Basic Digital Literacy

Content

- Viewing & Navigating
- Creating
- Manipulating
- Installing
- Versioning
- Publishing
- Collaborating

Part – I : Viewing & Navigating

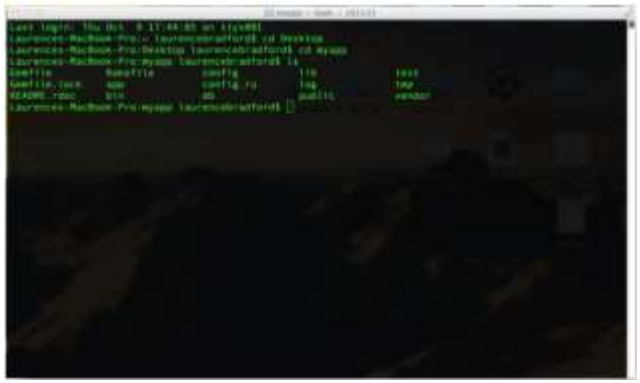
- Introduction: The Terminal
- Moving around
- Reading directories
- Reading files
- Getting help

I - Viewing & Navigating

The Terminal Intro

Command Line Interface (CLI)

Graphical User Interface (GUI)



A means or tool to interact with computer

The Command Line Interface does that too, but it allows you to do it with more precision and power.

You type words and hit enter, the shell interprets those words, and works with the OS kernel and files to execute the command.

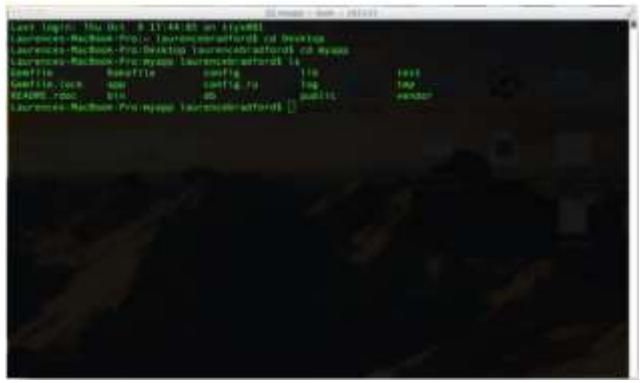
certain buttons perform certain tasks on the computers.

most people use GUI 95% of the time to tell their computer what to do.

I - Viewing & Navigating

The Terminal Intro

Command Line Interface (CLI)



terminal, console

command prompt

- faster
- more powerful
- easy to use and navigate
- the choice of programmers (esp. backend developers)
- offers greater flexibility and control
- useful programs work best on the CLI (git, Github, vsCode etc.)

I - Viewing & Navigating

Moving around

Commands	Explanation
pwd	To know which directory you are in - print working directory - It gives us the absolute path, which means the path that starts from the root.
cd /	root folder (The root is the base of the Linux file system)
cd ~	home folder (/home/username , /users/username)
cd	home folder (/home/username , /users/username)
cd <directory>	change directory (to go to a directory) - (this command is case sensitive, and you have to type in the name of the folder exactly as it is) - **"cd Downloads". → Ok** - the shell will take the second argument of the command as a different one - **"cd Raspberry Pi"** → the directory does not exist **"cd Raspberry\ Pi"**
cd ..	To go back one level up (to go to parent folder)
cd -	The shortcut to last visited directory

I - Viewing & Navigating

Reading directories

Commands	Explanation
ls	To know what files and directories are in the directory you are in
ls -l	To list directories and files in the directory you are in
ls -a	To show hidden files in the directory you are in
ls -la	To list directories, files and hidden files in the directory you are in

I - Viewing & Navigating

Reading directories

```
drwx------@   3 albatros  staff    96 Nov 20  2019 Applications
drwx------+  37 albatros  staff  1184 Jul 29 11:40 Desktop
drwx------+  33 albatros  staff  1056 Jul 28 13:35 Documents
drwx------+ 225 albatros  staff  7200 Jul 28 22:47 Downloads
drwxr-xr-x    4 albatros  staff   128 May 13 01:37 IdeaProjects
drwx------@  70 albatros  staff  2240 Feb 16 16:07 Library
drwx------+  11 albatros  staff   352 Apr 19 18:56 Movies
drwx------+   4 albatros  staff   128 Feb 13 21:22 Music
drwx------+   5 albatros  staff   160 Feb 13 20:55 Pictures
drwxr-xr-x    3 albatros  staff    96 Nov 20  2019 Postman
drwxr-xr-x+   4 albatros  staff   128 Nov 20  2019 Public
-rw-------    1 albatros  staff  2675 May  3 14:49 bzs_academy
-rw-r--r--    1 albatros  staff   590 May  3 14:49 bzs_academy.pub
-rw-r--r--    1 albatros  staff  6816 Apr 19 15:14 evalonly.txt
```

Field	Meaning
`-rw-r--r--`	Access rights to the file. The first character indicates the type of file. Among the different types, a leading dash means a regular file, while a `d` indicates a directory. The next three characters are the access rights for the file's owner, the next three are for members of the file's group, and the final three are for everyone else. The full meaning of this is discussed in Chapter 9.
`1`	File's number of hard links. See the discussion of links at the end of this chapter.
`root`	The user name of the file's owner.
`root`	The name of the group that owns the file.
`32059`	Size of the file in bytes.
`2012-04-03 11:05`	Date and time of the file's last modification.
`oo-cd-cover.odf`	Name of the file.

I - Viewing & Navigating

Reading files

Commands	Explanation
less <filename.xxx>	To view the content of a file and navigate through file's content - esp. for larger files - faster because it does not load the entire file at once and allows navigation though file using page up/down keys.
more <filename.xxx>	- esp. for larger files - slower because it loads the entire file content

to be able to navigate :

---

arrow keys : move down, up, right, left
space bar : move down one page
b : move up one page
g : go to first line
G : go to last line
10g : go to the 10th line
50p or 50% : go to the line half-way through the output
/<term> : search forward from the current position
?<term> : search backward from the current position
n : when searching go to next occurrence
N : when searching go to previous occurrence
q : quit

I - Viewing & Navigating

Reading files

Commands	Explanation
head <filename.xxx>	To show 10 first lines
tail <filename.xxx>	To show 10 last lines
cat <filename.xxx>	to display the contents of a file on terminal.

I - Viewing & Navigating

Getting Help

Commands	Explanation
man <command>	To know more about a command and how to use it. Documentation of the command
<command> --help	- man less - less --help

I - Viewing & Navigating

https://web.mit.edu/mprat/Public/web/Terminus/Web/main.html

I - Viewing & Navigating

Let's Discuss

1. What does the "man" command do? Type in "man rm". How do you scroll and get out?

2. Look at the man page for ls. What does the -l flag do? What does the -a flag do?

3. Type the following command to download and save the contents of google.com:
   curl https://www.google.com > google.html

4. Use "less" to look at the contents of google.html.

5. Look at the man page for less. Read the section on "/pattern".
   Search for the text "class" in the google.html file.

