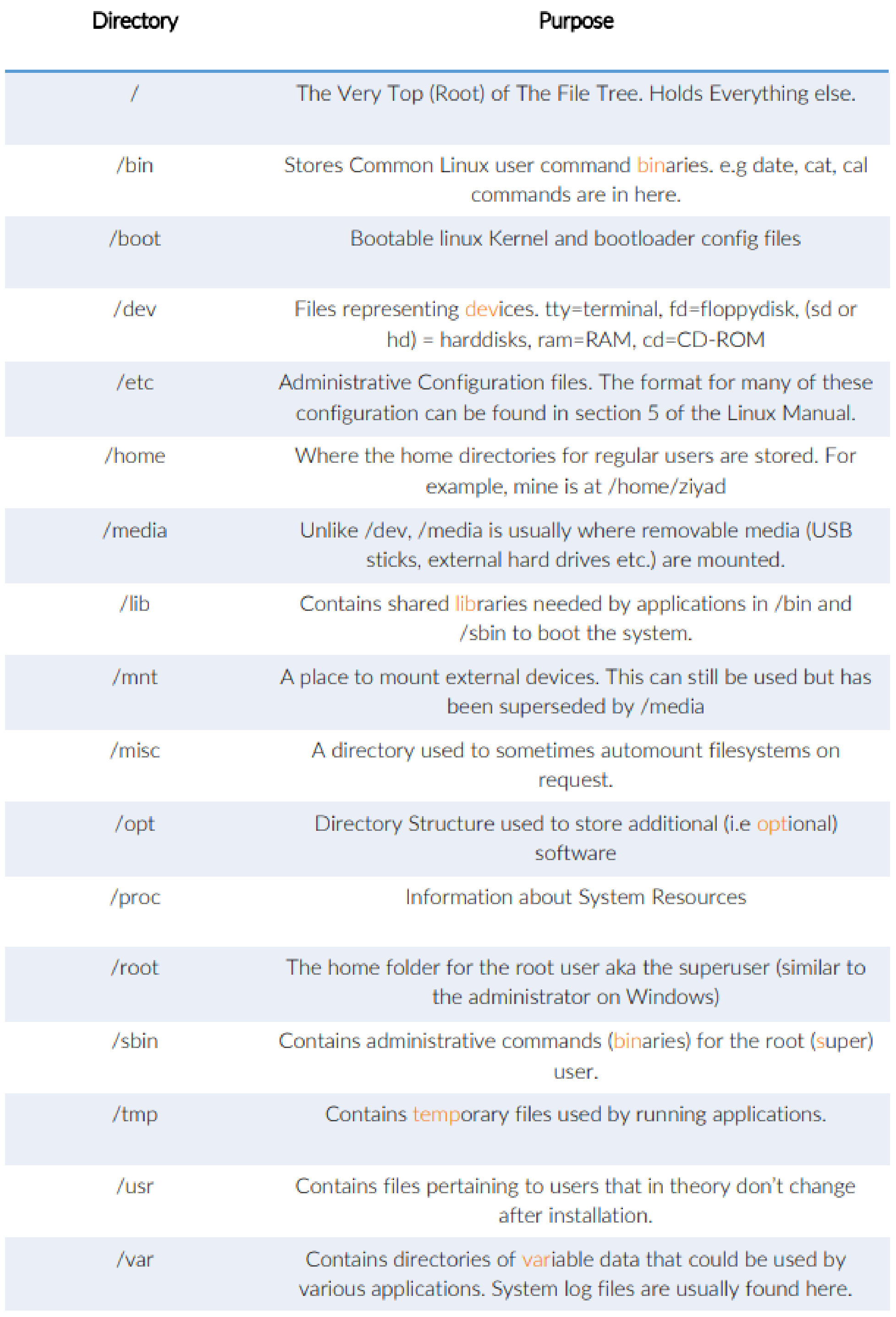
✓ alias aliasName="command1 –options args | command2 –options args ..."

✓ aliases are accessible when you restart your terminal ☺

✓ Bonus Point: Make sure the first command can be piped to!

FILE SYSTEM

Cheat sheet



Directory	Purpose
/	The Very Top (Root) of The File Tree. Holds Everything else.
/bin	Stores Common Linux user command binaries. e.g date, cat, cal commands are in here.
/boot	Bootable linux Kernel and bootloader config files
/dev	Files representing devices. tty=terminal, fd=floppydisk, (sd or hd) = harddisks, ram=RAM, cd=CD-ROM
/etc	Administrative Configuration files. The format for many of these configuration can be found in section 5 of the Linux Manual.
/home	Where the home directories for regular users are stored. For example, mine is at /home/ziyad
/media	Unlike /dev, /media is usually where removable media (USB sticks, external hard drives etc.) are mounted.
/lib	Contains shared libraries needed by applications in /bin and /sbin to boot the system.
/mnt	A place to mount external devices. This can still be used but has been superseded by /media
/misc	A directory used to sometimes automount filesystems on request.
/opt	Directory Structure used to store additional (i.e optional) software
/proc	Information about System Resources
/root	The home folder for the root user aka the superuser (similar to the administrator on Windows)
/sbin	Contains administrative commands (binaries) for the root (super) user.
/tmp	Contains temporary files used by running applications.
/usr	Contains files pertaining to users that in theory don't change after installation.
/var	Contains directories of variable data that could be used by various applications. System log files are usually found here.

Navigating the File System (pwd, ls, cd)

Using pwd to show which directory the terminal is currently working in

- pwd = Prints the path to the current working directory; Tells you the folder the shell is currently working in
    - Stands for print working directory
- ~ = The tilda in the bash shell (shell we are using) is the current users home directory

Using ls to show the files in in a directory

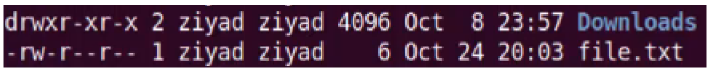
- ls = list the files in the current directory (only takes command line arguments
  - Command:

    ls /home/maaz

    ls ~

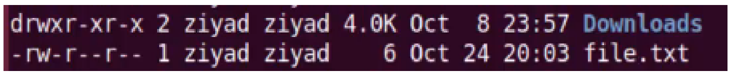
    ls
    - All result in the home directory's files shown. ~ means /home/maaz and ls naturally lists the current directory
  - Command: ls -F

    Result: Slash behind files that are folders and without slash is a file
  - Command: ls -l *(long form format option)*

    Result: 

```
drwxr-xr-x 2 ziyad ziyad 4096 Oct  8 23:57 Downloads
-rw-r--r-- 1 ziyad ziyad    6 Oct 24 20:03 file.txt
```

    - d = directory
    - Some file permissions on the left:
      - rwx = read write execute (this is for the user indicated by the first ziyad)
      - r-x = read and execute only (this is for the group user indicated for by the second ziyad)
      - r-x = read and execute only (this is for everyone else)
    - 4096 = how large the file is
    - Oct 8 23:57 = when the file was created
    - Downloads = File name
    - - = File, not a directory
  - Command: ls -lh

    Result: 

```
drwxr-xr-x 2 ziyad ziyad 4.0K Oct  8 23:57 Downloads
-rw-r--r-- 1 ziyad ziyad    6 Oct 24 20:03 file.txt
```

    - makes it human readable
  - Command: ls -a
    - Shows the hidden files in a folder
  - Command: ls Documents/ Downloads/ Pictures/
    - This allows you to see the contents of all the directories

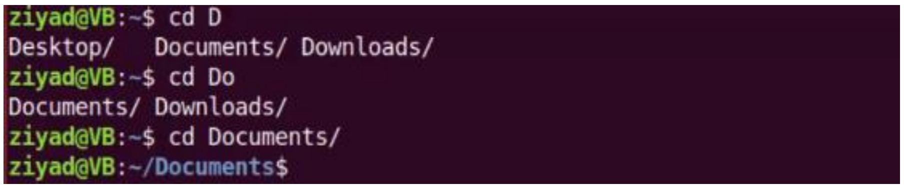
Using cd command to move around file system

- Move around the file system using the cd command
- Command: To get to the downloads folder *(the first two are absolute paths, the last is a relative path)*

        - cd /home/maaz/Downloads
        - cd ~/Downloads
        - cd Downloads *(relative path as currently in the terminal we are in the home folder)*
- Command: To get back to the home directory
    - cd /home/maaz
    - cd ~
    - cd *(if there are no command line arguments, then automatically will return to home folder)*
- Using ls - a command, you find a . and .. file hidden in every folder
    - . = refers to current folder
    - .. = refers to parent folder
- Command: To get to the parent folder
    - cd ..

