

</ CSE 314 Operating System Sessional

/>



Session 1

Introduction to the Linux CLI
(Command Line Interface)

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1

</ The Shell

- GUIs are limited
 - Can only interact what the programmer allows in the UI
 - Makes easy tasks easy
- Textual interface = Full advantage
 - Also known as “Shell” (command line interface)
 - Various implementations
 - Linux / MacOS: Bourne Again SHell (**bash**)
 - Makes difficult tasks possible!

</ The Shell

- `username@machine_name:~$`
 - Your username
 - The name of the machine you are on
 - The directory you are in (~)
 - \$ indicates you are a normal user
 - # ⇒ superuser (“Run as administrator”)

</ Handy Shortcuts

- `<CTRL> + <ALT> + T` : Open terminal
- `<CTRL> + D` : Close terminal
- `<CTRL> + L` : Clear terminal screen
- `<CTRL> + A/E` : Jump to **beginning/end** of line
- `<CTRL> + K/U` : Delete everything **ahead/before** of cursor
- `<CTRL> + Y` : Paste text cut by `<CTRL> + K/U`
- `<CTRL> + LEFT / RIGHT` : Move by word
- `TAB` : Auto-complete matching file names
- `<CTRL> + C` : Cancel current command
- `UP / DOWN` : Cycles between recently used commands

</ Copy & Paste

- Double click (or select) something with a mouse to copy and click the middle button to paste
- <CTRL> + SHIFT + C => Copy
- <CTRL> + SHIFT + V => Paste



</ Some Basic Commands

- date
- echo
 - Simply prints the arguments passed
 - Hello world → Two separate arguments
 - “Hello world” → A single argument
 - Can use quotes (“”) or escape sequences (\)

</ Environment Variables

- Where to find these programs (date, echo, ...)?
 - Environment Variable
 - \$PATH ⇒ Lists the paths to look for commands
 - Can modify it to include more directories
 - Usually in dotfiles (aka shell configuration files)
 - .bashrc, .profile, .bash_aliases
- which CMD ⇒ Prints path to CMD if it exists

</ Exploring the Shell

- pwd
 - Prints the path of the current directory (Print Working Directory)
 - Path on the shell is a sequence of folders, separated by /
 - `/home/terrarium`
 - `/` ⇒ Top of the filesystem
 - aka the “Root” directory
- cd pathname
 - Moves to the directory under pathname
 - Path can be absolute or relative