6. How do you jump between words in the terminal?

7. How do you get to the end of a line in terminal?

8. How do you move your cursor to the beginning in terminal?

9. What is the difference between a terminal and shell?

10. What is a flag? Give three examples of flags you have used.

Part – II : Creating

- Creating directories & files
- Basic Authoring (Markdown)

II - Creating

Creating directories & files

Commands	Explanation
mkdir <directory name>	To create a folder or a directory - mkdir Document - “Happy Hacking” → mkdir Happy\ Hacking
touch <filename.xxx>	to create a file It can be anything, from an empty txt file to an empty zip file.
nano <filename.xxx>	You can create a new file or modify a file content using this editor. It is like a text editor NOTE: - If the file already exists, then it is just an editor. - If the file does not exist, then it creates the file first after that you can modify the content Some other editors : **vi, jed, pico**

II - Creating

Creating directories & files

Commands	Explanation
echo ..text.. > filename.xxx echo ..text.. >> filename.xxx	To move some data, usually text into a file. - if you want to create a new text file or add to an already made text file you just need to type in - You do not need to separate the spaces by using the backward slash here - echo hello, my name is Hasan >> new.txt

II - Creating

Exercise

Exercise 1

1. Create a directory with the name of your favorite band.
2. Navigate into the folder
3. Create 4 directories. Each directory should have the name of one of the bands albums.
4. Inside each of those folders (albums), create a .md file fo each of the songs of the album

```
EXPECTED FOLDER STRUCTURE

BLACK SABBATH
  SABOTAGE
    hole_in_the_sky.md
    dont_start.md
    megalomania.md
    ...
  MASTERS OF REALITY
    sweet_leaf.md
    after_forever.md
    ...
  ...
```

Markdown

II - Creating

Exercise

Exercise 1

1. Create a directory with the name of your favorite band.
2. Navigate into the folder
3. Create 4 directories. Each directory should have the name of one of the bands albums.
4. Inside each of those folders (albums), create a .md file fo each of the songs of the album

```
EXPECTED FOLDER STRUCTURE

BLACK SABBATH
  SABOTAGE
    hole_in_the_sky.md
    dont_start.md
    megalomania.md
    ...
  MASTERS OF REALITY
    sweet_leaf.md
    after_forever.md
    ...
  ...
```

Markdown

II - Creating

Markdown (.md or .markdown)

What is Markdown?

Markdown is a text-to-HTML conversion tool for web writers.

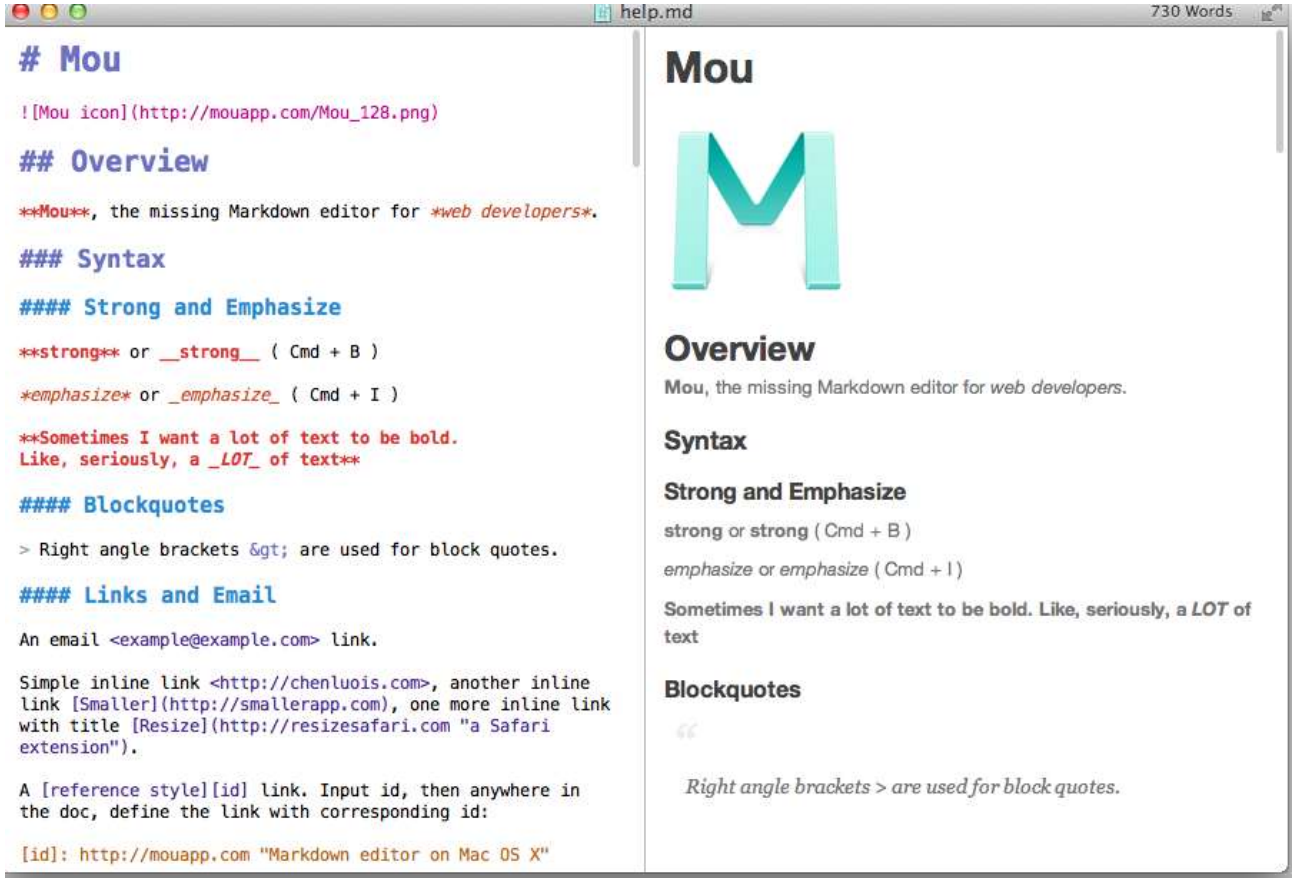
Markdown allows you to write using an easy-to-read, easy-to-write plain text format, then convert it to structurally valid HTML.

“Markdown” is two things:

- a plain text formatting syntax
- a software tool, written in Perl, that converts the plain text formatting to HTML.