Using tab to speed up navigation

- If you do cd /h then press tab, the shell will complete it if there is only one option for documents that start with that letter
- If there are more than one option, then press the starting letter and then double tab to show all the files that start with the letter, eg cd /home/maaz/d tab tab, then add more letters:



```
ziyad@VB:~$ cd D
Desktop/   Documents/ Downloads/
ziyad@VB:~$ cd Do
Documents/ Downloads/
ziyad@VB:~$ cd Documents/
ziyad@VB:~/Documents$
```

- You can right click from graphical UI as well to open it in the terminal

File extensions

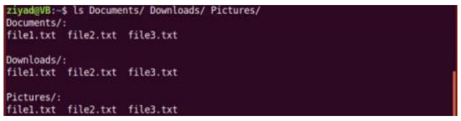
- Command: file tux.txt
  Result: *Info on the file type and about the file*
- Linux does not determine the file type using the file extension, rather it uses a *header*
    - Uses a code over the file to determine it and is like a label
    - Changing the extension may change the icon but using the file command shows us that the file type itself did not change
        - Trying to open the file, Linux will try to use a program that can open whatever the extension is but fail since the wrong extension is used, the operating system itself does not care that its changed but to open it you need the correct extension

- If the software cant recognise the file extension, then it will use the operating system to determine the filetype and suitably open it

Wildcards (, [ ], { })*

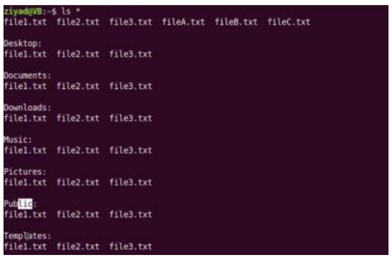
- Building blocks to creating regular expressions

<u>*Using the * wildcard*</u>

- Wildcard: *
    - Matches everything regardless of length
    - E.g., the ls command can list all the files within multiple directories or all if you add them in manually like; Command: ls Documents/ Downloads/ Pictures/ *(this isn't all but you can do all)* 

```
ziyad@VB:~$ ls Documents/ Downloads/ Pictures/
Documents/:
file1.txt  file2.txt  file3.txt

Downloads/:
file1.txt  file2.txt  file3.txt

Pictures/:
file1.txt  file2.txt  file3.txt
```

    - However if you done; Command: ls * - *this will list all of the contents of all the directories in the folder like so:*

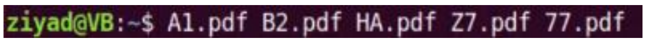


```
ziyad@VB:~$ ls *
file1.txt  file2.txt  file3.txt  fileA.txt  fileB.txt  fileC.txt

Desktop:
file1.txt  file2.txt  file3.txt

Documents:
file1.txt  file2.txt  file3.txt

Downloads:
file1.txt  file2.txt  file3.txt

Music:
file1.txt  file2.txt  file3.txt

Pictures:
file1.txt  file2.txt  file3.txt

Public:
file1.txt  file2.txt  file3.txt

Templates:
file1.txt  file2.txt  file3.txt
```

    - Command: ls D* - *this will now list out anything that starts with a letter D*
    - Command: ls *.txt - *this will now match anything that ends with .txt*

<u>*Using the ? wildcard*</u>

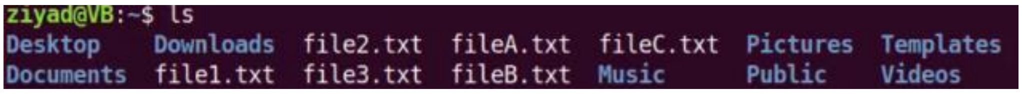
- Wildcard: ? or ?? or ??? etc
    - This will match with things that have the same number of question marks
    - Command: ls ??.pdf



```
ziyad@VB:~$ A1.pdf B2.pdf HA.pdf Z7.pdf 77.pdf
```

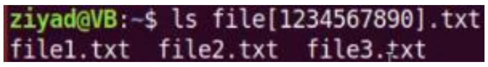
<u>*Using the [ ] wildcard*</u>

- Wildcard: [ ] or [ ] [ ]
    - Allows you to be more specific - like a single questions mark but you decide what that question mark matches
    - Say a directory has:

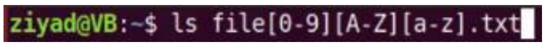


```
ziyad@VB:~$ ls
Desktop     Downloads  file2.txt  fileA.txt  fileC.txt  Pictures  Templates
Documents   file1.txt  file3.txt  fileB.txt  Music      Public    Videos
```

- Command: ls file[0-9].txt
  - this specifically limits the single space after the word file to require a number between 0 and 9, not the letters A to Z

  Result: 

```
ziyad@VB:~$ ls file[1234567890].txt
file1.txt  file2.txt  file3.txt
```

- Command: 

```
ziyad@VB:~$ ls file[0-9][A-Z][a-z].txt
```

Creating files and folders (touch, mkdir)

Using the touch command to make files and mkdir command to make directories

- Touch: Create new empty files
  - Command: touch file1

    Result: file1 is created in the directory you are in
  - Command: touch ~/Documents/distantfile1

    Result: distantfile1 will be made in the Documents directory even though this may have not been the working directory
- Can also use echo to create new files:
  - Command: echo "hello" > file.txt
    - Shell will create this file in the working directory
- mkdir: Make directories
  - Command: mkdir folder

    Result: directory called "folder" created in the working directory
  - Command: mkdir ~Documents/folder

    Result: directory created called folder in the Documents directory
  - Command : mkdir ~bla/thing/shablam

    Result: Fail as bla doesn't exist in the home folder

    Command: mkdir -p ~bla/thing/shablam

    Result: The whole path is created in the home folder
  - Command: mkdir eggs are tasty

    Result: Three directories created in the working folder called eggs, are, and tasty

    Command: mkdir "eggs are tasty"

    Result: One directory created in the working folder called eggs are tasty

    Better to just use underscores (_) instead of spaces

Using brace expansion: make large number of directories and files

- Command: mkdir {jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec}_{2018,2019,2020,2021,2022} *(could have done 2018..2022 since the numbers are in order)*
  Result: They expand like double brackets creating folders of jan to dec in the 2018 folder, 2019 folder and so forth
- Command: touch {jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec}_{2018..2022}/file{1..100)
  Result: This lets you make 100 files in all of those folders created above
- Command: touch file{A..F}.txt
  Result: fileA.txt until fileF.txt is created in the working directory

Deleting files and directories (rm, rmdir)

<u>Using rm command to delete files</u>

- Say there is file1.txt in the home directory, file2.txt in the documents folder, file3.txt in the downloads folder and you are in the home directory
  Command: rm file1.txt Documents/file2.txt Downloads.txt
  Result: All of them are deleted, note there is no need for ~/file1.txt since file1 is already in the home directory
- Useful to use wildcards here as much as possible

<u>Using rm -r command to delete directories</u>

- Create the following path: mkdir -p delfolder/delme{1,2,3}
  Command: rm -r delfolder
  Result: Entire path is deleted
  - Risky since the whole path may be deleted, also notice the need of using -r (recursive) option

<u>Using rm -ri command to delete directories salfely</u>

- Command: rm -ri delfolder
  Result: This also essentially allows you to delete the entire path but in a more careful manner, you need to check off if you want to delete or not

<u>Using rmdir command to delete empty directories</u>

- Create path: mkdir -p delfolder/folder{1..3}
  Create files: touch delfolder/folder{1,2}/file{1..10}.txt
  Command: rmdir delfolder/*
  Result: Only folder 3 is deleted since that’s the only empty directory

Copying files and folders (cp)

Using the cp command to copy files

- Create: echo "hello there you eggs" > Desktop/file1.txt
  Command: cp ~/Desktop/file1.txt file2.txt
  Result: This copies file1.txt into the working directory
- Move file2.txt into desktop as well and create command: mkdir ~/Desktop/destination/
  Command: cp ~/Desktop/file1.txt file2.txt destination/
  Result: file1.txt and file2.txt is copied from Desktop to the destination directory
- Delete file1.txt and file2.txt from the Desktop directory. Whilst in the desktop directory do the following
  Command: cp ~/Desktop/destination/* .
  Result: All the files are copied from the destination directory into the working directory. This is done by the wildcard * and the . which represents the working directory which is the desktop in this case

Using the cp -r command to copy directories (folders)

- Create: touch ~/Desktop/copy_me/file(A..C}
  Command: cp -r ~/Desktop/copy_me/ destination/
  Result: The copy_me folder is copied into the destination folder

Moving and renaming files and directories (mv)

Using mv to rename files and folders

- Create: mkdir ~/Desktop/oldfolder AND touch ~/Desktop/oldname.tx AND go into desktop directory cd Desktop/
  Command: mv oldname.txt newname.txt
  Result: The file name changes from oldname.txt to newname.txt
- Create: touch ~/Desktop/oldfolder/file{1..10}
  Command:mv oldfolder/ newfolder/
  Result: Old folder is renamed as new folder
  - Note that to execute command you must be in the directory, otherwise write the full path of the folders name that you are changing

Using mv command to move files and directories

- Working of the above scenario, adding: touch ~/Desktop/newfolder/file{1..10}
  Command: mv ~/Desktop/newfolder/* .
  Result: Moves everything in the newfolder into the working directory (due to the .)