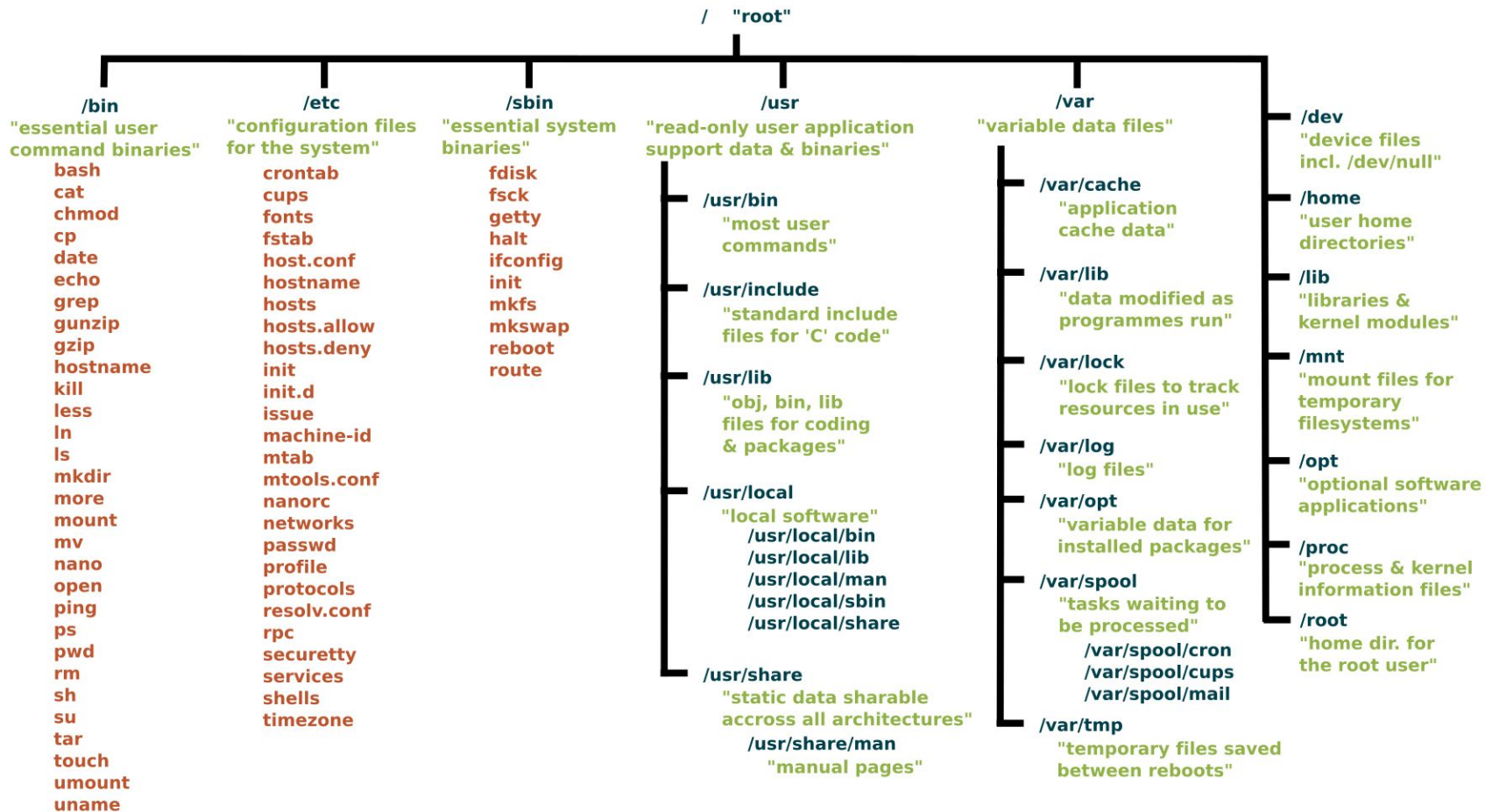
</ Paths

- Absolute path
 - Starts with /
 - Specifies the full location
- Relative path
 - Based on the current working directory
 - . current directory
 - .. parent of the current directory
- Home directory represented by ~
- Previously working directory represented by -
- Just type “cd” without anything else ⇒ Takes you to home directory

</ Which one to use?

- Use whichever is shorter!
- In scripts or programs, use absolute paths (recommended)

</ Linux File System



</ Helpful Commands

- `cd` : Change directory
- `ls` : List contents in directory
- `mkdir` : Make directory
- `mv` : Move or change name of something
- `touch` : Create files
- `rm` : Remove something
- `cp` : Copy something

</ Listing files and directories

- `ls [OPTION]... [FILE]...`
 - Lists information about directory or file
 - `-a` Lists the hidden files as well
 - `-l` List in details
 - `-R` List subdirectories recursively
 - `-S` Sort by file size
 - `-F` Append `/` for dirs, `*` for executables, `@` for symbolic links

</ Listing files and directories

- `ls -la`

```
terrarium@asus:~/playground/cse314$ ls -la
total 5852
drwxrwxr-x  3 terrarium terrarium    4096 Aug 31 01:49 .
drwxrwxr-x 10 terrarium terrarium    4096 Aug 31 01:36 ..
-rw-rw-r--  1 terrarium terrarium      2 Aug 31 00:10 1.txt
-rw-rw-r--  1 terrarium terrarium      2 Aug 31 00:16 2.txt
-rw-rw-r--  1 terrarium terrarium     21 Aug 31 01:02 input_nums.txt
-rw-rw-r--  1 terrarium terrarium     57 Aug 31 01:04 input_words.txt
-rw-rw-r--  1 terrarium terrarium 5960838 Aug 29 01:11 ostep.pdf
drwxrwxr-x  3 terrarium terrarium    4096 Aug 31 02:53 test
```

File permissions

Owner

Group

Size in
bytes

Last
modification
time

Content

No. of hard links

</ Listing files and directories

- Everything is a file in Linux
- Directories are a special type of file
 - contains a list of filenames and their corresponding inode numbers
 - inode is a data structure that stores information about files such as permissions, ownership, and file location on the disk.
- The size 4096 bytes is the smallest unit of space that the filesystem can allocate
 - aka “Default Block Size”

</ Creating directories

- `mkdir [OPTION]... DIRECTORY...`
 - `-p` Create parent directories as needed, do nothing if it exists
 - `-m` Provide file mode like: `rw-rw-r--` (more on this later!)
 - `-v` Verbose output