II - Creating

Markdown



https://github.com/bradtraversy

II - Creating

Markdown

Let’s make fingers dirty

II - Creating

Exercise

https://github.com/HsnAkk/Markdown_Example/blob/master/Readme_Exe.md

Lemon drizzle slices

A classic British cake from the Bake Off judge, Paul Hollywood's lemon drizzle is a simple traybake, made extra special with feather icing

Nutrition (per slice)

kcal	fat	saturates	carbs	sugars	fibre	protein	salt
236	6g	3g	43g	34g	1g	2g	0.2g

Prep: 25min

Cook: 30min

Cuts into 12 slices

(84 ratings)

Ingredients

- For the cake
  - 70g softened unsalted butter
  - 120g caster sugar
  - 2 medium eggs
  - 140g self-raising flour
  - 1 tsp baking powder
  - finely grated zest 1 lemon
  - 1 tbsp lemon curd
  - 2 tbsp full-fat milk
- For the drizzle topping
  - 30g granulated sugar
  - juice 1 lemon
- For the feather icing
  - 250g icing sugar
  - 3 tbsp water
  - splash of yellow food colouring

Method

1. Heat oven to 180C/160C fan/ gas 4. Line a 20 x 20cm square baking tin with baking parchment.
2. Using an electric whisk, beat the butter and sugar together until pale, light and fluffy. Add the eggs and mix again. Add the flour, baking powder, lemon zest, lemon curd and milk, and mix with a wooden spoon until all the ingredients are thoroughly combined. Pour the mixture into the prepared tin and bake for 25-30 mins or until a skewer comes out clean.
3. Mix the sugar and lemon juice together and pour over the hot cake. Leave to cool in the tin. You can eat the cake as it is, or for a fancy finish, try making this feather icing.
4. Mix the icing sugar with just enough water to give a runny, but not watery, icing. Put a small amount of icing in a separate bowl. Add a few drops of the food colouring to the icing until pale yellow. Spoon into a disposable icing bag.
5. Remove the cake from the tin and peel off the baking parchment. Sit the cake on a wire rack over a baking tray. Spread the white icing over the top. Pipe thin lines of the coloured icing across the width of the cake. Use a cocktail stick to drag through the lines in opposite directions to create a feathered effect. Leave to set before cutting into slices.

Recipe from Good Food magazine, July 2016

By Paul Hollywood

Part – III : Manipulating

- Copying
- Deleting
- Moving & Renaming

III - Manipulating

Copying

Commands	Explanation
cp <file location> <where>	To copy a file through the command line to anywhere. - It takes two arguments: - The first is the location of the file to be copied, - The second is where to copy - cp /home/user/Desktop/new.txt /home/user/destination - cp /home/user/Desktop/dir1/dir2/* /home/user/destination
cp -R <directory location> <where>	To copy a directory through the command line to anywhere. - It takes two arguments: - The first is the location of the directory to be copied, - The second is where to copy - cp –R /home/user/Desktop/Bands /home/user/Documents/Music

III - Manipulating

Copying

Commands	Explanation
To copy the contents of a directory - cp /home/user/Desktop/dir1/dir2/* /home/user/destination	
To copy multiple files at once - cp /home/usr/dir/{file1.xxx,file2.xxx,file3.xxx,file4.xxx} /home/usr/destination/ NOTE : - The syntax uses the cp command followed by the path to the directory the desired files are located in with all the files you wish to copy wrapped in curly brackets and separated by commas. - Make sure to note that there are no spaces between the files. - The last part of the command, /home/usr/destination/, is the directory you wish to copy the files into.	

III - Manipulating

Deleting

Commands	Explanation
rmdir <path to directory>	To delete a directory. NOTE : But **rmdir** can only be used to delete an empty directory. Otherwise, it will give 'File exists / directory not empty' error.
rm <path to file.xxx>	To delete the files - rm file.txt
rm -r <path to directory>	To delete both the folder and the files it contains - rm -r /home/user/Desktop/dir1/dir2
rm –f <path to file.xxx>	to delete the files without prompting for confirmation, regardless of the file's permissions. - rm -f /home/user/Desktop/dir1/dir2/file.txt

III - Manipulating

Moving & Renaming

Commands	Explanation
mv (Linux) move (windows)	- To move files through the command line - To rename a file - mv new.txt file.txt → (renaming) - mv myfile.txt backup → (moving in Linux) - move stats.doc, morestats.doc c:\statistics → (moving in Win, multiple files) - move stats.doc c:\statistics → (moving in Win) - move example.doc d:\ → (moving in Win)

III - Manipulating

Exercise

1. Where am I?
2. Go to “home” directory.
3. Go to the “Desktop” and create a directory called “Project-1”
4. Create files called “index.html”, “style.css”, "main.js”, “Readme.md”, “text.txt”
5. Change the name of “text.txt” to “my_project.txt”
6. Write the code below in “index.html” file.

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="style.css">
</head>
<body>
<h1>Hello World</h1>
<p>This is our first WebDev study</p>
</body>
</html>
```

7. Display the contents of “index.html”on the console.
8. Write the code below in “style.css” file.