- Move files back to newfolder: mv file* newfolder/
  Command: mv /~/Desktop/newfolder/ ~/Documents
  Result: The folder is moved to the Documents folder

Using mv command to move files and directories while renaming

- Command: mv ~/Documents/newfolder ~/Desktop/eggman
  Result: newfolder is now moved into the desktop but is now called eggman

Editing files (nano)

- ^ = Ctrl
  M- = Alt
- Directory: cd ~/Desktop/
  Command: nano diary.txt
  Result: This creates a file and it also opens it in the terminal for editing
  - The arrow ^ at the bottom represents ctrl, this is how you can access what is there
  - You can now write into the file and then save it using the ctrl+O

  Write into nano: I am learning Linux intensely NOW: Ctrl O to save and press enter
  Result: diary.txt is created with what's written above
- Create: echo "eggs are tasty" ~/Desktop/file.txt
  Command: nano ~/Desktop/diary.txt
  Use ctrl+r to insert the file.txt file from the working directory
- Use ctrl+w to search withing the diary.txt file then use the *Alt key* to use the m- further options. Eg., after ctrl+w, alt+c resulting in case sensitive search
- Ctrl+k is for cutting and pasting multiple times
- Crl+j justifies - takes the terminal form factor on
- F12 can be used for spell checking (found in the help section and different from udemy course); Wont work unless you set it up for the nano command
  - Command: nano /etc/nanorc
    - This is unwritable so you need to use:
  - Command: sudo nano /etc/nanorc
    - Enter password and get into it
    - Search for aspell using the crl+w shortcut and get rid of the # and space at the start of that sentence
  - Go back to the: nano diary.txt and click F12 and the spell checker will now be working
- Ctrl+c = tells you the position you are at
- Ctrl+shift+_ = enter the line you want to go to and then the column number on that line

- Alt+u = undo
- Alt+6 = copy

The Locate Command

Using the locate command

- Will search a database to locate the pattern that you have set and locate all the files that match
- Command: locate *.conf

  Result: All the files that end in .conf will be opened
    - Command: locate *.CONF
        - This gives nothing since its case sensitive
    - Command: locate -i *.CONF
        - This will give the same as the above ignoring case sensitivity
    - Command: locate -i --limit 3 *.conf
        - This will limit the results to 3 lines
- Command: locate -S (THIS DOESN'T WORK)
    - The point is that locate is a database and this gives the location
    - The data base is only updated once daily so this can cause issues
- Command: locate -e *.conf
    - This ensures that only things that actually exists and haven't been moved or deleted in that day is pulled from the data base
- Command: locate -L (or --follow) *.conf
    - This ensures only the locate pulls things with shortcuts and links working

Updating the database using updated command and the sudo command

- Create: touch ~/Desktop/findme.txt

  Command: locate findme.txt
    - Noting is found since this was made after the database was last updated

  Command: updatedb
    - Wont run as it requires administrative privileges as the man pages tells us its in section 8 of the manual
- Command: sudo updatedb
    - This updates the database as an administrative user and you put your password into here

  Command locate findme.txt

  Result: The locate command finds it

The Find Command

Using the Find Command

- Command: find
  - Lists out all the files AND directories starting in the current working directory and all the subsequent directories beneath this to an infinite depth
  - Doesn't need a database to work but thus a little slower
- Command: find ~/Desktop
  - will list out all files and directories withing the Desktop directory to an infinite depth

Using the -maxdepth option

- Command: find ~/ -maxdepth 1
  - This results to a max depth of 1 from the working directory, you can use this option for whenever you're using the find command
  - Notice the option comes after you write the location you are finding at
  - This is a good way of listing whats in the the current folder

Using the -type option

- Command: find . -type f
  - This finds only the files
- Command: find . -maxdepth 2 -type d
  - This finds only directories
  - Notice the specified maxdepth here as well (must be in this order)

Using the -name option

- Create: touch ~/Desktop/nol.txt
  Command: find ~/ -name "nol.txt"
  Result: This gives the path to the nol.txt starting from the working directory (~/)
  - Using -iname as the option makes it case insensitive
- Command: find ~/ -maxdepth 1 -iname "nol.txt"
  Result: Wont find it since the depth was restricted

Using the -size option and the wc command

- Command: find / -type f -size +100k
  Result: Only gives files and the files that are given are all greater than 100k (can do less than 100k using -), you also get lots of permissions denied since to enter some folders you need administrative powers, so just start with sudo then the command
- Command: sudo find / -type f -size +100k | wc -l

Result: The wordcount command with the -l option counts the number of lines so we pipe into it, so we get that info as well

We get = 9459

- Command sudo find / -type f -size +100k -size -2M | wc -l

  Result: We get = 8108 files, which is smaller due to restricting the max size of the files
- If you want to say less than something OR greater than something, put the -o option in the middle of both requirements
- Command: sudo find / -type f -size -100k -o -size +2M | wc -l

<u>Using -exec C: \; or -ok C: \; option to carry out commands on found files and directories</u>

- Create: mkdr ~/Desktop/copy_here

  Command: sudo find / -type f -size +100k -size -5M -exec cp {} ~/Desktop/copy_here \;
  - -exec = This allows you to carry out a command on whatever you have just found
  - cp is the copy command and after the command you specify what you want to copy and then where you want to copy it:
    - {} = this is what you want to copy; this means all that was found
    - ~/Desktop/copy_here = this is where you want to copy {}
  - \; = This shows you have stopped the -exec option

  Result: All the files from the root directory to an infinite depth above 100k and less than 5M is copied over to the copy_here file found on the desktop
- Can do the same thing but instead of -exec which can be dangerous, you can use the -ok option
- Create: mkdir ~/Desktop/haystack AND mkdir ~/Desktop/haystack/folder{1..500} AND touch ~/Desktop/haystack/folder{1..500}/file{1..100} AND touch haystack/folder$(shuf -I 1-500 -n 1)/needle.txt (this basically places a file called needle.txt in one of the 500 folders)

  Command: find ~/Desktop/haystack/needle/ - type f -name "needle.txt"

  Result: /home/maaz/Desktop/haystack/folder285/needle.txt
- Command: find ~/Desktop/haystack/ -type f -name "needle.txt" -exec mv {} Desktop/ \;
  - Recall {} = this means the result of what was found i.e. in this case the location of the needle.txt file

  Result: the needle.txt file is moved into the desktop

Viewing files

<u>Using the cat command to stick things together or read content</u>

- Create: touch ~/Desktop/file{1..5}.txt AND cd ~/Desktop AND echo “hello” > file1.txt AND echo “eggs” >file2.txt and so forth until hello eggs are beautiful yk

  Command: cat file1.txt

  Result: hello

  Command: cat file1.txt file2.txt file3.txt file4.txt file.txt

  Result:

  hello

  eggs

  are

  beautiful

  yk
- Command: file[1-5].txt

  Result:

  hello

  eggs

  are

  beautiful

  yk

Using tac to reverse what it receives as an input vertically

- In desktop directory; Create: echo “abc” > alpha.txt AND echo “def” >> alpha.txt
  - Note the >> to ensure you append

  Command: cat alpha.txt

  Result:

  abc

  def

  Command: tac alpha.txt

  Result:

  def

  abc
  - Essentially it reversed this file vertically
- Command: cat file[1-5].txt | tac

Using rev to reverse what it receives as an input horizontally

- Command: cat file[1-5].txt | rev

  Result:

  Olleh

Sgge

Era

Lufituaeb

Ky

  - Essentially it reversed the concatenated files horizontally

Using less instead of cat for larger files

- Create: Find a large file; sudo find / -maxdepth 4 - size +20k -name “*.conf”

  One large file found was /etc/cups/cups-browsed.conf
- Command: cat /etc/cups/cups-browsed.conf

  Result: Large mess in the terminal

  Command: less /etc/cups/cups-browsed.con

  Result: You can scroll through this with the arrow key and it isn’t spat into the terminal, quit with
- This can also be piped into, keeping things neat
- Press q to exit