</ Creating files

- `touch [OPTION]... FILE...`

</ Moving and renaming

- `mv [OPTIONS]... SRC... DEST`
 - `-i` Prompt before overwriting
 - No “`-r`” option

</ Moving and renaming

- `mv OLD NEW`
 - if 'new' is a directory: 'old' is moved inside 'new'
 - if 'new' does not exist: 'old' is renamed to 'new'
 - if 'new' is a file, and 'old' is file: 'old' replaces 'new', and the previous content of 'new' is lost forever!!
 - if 'new' is a file, and 'old' is directory: it's an error (cannot overwrite a file with a directory)

</ Copying files and directories

- `cp [OPTION]... SRC... DEST`
 - `-r` Recursive copy
 - `-i` Interactive prompt

</ Removing

- `rm [OPTION]... [FILE]...`
 - No way to undo!
 - `-f` Never prompt (needed for write-protected files)
 - `-i` Always prompt
 - `-r` Remove recursively (needed for directory removal)
 - `-v` Verbose print
- Kill your system (DON'T!): `sudo rm -rf /`
- Use preventive measures (aliases)



</ Exercise

- Create 4 files named `project_<your_id>.java`, `project_<your_id>.js`, `project_<your_id>.html`, and `project_<your_id>.css`.
- Then, create a directory called `web_project`. Inside `web_project`, create subdirectories named `backend`, `frontend`, and `styles`. Move the `.java` file to `backend`, the `.js` and `.html` files to `frontend`, and the `.css` file to `styles`.

</ Exercise

- Create 4 files named `old_report.docx`, `draft.docx`, `old_photo.png`, and `snapshot.png`. Create two directories named `documents` and `images`.
 - Rename `draft.docx` to `final_report.docx`.
 - Move all `.docx` files to the `documents` directory.
 - Move all `.png` files to the `images` directory.
 - List the contents of both `documents` and `images` directories.
 - Now move all files beginning with `old` to a new directory named `archived`

</ I/O Redirection

- Default input from keyboard, default output to screen
- I/O redirection allows us to change this
 - < get input from a file other than keyboard (stdin)
 - > output to a file other than the screen (stdout)
 - >> append output to a file
 - 2> Redirects stderr
 - &> Redirects both stderr and stdout
 - `ls /nonexistent &> all_output.txt`
 - `ls /nonexistent > all_output.txt 2>&1`
 - redirect stderr to wherever stdout is going

</ I/O Redirection

- `cat nonexistent.txt 2> error.log`
 - Here, the `cat` program attempts to open the given file. This will redirect the error message if the given file does not exist
- `cat < nonexistent.txt 2> error.log`
 - The same does not apply here. This is because the shell itself attempts to open the file, hence the error is generated by the shell, not the `cat` program.
 - Workaround: `{ cat < nonexistent.txt; } 2> error.log`
 - The command is placed inside a subshell `{}`. The error generated by the shell is captured by `2>`

</ Piping

- Pass the output of one command directly as input to another command
- `command1 | command2 | command3 ...`
- `ls /bin /usr/bin | sort | uniq | less`
 - List all files in /bin and /usr/bin, sort them, remove duplicates, then display using less pager
- `sort path/to/file | uniq -c | sort -nr`
 - Display number of occurrences of each line, sorted by the most frequent
- Extremely powerful and versatile!
- Difference between `>>` and `|`?
 - `>>` Append output to files
 - `|` Chain commands

</ Piping

- Counting the frequency of words from a file:
- `tr -c '[:alnum:]' '\n' < file.txt | tr '[:upper:]' '[:lower:]' | sort | uniq -c | sort -nr | head -10`
- Explanation:
 - `tr` takes two arguments: PATTERN1 and PATTERN2, and it replaces PATTERN1 with PATTERN2
 - `tr -c '[:alnum:]' '\n'` this replaces all non-alphanumeric characters (tabs, spaces, etc) with newlines
 - `tr '[:upper:]' '[:lower:]'` converts all words to lowercase
 - The rest of the command sorts and displays the words by frequencies

</ How to remember the options?

- `CMD --help`
- `man CMD`
- `tldr CMD` (more on this later)
- Copilot for bash (more on this later)

</ less Pager

- Shows a file's contents one screen at a time
- Real-time monitoring:
`less +F FILENAME`

Shortcuts	Action
Down Arrow, Enter, e, j	One line forward.
Up Arrow, y, k	One line backward.
Space bar, Page Down	One page forward.
Page Up, b	One page backward.
Right Arrow	Scroll right .
Left Arrow	Scroll left .
Home, g	Jump to the beginning of the file.
End, G	Jump to the end of the file.
/[string]	Search forward for the specified string.
?[string]	Search backward for the specified string.
n	Next match during a search.
N	Previous match during a search.
q	Quit less .