```
h1{color:red;}
p{color:blue;}
```

III - Manipulating

Exercise

9. Create two new directories called “AboutUs” and “Services”
10. Copy “index.html”, “style.css” and “main.js” to directory “AboutUs” in one line
11. Copy “index.html”, “style.css” and “main.js” to directory “Services” in one line
12. Go to “Services” directory and list the content with hidden files as well
13. Change the “index.html” file content “<h1>Hello WORLD</H1>” as “<h1>Services Page</H1>” in Services directory
14. Go to “AboutUs” directory and change the “index.html” file content “<h1>Hello WORLD</H1>” as “<h1>AboutUs Page</H1>” in Services directory
15. Display the contents of “index.html” in AboutUs directory on the console.
16. Go to the Project-1 directory and write some explanation about project in “my_project.txt” file
17. Copy the directory “AboutUs” with its all content to a new directory called ”Contact”

III - Manipulating

Exercise - Answers

1. pwd
2. cd or cd ~
3. cd Desktop mkdir Project-1
4. cd Project-1 touch index.html touch style.css touch main.js touch text.txt
5. mv text.txt my_project.txt
6. nano index.html (write the code inside the editor)
7. cat index.html (other option : less index.html or more index.html)
8. nano style.css (write the code inside the editor)
9. mkdir AboutUs mkdir Services
10. cp {index.html,style.css,main.js} AboutUs
11. cp {index.html,style.css,main.js} Services
12. cd Services ls –la
13. nano index.html (correct the content in editor)
14. cd .. cd AboutUs nano index.html (correct the content in editor)
15. cat index.html
16. echo This is my first project about web development with linux command line codes >> my_project.txt
17. cp -r AboutUs/ Contact

III - Manipulating

Questions

1. Display the contents of 'passwd' file on the screen interactively (so you can search, scroll up and down)
   -
2. How do you get to the home folder from anywhere
   -
3. List contents of 'series3' directory in details.
   -
4. In one commandline make a copy of directory 'series3' and all its contents under name 'series3copy'.
   -

III - Manipulating

Questions - Answers

1. Display the contents of 'passwd' file on the screen interactively (so you can search, scroll up and down)
   - less passwd

2. How do you get to the home folder from anywhere
   - cd

3. List contents of 'series3' directory in details.
   - ls -l series3

4. In one commandline make a copy of directory 'series3' and all its contents under name 'series3copy'.
   - cp -r series3/ series3copy

Part – IV : Installing

- Introduction - Package managers (npm, apt)
- ‘Sudo’
- Using ‘apt’
- Using ‘npm’

IV - Installing

Introduction - Package managers

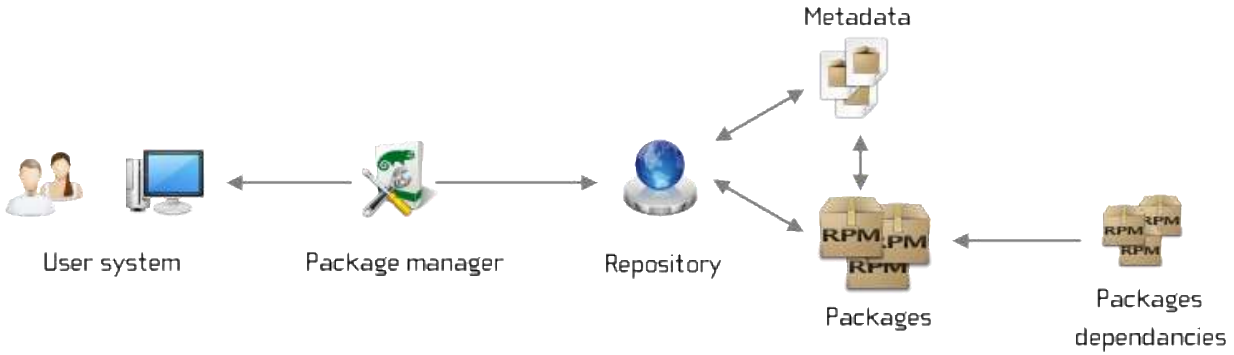
A **package manager** is a programming language's **tool** to create project environments and easily import **external dependencies.**

- A **Package** is simply an archive that contains binaries of software, configuration files, and information about dependencies.

- **Package Managers** essentially automate the process of installing, upgrading, configuring, and removing these packages (programs).

IV - Installing

Introduction - Package managers



Command Line Interface (CLI)

- Go 883K Packages
- Maven 138K Packages
- WordPress 56.6K Packages
- Cargo 16.1K Packages
- Atom 11.1K Packages
- Homebrew 4.7K Packages
- Carthage 2.93K Packages
- Elm 1.4K Packages
- Nimble 702 Packages
- Shards 31 Packages
- npm 763K Packages
- PyPI 135K Packages
- CocoaPods 44.3K Packages
- Meteor 13.4K Packages
- Hex 6.4K Packages
- Emacs 4.23K Packages
- Julia 2.27K Packages
- Racket 1.24K Packages
- Alcatraz 465 Packages
- Packagist 215K Packages
- NuGet 119K Packages
- CPAN 35.6K Packages
- CRAN 13.3K Packages
- Puppet 5.69K Packages
- SwiftPM 4.15K Packages
- Sublime 1.97K Packages
- Haxelib 1.19K Packages
- PureScript 237 Packages
- Rubygems 146K Packages
- Bower 68.2K Packages
- Clojars 17.7K Packages
- Hackage 12.7K Packages
- PlatformIO 5.14K Packages
- Pub 3.48K Packages
- Dub 1.45K Packages
- Jam 772 Packages
- Inqlude 217 Packages

- Installing
- updating
- upgrading
- removing

IV - Installing

Introduction - Package managers

OS

APT (**Advanced Package Tool**) is the package manager we're using with our Linux system - **Ubuntu**

- **SCOOP** (Simple Concurrent Object Oriented Programming)
- **Chocolatey**

Homebrew

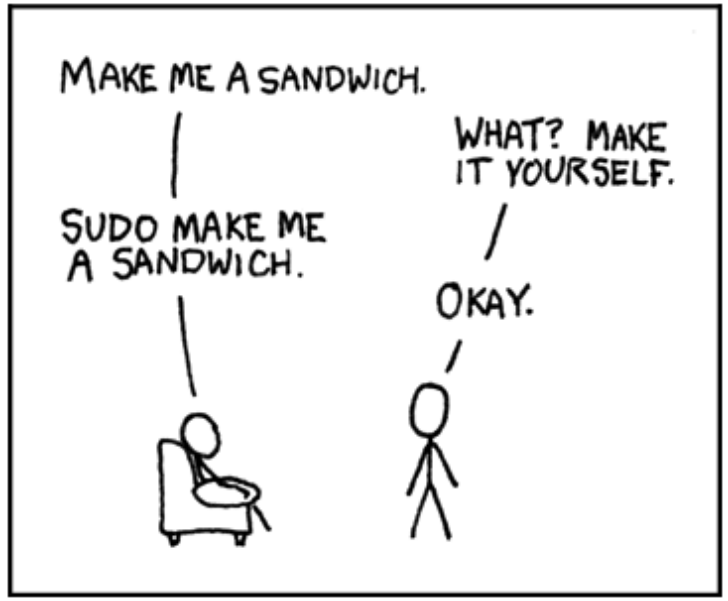
npm is the package manager for the Node JavaScript platform.

IV - Installing

sudo

sudo, the one command to rule them all. It stands for "super user do!"

"Access Denied?"



- The **Sudo** (**S**uper **U**ser **DO**) command in Linux is generally used as a prefix of some command that only superuser are allowed to run.
- If you prefix "**sudo**" with any command, it will run that command with elevated privileges or in other words allow a user with proper permissions to execute a command as another user, such as the superuser.
- This is basically the equivalent of "**run as administrator**" option in Windows.

IV - Installing

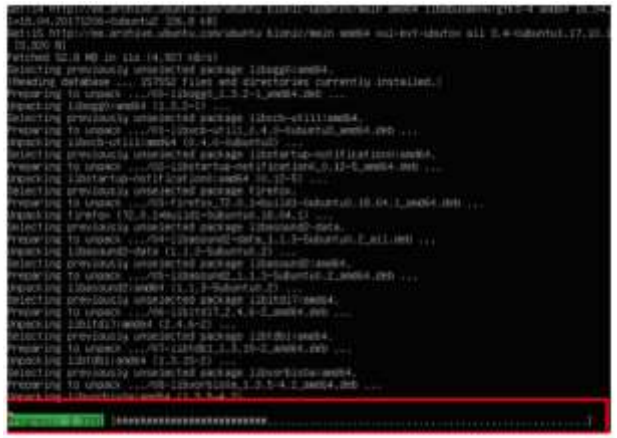
Using ‘apt’ (Advanced Package Tool)

sudo apt install <software name>

sudo apt install firefox

new

Has progress bar



Has limited functionality

sudo apt-get install <software name>

sudo apt-get install firefox

older

Has NO progress bar

Has more functionality

IV - Installing

Ubuntu

Using ‘apt’ (Advanced Package Tool)

apt	apt-get	Action
apt install <package>	apt-get install <package>	Install a package
apt remove <package>	apt-get remove <package>	Removes a package but keeps configuration (user settings) files
apt purge <package>	apt-get purge <package>	Removes a package with configuration files
apt update	apt-get update	Updates packages list
apt upgrade	apt-get upgrade	Upgrades software to their latest versions
apt full-upgrade	apt-get dist-upgrade	upgrades packages and removes old unnecessary ones