Using head and tail command to see snippets

- cat ~/Desktop/file[1-5].txt | head

  Result: The normal output since by default it shows the first 10 lines

  hello

  eggs

  are

  beautiful

  yk
- Command: cat ~/Desktop/file[1-5].txt | head -n 2

  Result: Now you specified to show the first 2 rows instead

  hello

  eggs
- cat ~/Desktop/file[1-5].txt | tail -n 2

  Result: specified the last two lines instead of standard last 10 lines

  beautiful

  yk

Sorting Data

Using sort command

- Command: sort file.txt

Result: puts it into alphabetical order

- Command: sort -r file.txt OR sort file.txt | tac

  Result: puts it into reverse alphabetical order

- Command: sort -n file.txt

  Result: puts into numerical order

  Command: sort -nr file.txt

  Result: puts into reversed numerical order

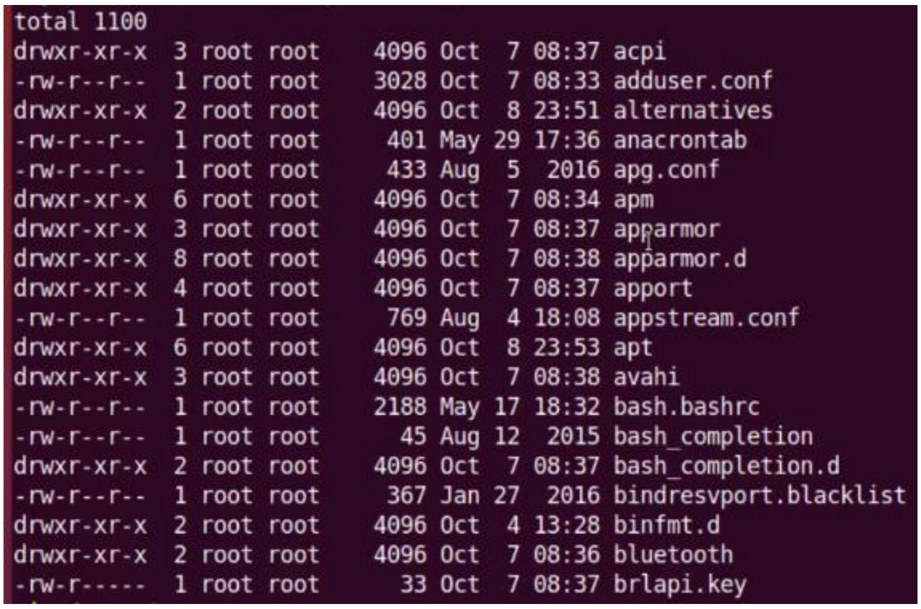
- Command: sort -u file.txt

  Result: sorts repeated data and shows each result only once

Sorting tables

- Create: ls -l /etc/ | head -n 20

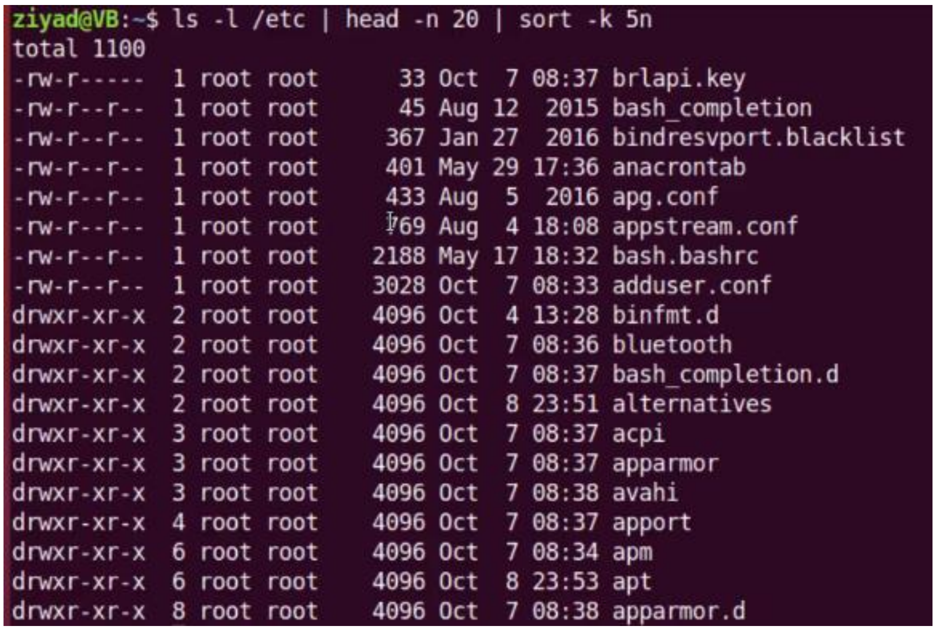
  This lists the first 20 lines of the /etc folder:



```
total 1100
drwxr-xr-x  3 root root    4096 Oct  7 08:37 acpi
-rw-r--r--  1 root root    3028 Oct  7 08:33 adduser.conf
drwxr-xr-x  2 root root    4096 Oct  8 23:51 alternatives
-rw-r--r--  1 root root     401 May 29 17:36 anacrontab
-rw-r--r--  1 root root     433 Aug  5  2016 apg.conf
drwxr-xr-x  6 root root    4096 Oct  7 08:34 apm
drwxr-xr-x  3 root root    4096 Oct  7 08:37 apparmor
drwxr-xr-x  8 root root    4096 Oct  7 08:38 apparmor.d
drwxr-xr-x  4 root root    4096 Oct  7 08:37 apport
-rw-r--r--  1 root root     769 Aug  4 18:08 appstream.conf
drwxr-xr-x  6 root root    4096 Oct  8 23:53 apt
drwxr-xr-x  3 root root    4096 Oct  7 08:38 avahi
-rw-r--r--  1 root root    2188 May 17 18:32 bash.bashrc
-rw-r--r--  1 root root      45 Aug 12  2015 bash_completion
drwxr-xr-x  2 root root    4096 Oct  7 08:37 bash_completion.d
-rw-r--r--  1 root root     367 Jan 27  2016 bindresvport.blacklist
drwxr-xr-x  2 root root    4096 Oct  4 13:28 binfmt.d
drwxr-xr-x  2 root root    4096 Oct  7 08:36 bluetooth
-rw-r-----  1 root root      33 Oct  7 08:37 brlapi.key
```

  Command: ls -l /etc | head -n 20 | sort -k 5n

  Result: sorts it now by the 5th column:



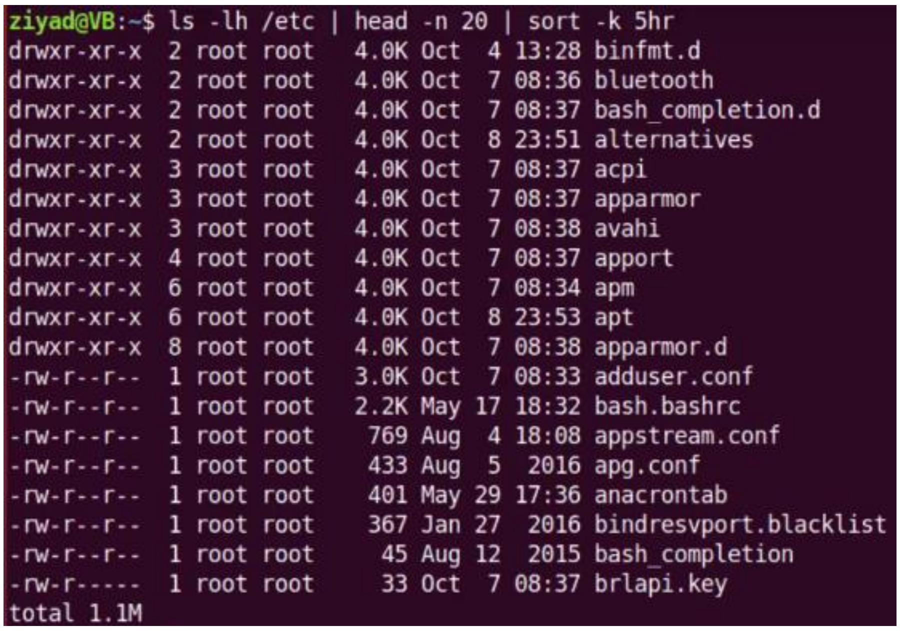
```
ziyad@VB:~$ ls -l /etc | head -n 20 | sort -k 5n
total 1100
-rw-r-----  1 root root    33 Oct  7 08:37 brlapi.key
-rw-r--r--  1 root root    45 Aug 12  2015 bash_completion
-rw-r--r--  1 root root   367 Jan 27  2016 bindresvport.blacklist
-rw-r--r--  1 root root   401 May 29 17:36 anacrontab
-rw-r--r--  1 root root   433 Aug  5  2016 apg.conf
-rw-r--r--  1 root root   769 Aug  4 18:08 appstream.conf
-rw-r--r--  1 root root  2188 May 17 18:32 bash.bashrc
-rw-r--r--  1 root root  3028 Oct  7 08:33 adduser.conf
drwxr-xr-x  2 root root  4096 Oct  4 13:28 binfmt.d
drwxr-xr-x  2 root root  4096 Oct  7 08:36 bluetooth
drwxr-xr-x  2 root root  4096 Oct  7 08:37 bash_completion.d
drwxr-xr-x  2 root root  4096 Oct  8 23:51 alternatives
drwxr-xr-x  3 root root  4096 Oct  7 08:37 acpi
drwxr-xr-x  3 root root  4096 Oct  7 08:37 apparmor
drwxr-xr-x  3 root root  4096 Oct  7 08:38 avahi
drwxr-xr-x  4 root root  4096 Oct  7 08:37 apport
drwxr-xr-x  6 root root  4096 Oct  7 08:34 apm
drwxr-xr-x  6 root root  4096 Oct  8 23:53 apt
drwxr-xr-x  8 root root  4096 Oct  7 08:38 apparmor.d
```