</ Viewing Files

- less
- head
- tail
 - Follow with `-f` or `-F`
- cat
- wc

</ Executing scripts

- In order to execute any script, we type `./script_name`
- `./` is shorthand for "the current directory." When you type `./script_name`, you are explicitly telling the shell to look in the current directory and execute `script_name` from there.
- Without the `./`, the shell would only look for `script_name` in the directories listed in the `PATH` variable.
- The script must have a special line called “shebang” or “hashbang”

</ Executing scripts

- The shebang is a character sequence at the beginning of a script file that indicates which interpreter should be used to execute the script. It is typically written as `#!` followed by the path to the interpreter.
- `#!/usr/bin/python3`
 - This shebang line tells the system to use `/usr/bin/python3` to interpret the script.



1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1

</ Permissions

- Access rights to files and directories are defined in terms of read access, write access, and execution access.
- Type ``ls -la`` in the terminal:

```
terrarium@asus:~/playground/cse314/test$ ls -la
total 16
drwxrwxr-x 3 terrarium terrarium 4096 Aug 31 01:50 .
drwxrwxr-x 3 terrarium terrarium 4096 Aug 31 01:49 ..
-rw-rw-r-- 1 terrarium terrarium    0 Aug 31 01:49 1.txt
-rwxrwxr-x 1 terrarium terrarium   35 Aug 31 01:49 hello.py
drwxrwxr-x 2 terrarium terrarium 4096 Aug 31 01:50 xv6
terrarium@asus:~/playground/cse314/test$
```

</ Permissions

d r w x r w x r w x

user group other

A diagram showing the breakdown of permissions. The string 'd r w x r w x r w x' is displayed with the first character 'd' in white and the subsequent characters in groups of three. Brackets are placed under each group of three characters: 'r w x' (yellow), 'r w x' (green), and 'r w x' (pink). An arrow points from the first bracket to a box on the left. Below the brackets, the labels 'user', 'group', and 'other' are written in yellow, green, and pink respectively, corresponding to the colors of the brackets.

d	Directory
l	Symbolic link
s	Socket

r	read
w	write
x	execute

</ Users in Linux

- **Root User:** Always present. Highest privilege level
- **Default User:** Created during installation, usually one user with administrative privileges.
- **System Users:** Non-human users that are used by various system services and processes. Limited or no login capabilities and exist primarily for the purpose of running background services or daemons.
- `cut -d: -f1 /etc/passwd`

</ Permissions

r	read
w	write
x	execute

- `-rw-r--r--` means it is a file which has read and write permissions for the owner, read permission for group and read permission for others.
- This means, only the owner of the file can read and write. The rest can only read.
- However, sometimes you may need to write or execute something that you don't have permission for. What do you do in this case?

</ Permissions

r	read
w	write
x	execute

- You can change permissions using **chmod**
- Each permission segment can be set using bits by the owner
 - For example: `chmod 755 FILE`
 - `755 => 111 101 101 => rwx r-x r-x`
- Or by symbolic notation

Octal notation

```
- Give the [u]ser who owns a file the right to e[x]ecute it:
  chmod u+x path/to/file

- Give the [u]ser rights to [r]ead and [w]rite to a file/directory:
  chmod u+rw path/to/file_or_directory

- Remove e[x]ecutable rights from the [g]roup:
  chmod g-x path/to/file

- Give [a]ll users rights to [r]ead and e[x]ecute:
  chmod a+rx path/to/file

- Give [o]thers (not in the file owner's group) the same rights as the [g]roup:
  chmod o=g path/to/file

- Remove all rights from [o]thers:
  chmod o= path/to/file
```

d r w x r w x r w x

user group other

</ Permissions

r	read
w	write
x	execute

- You can also change owner or group for a file:
 - `chown username <FILE>`
 - `chgrp groupname <FILE>`

d r w x r w x r w x



user group other

Q: Two files may have the same permission bits, but still one needs `sudo` to write, and another does not. Why?