apt autoremove	apt-get autoremove	Removes unnecessary dependencies
apt search <keyword>	apt-cache search <keyword>	Searches packages based on keyword
apt show	apt-cache show	Shows information about a package like version, dependencies, etc,.
apt list	-	Lists packages based on certain criteria
apt edit-sources	-	Edits sources list

Ubuntu

Using ‘apt’ (Advanced Package Tool)

Checking for updates before installing/upgrading packages.

You should first run **update**, then **upgrade**. Neither of them automatically runs the other.

- apt-get update **updates** the list of available packages and their versions, but it **does not install** or **upgrade** any packages.

- apt-get upgrade actually installs newer versions of the packages you have. After updating the lists, the package manager knows about available **updates** for the software you have installed. This is why you first want to update.

IV - Installing

NodeJS & Npm Installation

Go to Terminal, and write : node - - version or node –v → **12.18.3**
npm - - version or npm -v → **6.14.6**

NODEJS + NPM

https://nodejs.org/en/download/

1. sudo apt –cache show nodejs
2. curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash –
   - sudo apt install curl
   - curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash –
3. sudo apt-get install -y nodejs
4. node –version
5. npm --version

VSCODE

https://code.visualstudio.com/

1. sudo apt update
2. sudo apt install nodejs
3. sudo apt install npm
4. nodejs -v
5. npm -v

IV - Installing

NodeJS & Npm

NodeJS

- It is a JavaScript runtime environment that executes JavaScript code **outside of a web browser.**
- It is an open source server environment and allows you to run JavaScript on the server.

Npm

- It is a package manager for the JavaScript programming language.
- It is the default package manager for the JavaScript runtime environment NodeJS

https://www.npmjs.com/

IV - Installing

Npm Commands

Install & Uninstall Packages to Dependency or DevDependency of a Project

Command	Explanation
npm install <package> npm i <package>	To install a package as a **Dependency** to the project. npm install --save <package> → for old versions of npm, no need more
npm install -D <package> npm i -D <package>	To install a package as a **DevDependency** to the project. npm install --save-dev <package> → for old versions of npm, no need more
npm uninstall <package> npm un <package>	To uninstall a package from **Dependency** of the project. npm uninstall --save <package> → for old versions of npm, no need more npm un --save <package> → for old versions of npm, no need more
npm uninstall -D <package> npm un -D <package>	To uninstall a package from **DevDependency** of the project. npm uninstall --save-dev <package> → for old versions of npm, no need more npm un --save <package> → for old versions of npm, no need more

IV - Installing

Npm Commands

Install & Uninstall Packages to the Computer GLOBALLY

Command	Explanation
sudo npm install -g <package> sudo npm i -g <package>	To install a package to computer **globally.** Exmp : sudo npm i -g markdownpreview
sudo npm -g uninstall <package> sudo npm -g un <package>	To uninstall a package from computer, which is **globally** saved. Exmp : sudo npm -g un markdown-preview

* It requires **SUDO** permission to install and uninstall a package to computer as globally

IV - Installing

Npm Commands

To install a package with a specific version

Command	Explanation
npm install <package>@version npm i <package>@version	To install a package with a **spesific version** Exmp : npm i lodash@4.16.0
npm install <package>@latest npm i <package>@latest	To install a package with the **latest version** Exmp : npm i lodash@latest

IV - Installing

Npm Commands

To list packages in a project

Command	Explanation
npm list	To list the packages

To see which packages need updating

Command	Explanation
npm outdated	To see which packages need updating

To update packages individually

Command	Explanation
npm update <package>	To update packages individually

IV - Installing

Npm Commands

To list globally saved packages in a computer

Command	Explanation
npm list -g --depth=0	To list the globally saved packages

To see which globally saved packages need updating

Command	Explanation
npm outdated -g --depth=0	To see which globally saved packages need updating

To update global packages individually

Command	Explanation
npm update -g <package>	To update global packages individually

Part – V : Versioning

- Introduction – Version Control Systems
- Initializing
- Basic Workflow
- Branching

V - Versioning

GIT - Version Control System

Why Version Control?

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

The purpose of version control is to:

- Record changes in a project (or, in other words, versions of a project), so that
- You can return to them later.

V - Versioning

GIT - Version Control System

- GIT is a distributed version control system that tracks changes to our project files over time.
- It enables us to record project changes and go back to a specific version of the tracked files, at any given point in time.
- This system can be used by many people to efficiently work together and collaborate on team projects, where each developer can have their own version of the project, distributed on their computer.
- Later on, these individual versions of the project can be merged and adapted into the main version of the project.
- Basically, it's a massively popular tool for coordinating parallel work and managing projects among individuals and teams.
- Git is one of the most important skills for any developer nowadays.

V - Versioning

GIT - Version Control System

What is Version Control?

https://git-scm.com/video/what-is-version-control

Download GIT

https://git-scm.com/downloads

git --version

V - Versioning

GIT - Version Control System

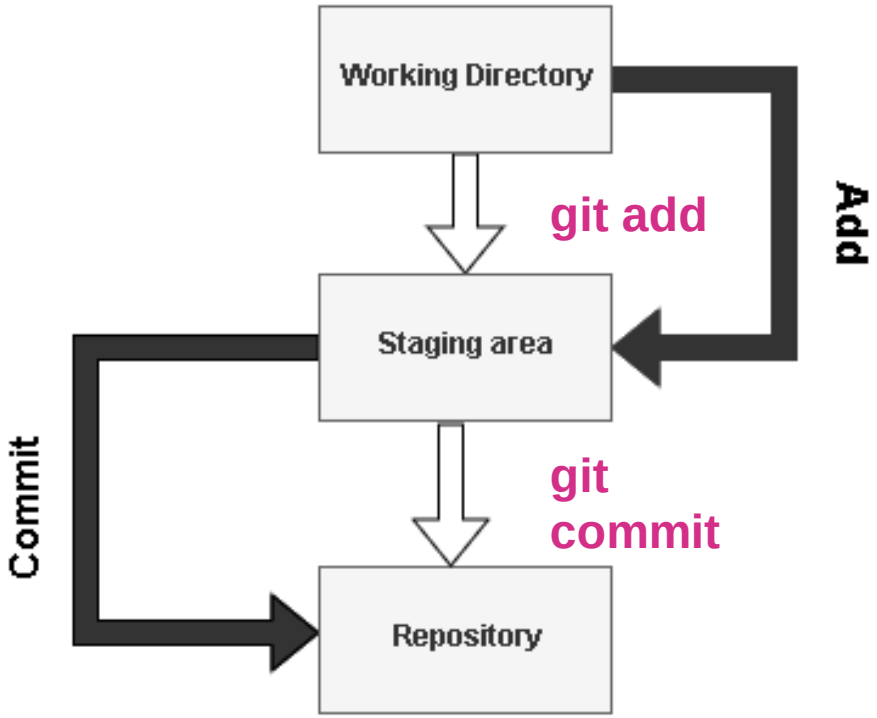
Configuring Your Name & Email