Command: ls -l /etc | head -n 20 | sort -k 5nr

Result: Sorts it now by the 5th column numerically in the reverse order

Command: ls -lh /etc| head -n 20 | sort -k 5hr

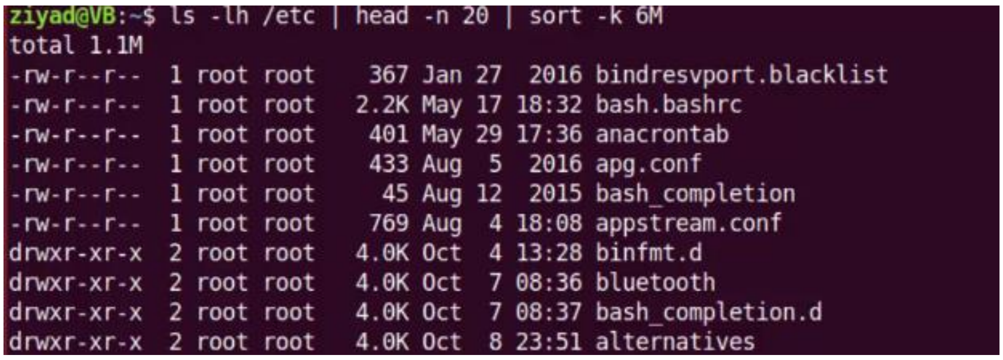
Result: sorts in reverse numerical order using for human readable column



```
ziyad@VB:~$ ls -lh /etc | head -n 20 | sort -k 5hr
drwxr-xr-x  2 root root  4.0K Oct  4 13:28 binfmt.d
drwxr-xr-x  2 root root  4.0K Oct  7 08:36 bluetooth
drwxr-xr-x  2 root root  4.0K Oct  7 08:37 bash_completion.d
drwxr-xr-x  2 root root  4.0K Oct  8 23:51 alternatives
drwxr-xr-x  3 root root  4.0K Oct  7 08:37 acpi
drwxr-xr-x  3 root root  4.0K Oct  7 08:37 apparmor
drwxr-xr-x  3 root root  4.0K Oct  7 08:38 avahi
drwxr-xr-x  4 root root  4.0K Oct  7 08:37 apport
drwxr-xr-x  6 root root  4.0K Oct  7 08:34 apm
drwxr-xr-x  6 root root  4.0K Oct  8 23:53 apt
drwxr-xr-x  8 root root  4.0K Oct  7 08:38 apparmor.d
-rw-r--r--  1 root root  3.0K Oct  7 08:33 adduser.conf
-rw-r--r--  1 root root  2.2K May 17 18:32 bash.bashrc
-rw-r--r--  1 root root   769 Aug  4 18:08 appstream.conf
-rw-r--r--  1 root root   433 Aug  5  2016 apg.conf
-rw-r--r--  1 root root   401 May 29 17:36 anacrontab
-rw-r--r--  1 root root   367 Jan 27  2016 bindresvport.blacklist
-rw-r--r--  1 root root    45 Aug 12  2015 bash_completion
-rw-r-----  1 root root    33 Oct  7 08:37 brlapi.key
total 1.1M
```

Command: ls -lh /etc| head -n 10 D| sort -k 6M

Result: Now sorts by month



```
ziyad@VB:~$ ls -lh /etc | head -n 20 | sort -k 6M
total 1.1M
-rw-r--r--  1 root root    367 Jan 27  2016 bindresvport.blacklist
-rw-r--r--  1 root root   2.2K May 17 18:32 bash.bashrc
-rw-r--r--  1 root root    401 May 29 17:36 anacrontab
-rw-r--r--  1 root root    433 Aug  5  2016 apg.conf
-rw-r--r--  1 root root     45 Aug 12  2015 bash_completion
-rw-r--r--  1 root root    769 Aug  4 18:08 appstream.conf
drwxr-xr-x  2 root root   4.0K Oct  4 13:28 binfmt.d
drwxr-xr-x  2 root root   4.0K Oct  7 08:36 bluetooth
drwxr-xr-x  2 root root   4.0K Oct  7 08:37 bash_completion.d
drwxr-xr-x  2 root root   4.0K Oct  8 23:51 alternatives
```

Searching file content: grep command

Will search any for lines of a particular text.

- Create: echo “hello” > ~/Desktop/hello.txt; echo “eggs” >>~/Desktop/hello.txt AND so forth until lines are;
  hello
  eggs
  are
  beautiful
  yk
  Command: grep e hello.txt
  Result: All the lines that have an e are outputted to the terminal
  hello
  eggs
  are
  beautiful
  - Use -i to make it case insensitive
  - Use -c option to see how many lines of occurrences
  - Use -v to show the lines that don’t match and combine with -c to count how many of these lines there are; -vc or -cv
    - E.g., grep -vc “our boys” gadsby_manuscript.txt
- Command: grep -i “e” gadsby/gadsby_manuscript.txt hello.txt
  - This is how you can search for something in more than one file at once
  - Can conbine with -c option to see how many lines from both like -ic
- Create: mkdir ~/Desktop/hello AND mv ~/Desktop/hello.txt ~/Desktop/hello
  Command: ls ~/Desktop/hello | grep hello.txt

Result: It highlights hello.txt it to show you that it is present and filters out what you are looking for

- Create: ls -lF /
    - This gives me all the permissions of the base directory
    - Note the F option classifies whether what is found is a file or directory with a slash

  Command: ls -lF / | grep root
    - Now it filters out only lines that contain root in it
- Create: ls -F /etc/

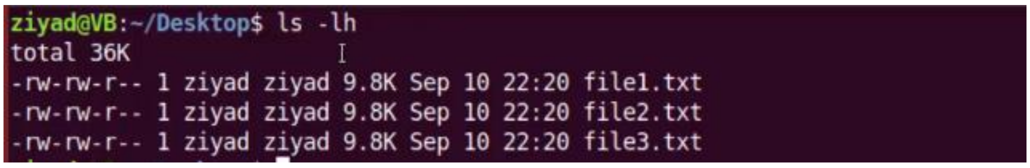
  Command: ls -F /etc/ | grep -v /
    - This will now show only the files and not the folders
- Command: man -k “print” | grep files
    - Now you’re filtering the man pages as well for files

File Archiving and Compression

1) Make a tarball (storing the files in one place) 2) Compress the tarball with a compression algorithm

Using tar command to make tar balls

- Create: find /etc/ > ~/Desktop/file1.txt (repeat for file2.txt and file3.txt:



```
ziyad@VB:~/Desktop$ ls -lh
total 36K
-rw-rw-r-- 1 ziyad ziyad 9.8K Sep 10 22:20 file1.txt
-rw-rw-r-- 1 ziyad ziyad 9.8K Sep 10 22:20 file2.txt
-rw-rw-r-- 1 ziyad ziyad 9.8K Sep 10 22:20 file3.txt
```

  Command: tar -cvf ourarchive.tar file[1-3]
    - -c= create a new archive
    - -v= verbose which basically makes the tar command tell you whats happening
    - -f= makes the tar command accept files
    - ourarchive.tar = what to call the tar ball
    - file[1-3].txt = what to put into the tar ball

  Result: Archive created in the desktop but the size is a little larger since the archive itself has some size
- Command: tar -tf ~/Desktop/ourarchive.tar

  Result: Shows whats in the tarball
- Command: tar -xvf ourarchive.tar

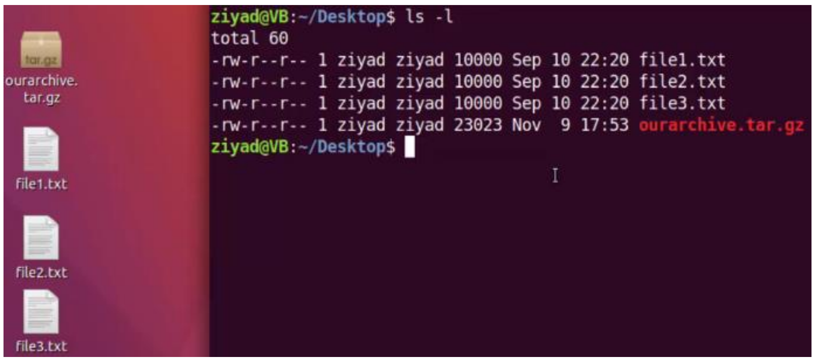
  Result: The files are extracted from the tar ball (although the files are still in there)

Using, gzip & gunzip; bzip2 & bunzip2; zip &unzip, commands to compress files

Gzip is usually faster but bzip2 can usually give a more compressed file

- Command: gzip ourarchive.tar

  Result: The file is compressed in place, can check this with the file command, in this case 7k smaller, compression of 23% from the

```
ziyad@VB:~/Desktop$ ls -l
total 60
-rw-r--r-- 1 ziyad ziyad 10000 Sep 10 22:20 file1.txt
-rw-r--r-- 1 ziyad ziyad 10000 Sep 10 22:20 file2.txt
-rw-r--r-- 1 ziyad ziyad 10000 Sep 10 22:20 file3.txt
-rw-r--r-- 1 ziyad ziyad 23023 Nov  9 17:53 ourarchive.tar.gz
ziyad@VB:~/Desktop$
```