</ Permissions for Directories

r	read
w	write
x	execute

- If you have write permission for a directory, you can create new entries (files/folders/etc).
- If you have read permission for a directory, you may list `ls` the directories contents.
- If you have execute permission for a directory, you may change `chdir` into that directory.

d r w x r w x r w x



user group other

</ Permissions

- Switch users: `su [user]`
- The default user belongs to a special group called “sudo”
- Members of the sudo group are granted the ability to execute commands with superuser (root) privileges by using the sudo command.
- View all groups: `getent group`
 - `groupname:passwd:groupID:user1,user2,user3...`

```
terrarium@asus:~/playground/cse314/test$ getent group sudo
sudo:x:27:terrarium
terrarium@asus:~/playground/cse314/test$
```

</ Permissions

- Execute command as a superuser: `sudo CMD`
- Add new user: `sudo adduser USERNAME`
- Delete user: `sudo userdel -r USERNAME`

- `sudo usermod -aG groupname username`
 - `-a`: Appends the user to a group. Without this, the user would be removed from all other groups not listed.
 - `-G`: Specifies the groups to which the user should be added.
- `sudo` can be dangerous if used without caution!

</ Permissions

- How to write to a restricted file using sudo?
 - `sudo echo "hello" > file` ⇒ This doesn't work
 - Redirection (`>`) is not part of the command that `sudo` is applied to
 - `echo "Hello" | sudo tee file`
 - `tee` reads from `STDIN`, writes to `STDOUT` or files provided
 - Since this is a program, not a `STREAM`, we need `|`, not `>>`

</ Aliases



</ Aliases

- Aliases are handy way to create notations for long commands
 - `ll` expands to `ls -aLF`
 - `alias alias_name='command'`
- Aliases are temporary when set within the terminal
 - `alias glnice="git log --oneline --all --decorate --graph"`
 - `alias rm="rm -i"`
 - every `rm` command will be interactive, regardless of whether you use `-rf` or not. This adds an extra layer of safety by preventing accidental deletions.
- On Ubuntu versions 11.04 or later, you can add aliases to `~/.bash_aliases` file to make them permanent.

</ Searching within files

When you grep grep's manual:

```
↳ grep --help | grep "invert"
```



I used the grep to grep grep

</ Searching within files

- `grep [options] pattern [file...]`
 - `pattern`: The string or regular expression you want to search for.
 - `file...`: One or more files where you want to search for the pattern. If no file is specified, `grep` searches in the standard input (e.g., piped from another command).
- `grep "hello" file.txt`
- `grep -i "hello" file.txt` Case insensitive search
- `history | grep "cd"`
- `grep "^start" file.txt` Search for lines starting with "start"
- `grep "end$" file.txt` Search for lines ending with "end"

</ Searching within files

- Regex wildcards are allowed. (`*.??^$`)
- `grep -r "TODO" .` Search recursively in the current directory
- More options:
 - `-n` Display line numbers
 - `-w` Search for word, substring matches are ignored
 - `-c` Count number of matches
 - `-v` Invert match (Find lines that do not have the pattern)
- `grep -n -C 2 "error" logfile.txt`
 - Display the match with line numbers and 2 lines of context
- And many more!

</ Searching files and folders

- `find [path] [expression]`
 - `[path]`: Specifies the directory where the search begins. If omitted, `find` starts in the current directory (`.`)
 - `[expression]`: Defines the criteria for finding files and directories. This can include options like `-name`, `-type`, `-size`, `-mtime`, and more.

</ Searching files and folders

- `find /path/to/search -name "filename.txt"`
 - searches for a file named `filename.txt` within `/path/to/search` and its subdirectories.
 - Recursive by nature!
 - Limit recursion using `-maxdepth NUM`
- `find /path/to/search -name "*.txt"`
 - Searches for all files ending with `.txt`

</ Searching files and folders

- By default, the **find** command searches for both files and directories matching the pattern
- Can specify types:
 - `-type f` Only regular files
 - `-type d` Only directories
 - `find /path/to/search -type d -name "dirname"`
- `find` is even more powerful than you can imagine!

</ Searching files and folders

- `find /path/to/search -type f -name "*.txt" -size +1M`
 - Finds .txt files larger than 1 MB.
- The `-exec` option in the `find` command allows you to execute a command on each file or directory that `find` locates. This is a powerful feature that enables you to automate tasks directly on the files or directories that match your search criteria.