```
git config --global user.name "Your Name"
git config --global user.email "your@email.com"
```

V - Versioning

GIT - Workflow

The following areas are created when we run **“git init”** command



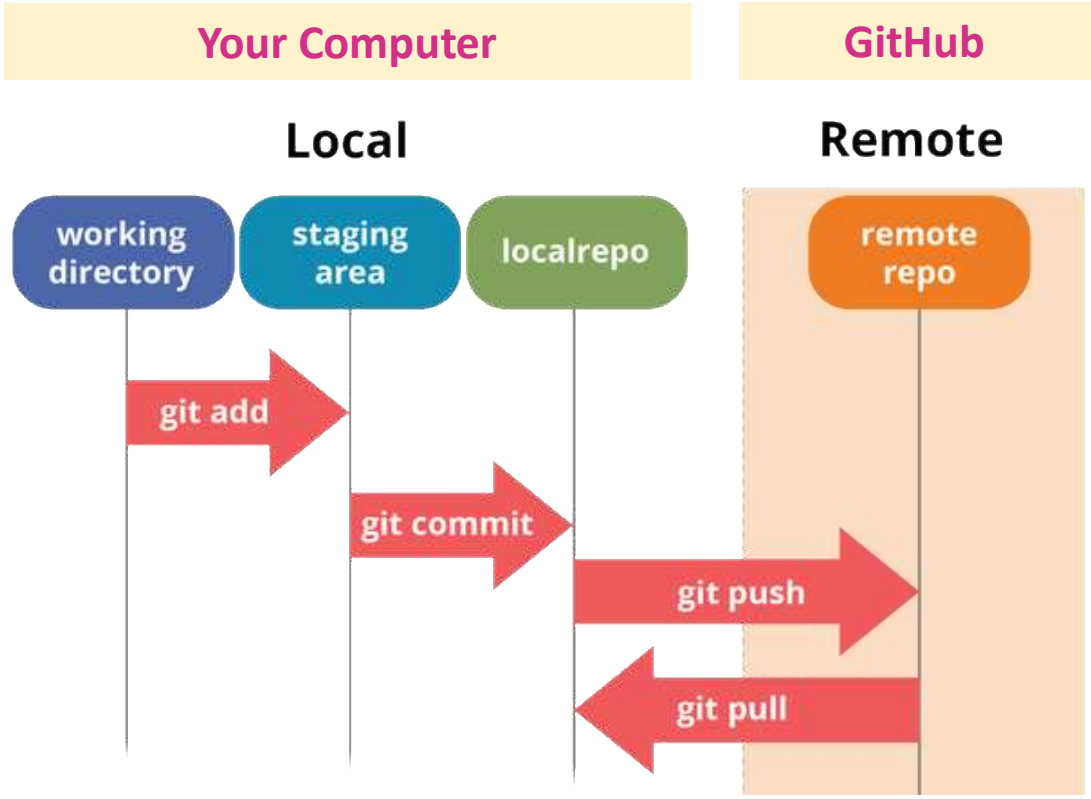
- It is the place where we keep all our files and folders of project
- It is the folder upon which you **initialize git**

- Changes made in the working directory can not be directly sent to the repository.
- Staging Area (Index) is used to add changes and then commit them to the repository.

- A Git repository is a container for a project that is tracked by Git.

V - Versioning

GIT - Workflow



Remote repository

- usually on a remote server.
- It's especially useful when working in teams
- This is the place where you can share your project code, see other people's code and integrate it into your local version of the project, and also push your changes to the remote repository.

Local repository

- an isolated repository stored on your own computer, where you can work on the local version of your project.

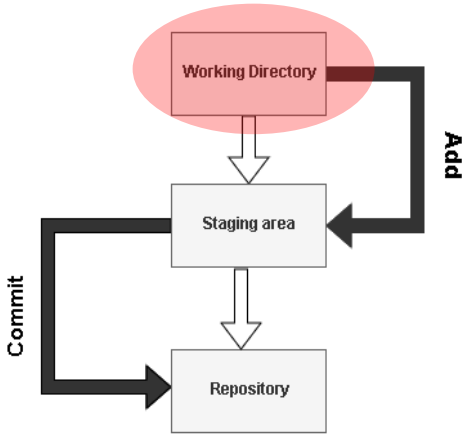
V - Versioning

GIT - Workflow

I – STEP : Initializing a repository

git init → To create a new repository and start tracking your project with Git (in your project folder)

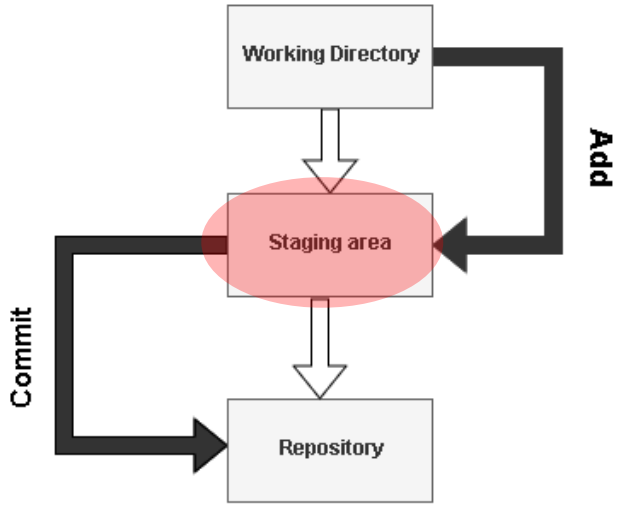
This command will generate a hidden **.git** folder for your project, where Git stores all internal tracking data for the current local repository.



file : untracked

V - Versioning

GIT - Workflow



II – STEP : Staging files

git add <filename>

(to add a specific file to the staging area)

git add <file1> <file2> <file3>

(to add multiple files to the staging area)

git add .

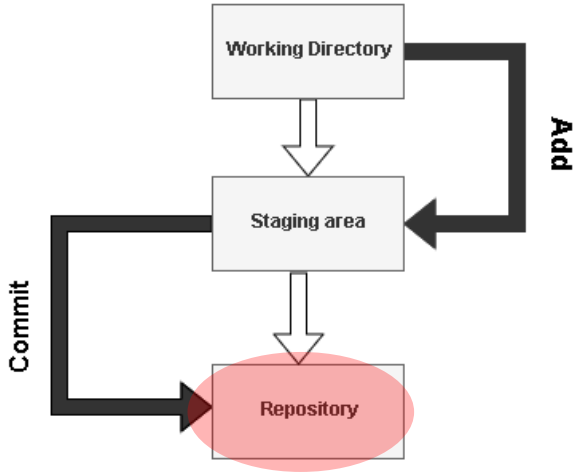
(to add all files to the staging area)

From the project folder, we can use the **git add** command to add our files to the staging area, which allows them to be tracked.

file : staged / tracked

V - Versioning

GIT - Workflow



III – STEP : Making commits

git commit -m “commit message”

(to commit the files from the staging area)

- **-m :** the commit's message.
- **-am** : commits modified files.