  Command: gunzip ourarchive.tar.gz

  Result: Brings it back to a normal tar file

- Command: bzip2 ourarchive.tar

  Result: Compressed, although less than gzip = these are text files in the archive and rather small so bzip2 doesn't really show that it can compress more here

  Command: bunzip2 ourarchive.tar.bz2

  Result: Brings it back to a normal tar file

- Command: zip ourthing.zip file1.txt file2.txt file3.txt

  Result: Zip file is created called ourthing.txt

  Command: unzip.txt

  Result: This extracts the files

<u>Create archives and compress OR uncompress and extract in one go</u>

- Command: tar -cvzf ourarchive.tar.gz file[1-3].txt
  - -z = These compresses into the gzip file
  - Note that we named the file .gz so we know its compressed with gzip
- Command: tar -cvbf ourarchive.tar.bz2 file[1-3].txt
  - -b = These compresses into the bzip2 file after creating an archive
  - This time we called it bz2
- Command: tar -xvzf ourarchive.tar.gz

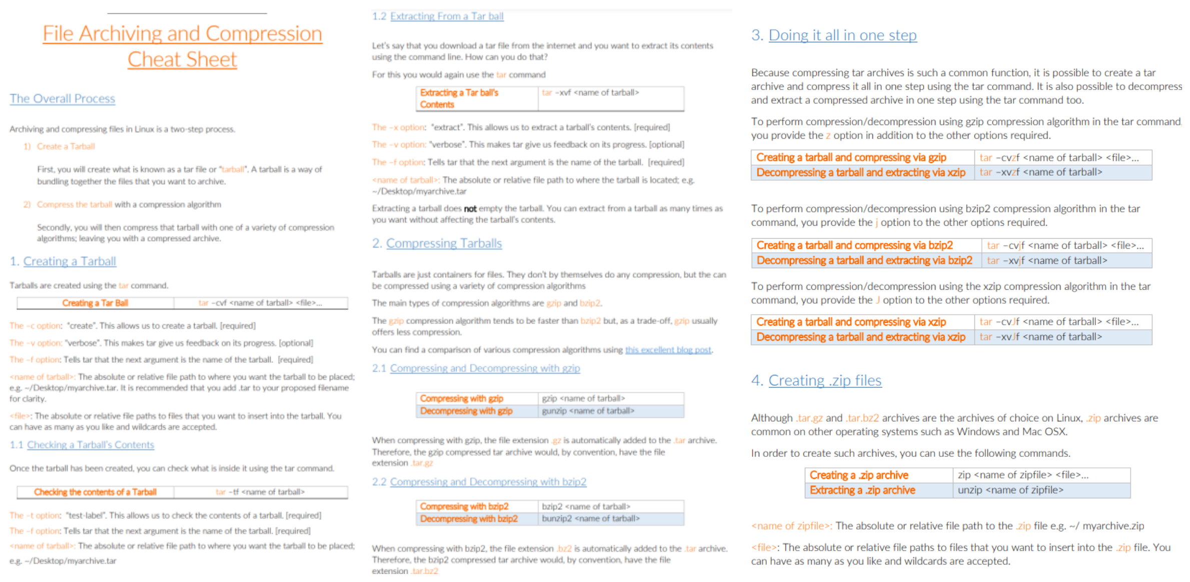
  Result: The files in the compressed gzip tar archive are extracted

- Command: tar - xvjf ourarchive.tar.bz2

  Result: The files in the compressed bzip2 tar archive are extracted

xz is also compression command but hasn't been shown

<u>Cheat sheets</u>



File Archiving and Compression Cheat Sheet

The Overall Process

Archiving and compressing files in Linux is a two-step process.

1) Create a Tarball

   First, you will create what is known as a tar file or "tarball". A tarball is a way of bundling together the files that you want to archive.

2) Compress the tarball with a compression algorithm

   Secondly, you will then compress that tarball with one of a variety of compression algorithms; leaving you with a compressed archive.

1. Creating a Tarball

Tarballs are created using the tar command.

Creating a Tar Ball	tar –cvf <name of tarball> <file>...

The –c option: "create". This allows us to create a tarball. [required]

The –v option: "verbose". This makes tar give us feedback on its progress. [optional]

The –f option: Tells tar that the next argument is the name of the tarball. [required]

<name of tarball>: The absolute or relative file path to where you want the tarball to be placed; e.g. ~/Desktop/myarchive.tar. It is recommended that you add .tar to your proposed filename for clarity.

<file>: The absolute or relative file paths to files that you want to insert into the tarball. You can have as many as you like and wildcards are accepted.

1.1 Checking a Tarball's Contents

Once the tarball has been created, you can check what is inside it using the tar command.

Checking the contents of a Tarball	tar –tf <name of tarball>

The –t option: "test-label". This allows us to check the contents of a tarball. [required]

The –f option: Tells tar that the next argument is the name of the tarball. [required]

<name of tarball>: The absolute or relative file path to where you want the tarball to be placed; e.g. ~/Desktop/myarchive.tar

1.2 Extracting From a Tar ball

Let's say that you download a tar file from the internet and you want to extract its contents using the command line. How can you do that?

For this you would again use the tar command

Extracting a Tar ball's Contents	tar –xvf <name of tarball>

The –x option: "extract". This allows us to extract a tarball's contents. [required]

The –v option: "verbose". This makes tar give us feedback on its progress. [optional]

The –f option: Tells tar that the next argument is the name of the tarball. [required]

<name of tarball>: The absolute or relative file path to where the tarball is located; e.g. ~/Desktop/myarchive.tar

Extracting a tarball does **not** empty the tarball. You can extract from a tarball as many times as you want without affecting the tarball's contents.

2. Compressing Tarballs

Tarballs are just containers for files. They don't by themselves do any compression, but the can be compressed using a variety of compression algorithms

The main types of compression algorithms are gzip and bzip2.

The gzip compression algorithm tends to be faster than bzip2 but, as a trade-off, gzip usually offers less compression.

You can find a comparison of various compression algorithms using this excellent blog post.

2.1 Compressing and Decompressing with gzip

Compressing with gzip	gzip <name of tarball>
Decompressing with gzip	gunzip <name of tarball>

When compressing with gzip, the file extension .gz is automatically added to the .tar archive. Therefore, the gzip compressed tar archive would, by convention, have the file extension .tar.gz

2.2 Compressing and Decompressing with bzip2

Compressing with bzip2	bzip2 <name of tarball>
Decompressing with bzip2	bunzip2 <name of tarball>

When compressing with bzip2, the file extension .bz2 is automatically added to the .tar archive. Therefore, the bzip2 compressed tar archive would, by convention, have the file extension .tar.bz2

3. Doing it all in one step

Because compressing tar archives is such a common function, it is possible to create a tar archive and compress it all in one step using the tar command. It is also possible to decompress and extract a compressed archive in one step using the tar command too.

To perform compression/decompression using gzip compression algorithm in the tar command, you provide the z option in addition to the other options required.

Creating a tarball and compressing via gzip	tar –cvzf <name of tarball> <file>...
Decompressing a tarball and extracting via xzip	tar –xvzf <name of tarball>

To perform compression/decompression using bzip2 compression algorithm in the tar command, you provide the j option to the other options required.

Creating a tarball and compressing via bzip2	tar –cvjf <name of tarball> <file>...
Decompressing a tarball and extracting via bzip2	tar –xvjf <name of tarball>

To perform compression/decompression using the xzip compression algorithm in the tar command, you provide the J option to the other options required.

Creating a tarball and compressing via xzip	tar –cvJf <name of tarball> <file>...
Decompressing a tarball and extracting via xzip	tar –xvJf <name of tarball>

4. Creating .zip files

Although .tar.gz and .tar.bz2 archives are the archives of choice on Linux, .zip archives are common on other operating systems such as Windows and Mac OSX.

In order to create such archives, you can use the following commands.

Creating a .zip archive	zip <name of zipfile> <file>...
Extracting a .zip archive	unzip <name of zipfile>

<name of zipfile>: The absolute or relative file path to the .zip file e.g. ~/ myarchive.zip

<file>: The absolute or relative file paths to files that you want to insert into the .zip file. You can have as many as you like and wildcards are accepted.