</ Searching files and folders

- `find /path/to/search -type f -name "*.log" -exec rm {} \;`
 - Deletes all .log files but leaves directories and other non-regular files untouched.
 - It uses {} as a placeholder for the matched files or directories and requires \; to terminate the command.
- `find /path/to/search -type f -exec tar -rvf backup.tar {} \;`
 - Creates an archive or backup that includes only regular files and excludes directories
- `find /path/to/search -type f | wc -l`
- Another useful tool is the fuzzy finder (fzf)

</ xargs [optional]

- xargs is a powerful tool that is used to build and execute commands from standard input. It takes input from a pipe or file and passes it as arguments to another command.
- `find . -name "*.txt" | xargs rm`
 - `find . -name "*.txt"` searches for all .txt files in the current directory and its subdirectories.
 - The results (file paths) are passed to xargs, which constructs and executes the `rm` command with those file paths as arguments.
- This is same as `find . -name "*.txt" -exec rm {} \;`
- The former can build commands in parallel, thus more efficient

</ Symbolic Links

- Symlinks are special type of file that contains a path to the target file or directory.
- `sudo ln -s /path/to/your/program /usr/local/bin/program_name`
- Here, `/usr/local/bin` is in `$PATH`. This makes your program system-wide available.
- `exec bash` Reloads the shell
- For interested readers:
<https://askubuntu.com/questions/108771/what-is-the-difference-between-a-hard-link-and-a-symbolic-link>

</ Helpful Commands

- `!!` : Re-run last command
- `sudo` : Run with elevated privileges
- **`clear`** : Clear the terminal
 - `<CTRL> + L` Actually moves to the end of output
- `whoami` : Print username
- `which CMD` : Print path to command

</ Some more...

- `htop` Task manager
- `df` Display free disk space
- `du` Display directory size and its contents
- `ps` Display running processes
- `kill` Terminate a process by its PID

</ Installing Programs

- `apt` : Advanced Package Tool
- `sudo apt update`
 - Refreshes the list of available packages and their versions.
- `sudo apt upgrade`
 - Upgrades all the installed packages to newest version
- `sudo apt install PKG`
- `sudo apt remove PKG`

</ Installing Programs

- `sudo apt autoremove`
 - Removes unused dependencies
- `sudo apt autoclean`
 - Removes old packages that have been superseded by newer versions
- Use with combination!
 - `sudo apt autoremove && sudo apt autoclean`

</ Installing Programs

- To install from standalone installers:
 - `sudo dpkg -i your_file.deb`
- Unzip .tar files
 - `tar -xvf your_file.tar.gz`

</ Too Long Didn't Read?

- `man CMD` Open command manual
 - TOO VERBOSE!
- `CMD --help`
 - No examples!
- Your best friend: `tldr`
 - [Install](#) using pip or npm
 - Do not use apt. You may end up with an old version.

PRACTICE

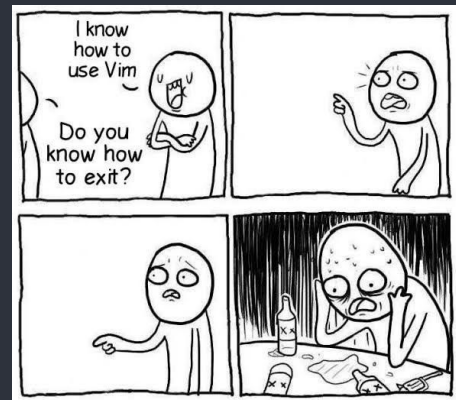
- [https://school.brainhackmtl.org/modules/introduction to terminal/](https://school.brainhackmtl.org/modules/introduction%20to%20terminal/)
- <https://cmdchallenge.com/>

</ Useful tools

- fzf
- tldr
- Copilot for CLI:
<https://docs.github.com/en/copilot/using-github-copilot/using-github-copilot-in-the-command-line>
- tmux
- ngrok You can set-up your own ssh using ngrok
- wget, curl
- vim, nano, emacs

</ Supplementary Lectures and Tools

- <https://missing.csail.mit.edu/>
 - Recommended to watch at least the first 6 lectures (upto and including Version Control)
- <https://explainshell.com/>
- vim tutorial: <https://www.youtube.com/watch?v=ggSyF1SVFr4>
 - Or just type vimtutor in the terminal
- tmux: <https://www.youtube.com/watch?v=DzNmUNvnB04>



</ Easter eggs 🥚

- telnet towel.blinkenlights.nl
- sudo insults: <https://itsfoss.com/sudo-insult-linux/>
- sl
- lolcat (Can be used as a pipe)
- figlet
- cmatrix
- moon-buggy
- nyancat