```
git commit -m "Subject" -m "Description..."
```

- A **commit** is a snapshot of our code at a particular time, which we are saving to the commit history of our repository.
- The **commit message** should be a descriptive summary of the changes that you are committing to the repository.
- **To create a new commit**, you will need to repeat the process of adding files to the staging area and then committing them after.

file : committed

V - Versioning

GIT - Workflow

Checking the status

git status

(to check the status of our repository)

- This is a command that is very often used when working with Git.
- It shows us which files have been changed, which files are tracked, etc.

V - Versioning

GIT - Workflow

Exercise - 1

1. Create a folder.
2. Initialize a git repo within that folder.
3. Create a readme file in the folder.
4. Make changes to the readme file.
5. Add the changes to the read me. Check the status of the repository. Then, leave a commit message and check the status once more.
6. Make more changes to the readme file. Use the shorthand command to add and commit changes.

V - Versioning

GIT - Workflow

Commit History

git log

(to see all the commits that were made for our project)

The logs will show details for each commit, like

- the author name,
- the generated hash for the commit,
- date and time of the commit,
- the commit message that we provided.

git log --oneline

(to see all the commits in short way)

git log --all --decorate --oneline --graph

V - Versioning

GIT - Workflow

To go back to a previous state of your project

git checkout <commit-hash>

Replace <commit-hash> with the actual hash for the specific commit that you want to visit, which is listed with the git log command.

To go back to the latest commit (the newest version)

git checkout master

V - Versioning

GIT - Workflow

Exercise - 2

1. Create a folder called ”learning_git”
2. Create a file in this folder called ”file1.txt”
3. Initialize an empty git repository
4. Add ”file1.txt” to the staging area
5. Commit with the message "adding file1.txt”
6. Check out your commit with ”git log”
7. Create another file called ”file2.txt”
8. Add file2.txt to the staging area
9. Commit with the message "adding file2.txt”
10. Remove the file1.txt file
11. Add this change to the staging area
12. Commit with the message "removing file1.txt”
13. Check out your commits using ”git log”

V - Versioning

GIT - Workflow

Ignoring Files

- To ignore files that you don't want to be tracked or added to the staging area, you can create a file called **.gitignore** in your main project folder.

- Inside of that file, you can list all the file and folder names that you definitely do not want to track

V - Versioning

GIT - Workflow

.gitignore – matching pattern

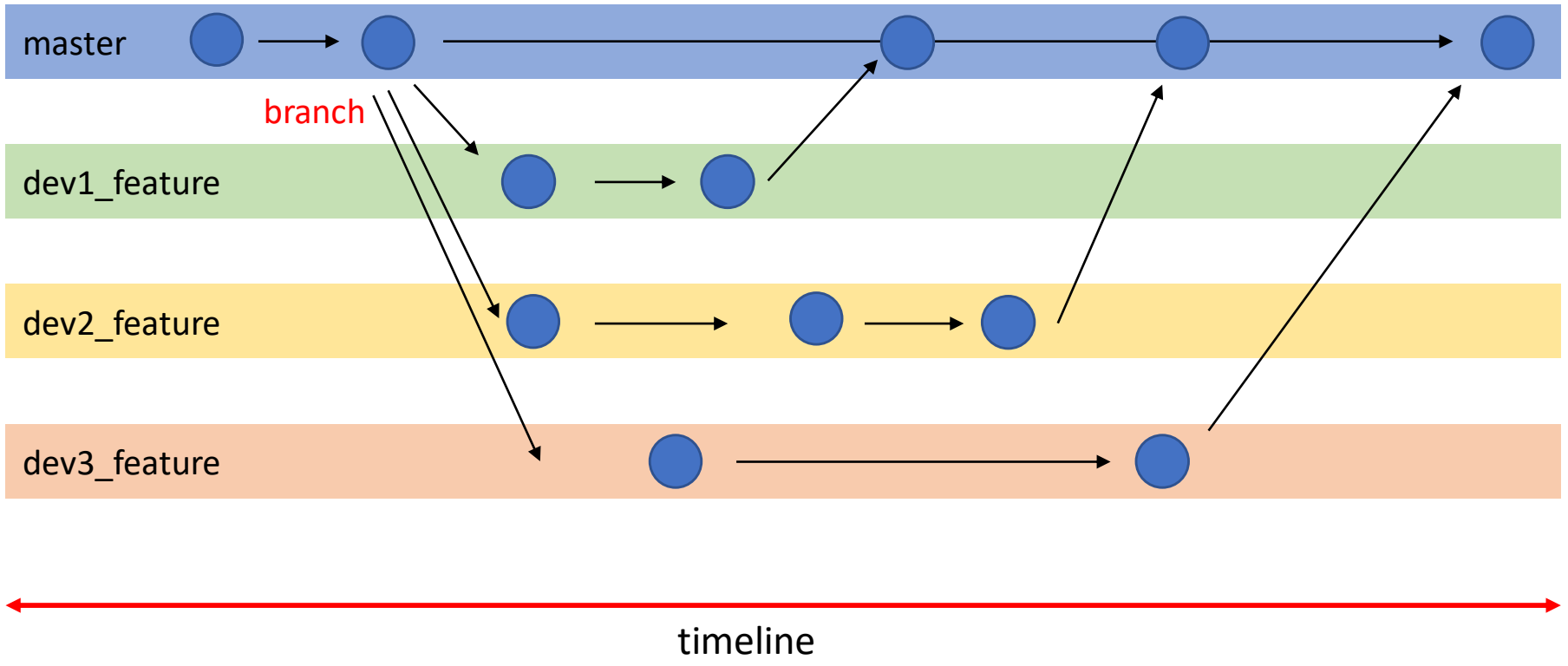
- `*` is used as a wildcard match
- `/` is used to ignore pathnames relative to the `.gitignore` file
- `#` is used to add comments to a `.gitignore` file