TASK AUTOMATION AND SCHEDULING

Creating Bash Scripts

<u>Using the bash command</u>

- Create: nano our_script.sh
  - .sh is for bash script
  - If you do file command on the file created, it isn't a special file
- Make it a special file - make it read with a certain interpreter i.e., as bash script; there is a specific method to do this outlined below:
  - #!/usr/bin/bash
    - This must be the first line when opening with nano
    - #! = sha bang
    - /bin/bash = path to interpreter (can be found with
    - Now using the file command it is found to be interpreted as a bash script
      - Interestingly, if you used the file path to python using which pyhton3, and put this as the path, the shell will now interpret it as a python script
    - Now anything written into here is interpreted as though it is the normal shell
- Start writing lines of commands or things you would do in their and then execute this bash script:
  - Using nano to write in our_script:

    mkdir ~/Desktop/cool

    touch ~/Desktop/cool/file{1..100}

ls -lh ~/Desktop/cool/ > ~/Desktop/cool.log
Command: bash ~/Desktop/our_script.sh
Result: commands are run all in one go

- Start over: delete lines in our script, remove cool/ and cool.log from resktop
  Use nano to create backup_script on desktop writing the #!/usr/bin/bash
  Write: tar -czf ~/Desktop/backup.tar.gz
  ~/{Desktop,Downloads,Documents,Pictures,Videos} 2>dev/null
  - Notice the 2>dev/null AND the lack of v in -czf
    This is to redirect the error output to stop the error from showing in the terminal and sends it to the bit bucket which deletes anything that's sent to it and to stop the verbose part of the command which shows more stuff in the terminal

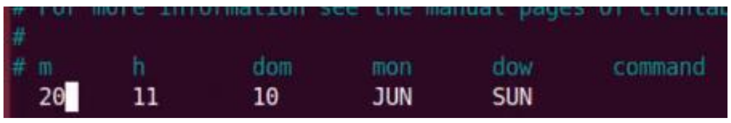
Using the chmod command to turns bash scripts into commands

- Make shell script commands more friendly
  - Move backup.sh into a new directory called bin in the home directory and rename it to backup (as all files in here will be bash scripts so no confusion)

  Command: chmod +x backup
  - This adds execute permissions on backup (ls shows it has gone green)
- Issue: If you try to run the backup command it wont work as the shells search path doesn't include the ~/bin/ folder
  Command: nano ~/.bashrc
  Add a line to the bottom: PATH="$PATH:$HOME/bin"
  - This is adjusting the path.
  - PATH=" " This means the current path will become
  - $PATH: This means the current path
  - $HOME/bin This means adding the bin/ folder to that current path

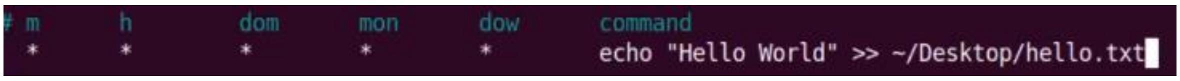
  Close the terminal and open another one. Check with echo $PATH to see the path for the bin/ folder and if the terminal recognises it
- Command: backup
  - This now runs the command within the backup bash script!

Scheduled bash scripts using cron command (automation)

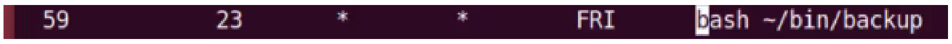
- Each user has a crontab which is a text file which lists the commands and scripts that are run automatically by a user and when they will run
- Command: crontab -e
  - This opens the crontab and it will ask what editor to use so choose 1 for nano
    - If you want to change this, go onto home directory and look at the hidden file called .selected_editor which stores the preference for this. To change the path to the editor you want, nano .selected_editor and put the path you want it in their. OR type the select-editor command and change it
  - The lines written starting with a hash indicate notes and are not actually used by the cron utility
  - There are 5 columns which must be written into for scheduling and the last column is for the command; they are split with spaces but these spaces may be larger than one space so you can make it larger to organise how you wish. Column 1= The minute you want to run, 2 = The hour of the day, 3 = Day of the month, 4 = The months e.g., DEC, 5 = Day of the week e.g., SUN
  - 

```
# For more information see the manual pages of cronta
#
# m      h          dom        mon        dow        command
  20     11         10         JUN        SUN
```

    - This command would run on 10/06 at 20:11 if this date fell on a Sunday this year
  - Example command:



```
# m      h          dom        mon        dow        command
  *      *          *          *          *          echo "Hello World" >> ~/Desktop/hello.txt
```

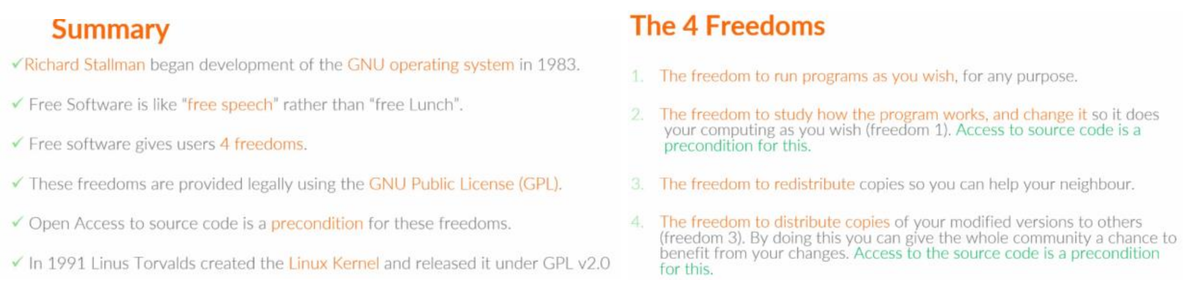
    This command will run every minute of every hour of everyday of the month of every month of every day of the week of every year; essentially every minute of the year
  - If you want: 0,15,30,45 in the minutes to execute the crom job, then you can write run every 15 mins with
    - */15 = Run every 15 mins
  - 

```
  59            23         *          *          FRI          bash ~/bin/backup
```

    - Run a crontab -e of a backup every Friday

OPEN SOURCE SOFTWARE

The GNU Project



Summary

✓Richard Stallman began development of the GNU operating system in 1983.

✓ Free Software is like "free speech" rather than "free Lunch".

✓ Free software gives users 4 freedoms.

✓ These freedoms are provided legally using the GNU Public License (GPL).

✓ Open Access to source code is a precondition for these freedoms.

✓ In 1991 Linus Torvalds created the Linux Kernel and released it under GPL v2.0

The 4 Freedoms

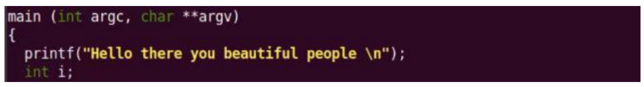
1. The freedom to run programs as you wish, for any purpose.
2. The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1). Access to source code is a precondition for this.
3. The freedom to redistribute copies so you can help your neighbour.
4. The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

- All the commands that we used on Linux are part of the GNU project and the Linux Kernel is the final part which is the interface between the hardware and the software
  - Command uname tells us about the computer like the hardware and the operating system, e.g.,
    - uname -o = Tells you the operating system
      Result: GNU/linux
- Can look at the code the computer runs on GNU.org
  - Navigating to the software tab and to the All-GNU Packages which shows all the code running on the GNU operating system including the Linux Kernel called Linux Libra (this has all the source codes to it in there)
  - You can find all the commands that we've used like ls grep etc in the coreutils package - these can be downloaded, edited and compiled then installed

Compiling software from source code

This is a manual process of compiling and editing software yourself, but you can instead use software repositories (this is the usual option - look at the next section)

- Download the coreutils latest version in the GNU website into the Downloads folder then extract it using:
  - tar -xJf coreutils-9.1.tar.xz
    - The j option extract bzip2 and the J options extracts xzip
  - This extracts it into the downloads folder
  - Cd into this and ls shows a file called src
    - This is where the source code is kept
  - cd into src: then ls | less
    - This shows a large amount of source code; notice .c at the end indicating it they are written with the c programming language
  - ls | grep ls
    - You see a file called ls.c = this is the source code for the ls command
- We can modify the ls.c code and recompile it into a running software (enter it with nano), then install the coreutils package with the modified code

- If you modify the main function, then you can change what the command does; for this command this is at line 1443; use the line function in nano to get there
- At the main function: 

```
main (int argc, char **argv)
{
  printf("Hello there you beautiful people \n");
  int i;
```

- To turn it into a runnable programme it now must be compiled into a machine language so we need a specific compiler for the c language ; gcc (GNU c-compiler) is the compiler in linux so we use this command to get it:

  Command: sudo apt-get install gcc
- This tells the computer to look at the software repositories available for ubuntu and donnwlad&install the gcc package
  - This was already in my ubuntu so nothing to install really
- Different computers have different architectures so installation of the code must be configured to suit the machine; use the configure bash script
  - Go back from the src folder to the main coreutils folder and ls for configure which will be green as it is a script
  - This will also create the make file which is responsible for installing the new package we want to install
  - To run this now, we need to use the make command which we will download with

    Command: sudo apt-get install make
      - Was already installed
  - Command: make

    Result: Now it is compiling all the c files that haven't been combined recently and are outstanding (including the ls.c file) - its making them binary so they can be run on the compute
- Whats left is to install the machine code to make it work

  Command: sudo make install

  <u>This installs each of the pieces of software that came with the software from the coreutils-9.1 package</u>

  Close and re-open the terminal to see the effect

- Command @~/: ls

  Result: The ls command with "hello there you beautiful people"
- To change it back to normal, you re-edit the source code, recompile the with make, then re-install with make install
  - Cd ~/Downloads/coreutils-9.1/src
  - nano ls.c, go to the line with the main function and remove the extra line
  - Go back to the coreutils.9.1 folder and execute:

make && sudo make install

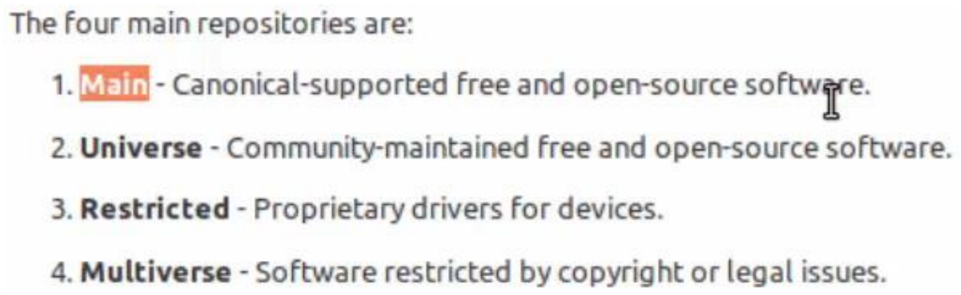
- This is much faster since the gcc was configured before for the package and make is aware to only recompile code that was affected by the change to the ls.c

- All is back to normal for ls now

Software repositories

This is a big library with software which you can install from and update from.

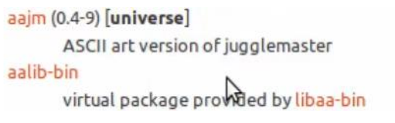
- There are 4 repositories:



The four main repositories are:

1. Main - Canonical-supported free and open-source software.
2. **Universe** - Community-maintained free and open-source software.
3. **Restricted** - Proprietary drivers for devices.
4. **Multiverse** - Software restricted by copyright or legal issues.

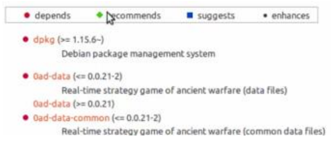
  - Main: most reliable so you should try to source from here more often as canolical manages it and they created ubuntu
  - Universe: this is maintained by the ubuntu community so it can be more prone to bugs theoretically although usually good
  - Restricted: ensures devices are useable outside the box usually (usually not free software)
  - Multiverse: may or may not have the source code (usually not free-software)

- You can find the repositories on packages.ubuntu.com
  - You will different links depending on the version of ubuntu you are running by using the command: lsb_release -a
    - In my case its jammy
  - In here you find a lot categories and loads of packs within the categories
  - Scroll to all packages at the bottom and load the web page
  - 

    aajm (0.4-9) **[universe]**
        ASCII art version of jugglemaster
    aalib-bin
        virtual package provided by libaa-bin

    The universe in square brackets tells us the repository is from, the aalib-bin has no brackets so this means that it is from the main repository

- Opening the first repository called 0ad

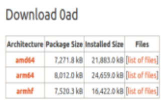
- 

  ● depends ◆ recommends ■ suggests • enhances

  ● dpkg (>= 1.15.6~)
  Debian package management system

  ● 0ad-data (<= 0.0.21-2)
  Real-time strategy game of ancient warfare (data files)
  0ad-data (>= 0.0.21)

  ● 0ad-data-common (<= 0.0.21-2)
  Real-time strategy game of ancient warfare (common data files)

  The red dot at the front tells us that in order for the 0ad package to work, you need to install all the packages with the red dot in front since its called "depend"
- The green diamond is recommended since it will function without it but not how it should, suggest blue square only tells you of optional stuff you may want to check, enhance is not needed but can add functions
- 

  Download 0ad

Architecture	Package Size	Installed Size	Files
amd64	7,271.8 kB	21,883.0 kB	[list of files]
arm64	8,012.0 kB	24,659.0 kB	[list of files]
armhf	7,520.3 kB	16,422.0 kB	[list of files]

  At the bottom you see the size of the file and files to download for different architectures
  - You can check the computers architecture using:

    Command: uname -m

    Result: x86_64

    This means amd64 is the best for this operating system

- Package managers: A command line tool manages all these package relationships and it is called apt in ubuntu

The Apt Cache

You can search for packages to install with apt and use cache to speed things up. Lets say we wanted to search packages that help us work with Microsoft word documents, and they end with docx.

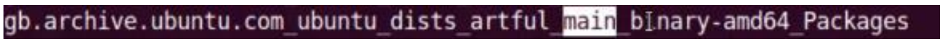
- Command: apt-cache search docx
  - You can see info and combine with grep to find what you are looking for e.g., text: apt-cache search docx | grep text

Now say you are interested in the docx.txt

- Command: apt-cache show docx.txt | less
  - This tells you more about it and where its from and what it does

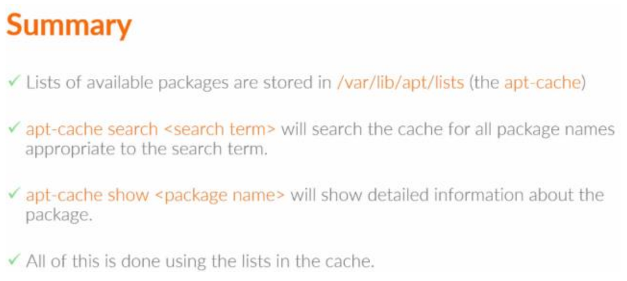
Both of these above commands can run this without an active internet connection since it is stored as cache, hence the use of apt- cache.

- Location of the lists of cache: /var/lib/apt/lists/
  - Here you can see for instance you wanted to find the gcc package, we know this is in the main repository

    ```
    gb.archive.ubuntu.com_ubuntu_dists_artful_main_binary-amd64_Packages
    ```

- Opening this with nano, we can look for the gcc package and it shows that when you execute e.g., apt -cache show gcc, this file contains that information



Summary

✓ Lists of available packages are stored in /var/lib/apt/lists (the apt-cache)

✓ apt-cache search <search term> will search the cache for all package names appropriate to the search term.

✓ apt-cache show <package name> will show detailed information about the package.

✓ All of this is done using the lists in the cache.

Updating the cache and upgrading software

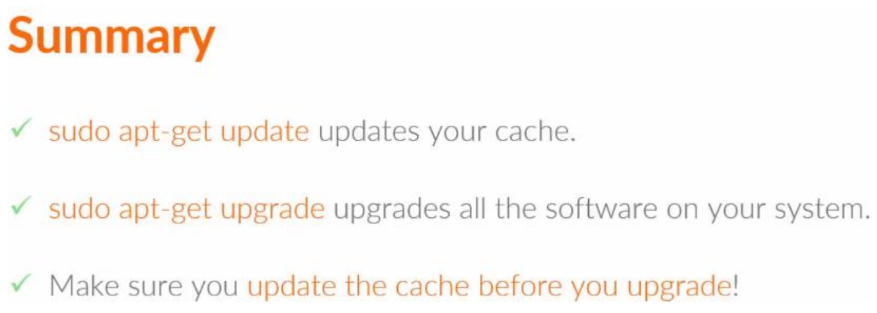
This is how you can keep cache and all the software on your system up to date.

To update cache:

- Command: sudo apt-get update (must be connected to the internet)

To update software that is out of date:

- Command: sudo apt-get upgrade



Summary

✓ sudo apt-get update updates your cache.

✓ sudo apt-get upgrade upgrades all the software on your system.

✓ Make sure you update the cache before you upgrade!

Installing new packages

Say you want a program called xeyes

- apt-cache search xeyes
  - choose the one you are interested in
- apt-cache show x11-apps | less
  - we find that the package provides miscellaneous assortment of X applications, where you find that xeyes is there being a demo program that tracks the pointer with eyes

- Command: sudo apt-get install x11-apps
  - Notice we use apt-get to install it straight from the web
  - This is a precompiled package meaning it is a binary package

- Command: xeyes
    - The command is now installed
    - It will also be a part of the manual pages as well

Also xman was downloaded with it, which is a nice way to graphically look at the manual pages.

Downloading source code

- Command: sudo nano /etc/apt/sources.list
    - This is a configuration file that the package file uses to determine which sources it downloads (i.e. repositories)
    - Delete the # in front of the deb-src which by default doesn't let you for efficiency but now activates the ability for the package manager to download source code
- Now you need to update the cache so: sudo apt-get update
    - Now all the source code will be downloaded
- Now you need to install: sudo apt-get install dpkg-dev
    - Once this is done you can now install source code
- To get the source code of the xeyes package which we installed from the x11-apps in the previous section, then we must use the following command:
    - sudo apt-get source x11-apps
        - This creates 3 downloaded things
    - cd into the directory and ls to find the xeyes then cd into this
    - nano into the xeyes.c and you can see the source code of the file now

Uninstalling packages

How to uninstall packages.

- Command: sudo apt-get remove <package name>
    - This is possible but the configuration files are left behind

Rather use this

- Command: sudo apt-get purge <package name>
    - This gets rid of the package and the configuration files

To get rid of the dependencies as well, use

- Command: sudo apt-get autoremove
    - This removes any downloaded un needed dependencies across the whole system

To save even more space now from stored compressed files use

- Command: sudo apt-get clean
    - These are usually found in the /var/cache/apt/archives
    - Can use autoclean to get rid of those things that you can no longer download from the repositries rather than all

Fin.