```
# Ignore Mac system files
.DS_store

# Ignore node_modules folder
node_modules

# Ignore all text files
*.txt

# Ignore files related to API keys
.env

# Ignore SASS config files
.sass-cache
```

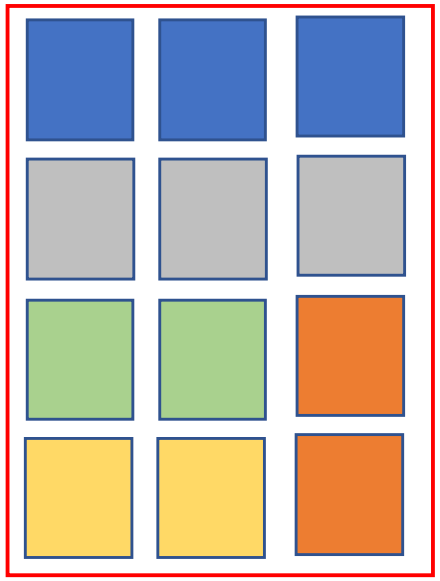
V - Versioning

GIT - Branching

Why need branching?

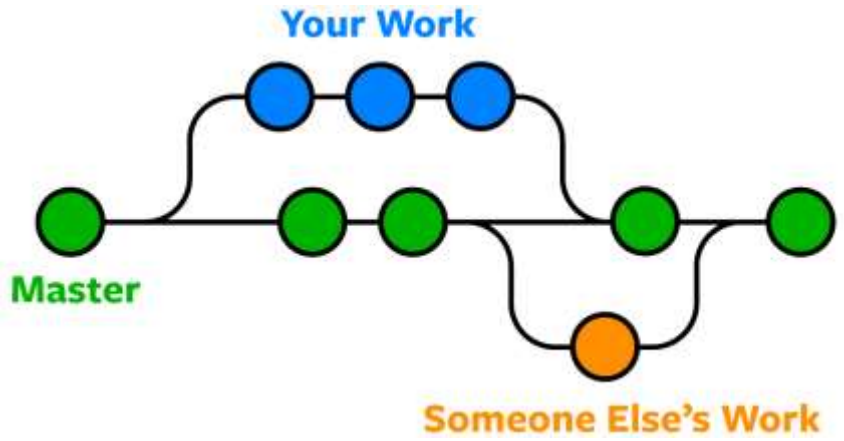


Your project



V - Versioning

GIT - Branching



- A **branch** could be interpreted as an individual timeline of our project commits.
- With Git, we can create many of these alternative environments (i.e. we can create different **branches**) so other versions of our project code can exist and be tracked in parallel.
- That allows us to add new features in separate branches, **without touching the '*official*' stable version of our project code (which is usually kept on the master branch).**

V - Versioning

GIT - Branching

Creating a new branch

git branch <branch name>

(to create a new branch)

- It's a good idea to create a **development** branch where you can work on improving your code, adding new experimental features, and similar.

- After development and testing these new features to make sure they don't have any bugs and that they can be used, you can **merge them to the master branch**.

V - Versioning

GIT - Branching

Changing branches

git checkout <branch name>

(to switch to a different branch)

- For example, you could be working on different features in your code and have a separate branch for each feature.
- When you switch to a branch, you can commit code changes which only affect that particular branch.
- Then, you can switch to another branch to work on a different feature, which won't be affected by the changes and commits made from the previous branch.

V - Versioning

GIT - Branching

Create a new branch and change to it

```
git checkout -b <branch name>
```

To list the local branches

```
git branch
```

-r : to list remote branches

-a : to list remote and local branches

To go back to master branch

```
git checkout master
```

V - Versioning

GIT - Branching

To rename the local branch

git branch -m <new_branch_name>

NOTE : If you want to rename the local branch and **you are in the same branch which you want to rename**

git branch -m <old_branch_name> <new_branch_name>

NOTE : If you are **in different branch** and want to rename the branch locally

V - Versioning

GIT - Branching

Merging branches

git merge <branch name>

(to merge branches)

- For example, after you fully implemented and tested a new feature in your code, you would want to merge those changes to the stable branch of your project (which is usually the default **master** branch).

NOTE : You would replace <branch-name> with the branch that you want to integrate into your current branch.

V - Versioning

GIT - Branching

Deleting a branch

git branch -d <branch name>

(to delete a branch)

NOTE : '**-d**' means deleting the branch *only if the branch is pushed and merged with the remote branch*.

NOTE : '**-D**' means deleting forcefully *without checking whether the branch is pushed or not*

- For example, after you fully implemented and tested a new feature in your code, you would want to merge those changes to the stable branch of your project (which is usually the default **master** branch).

NOTE : You would replace <branch-name> with the branch that you want to integrate into your current branch.

V - Versioning

GIT

Exercise - 3

1. Make sure you're on your desktop. Create a new folder called “project_git”.
2. Create a new repository locally “- git init”
3. Add a readme.md file to the master branch.

   Add a heading to file ⇒ ## heading. Make sure to add, commit your changes.
4. Create a branch named ”content” and switch to the new branch.
5. Add text to the readme file on the content branch. Don't forget to add and commit these changes.
6. Go back to master and create another branch named ”hotfix” from master. Note that the changes in your previous branch, content, are not there.
7. View all branches on terminal.

V - Versioning

GIT

Exercise - 3

1. Create a folder called ”branch_time”
2. cd into that folder.
3. Initialize an empty git repository.
4. Create a file called ”first.txt”, then add and commit the file.
5. Create a new branch called ”amazing_feature”
6. Create a file called ”best.txt” in ”amazing_feature” branch
7. Add the file.
8. Commit the file with the message -m "added best.txt".
9. Switch back to the master branch.
10. Merge your changes from the feature branch into master.
11. Delete the feature branch.

V - Versioning

GITHUB - Alternatives

ATLASSIAN Bitbucket

SOURCEFORGE

GitLab

APACHE ALLURA™

beanstalk

GitKraken

V - Versioning

GITHUB

git remote add origin <remote repository URL> → To connect remote repository to local repository

git remote -v → To check which remote repository was connected to local repository

git push origin master

V - Versioning

GITHUB

To delete the remote branch

`git push origin --delete <branch_name>` → to delete the **remote branch** from Terminal

`git push origin :<branch_name>`

To rename the remote branch by deleting

`git push origin :<old_branch_name> <new_branch_name>` → If you want to delete the remote branch and push the new branch from local mean

V - Versioning

GIT & GITHUB Case Study

1. To upload your project to GITHUB
2. To clone someone’s project to local repository
3. To fork someone’s project to remote repository for PR
4. (Advanced) Branching in local repository + git merge + CONFLICT solution
5. (Advanced) Branching in remote repository + git merge + git pull + git push CONFLICT solution
6. Helon’s FORK exercise

Part – VI : Publishing

- **How the internet work**

VI : Publishing

How the internet works?

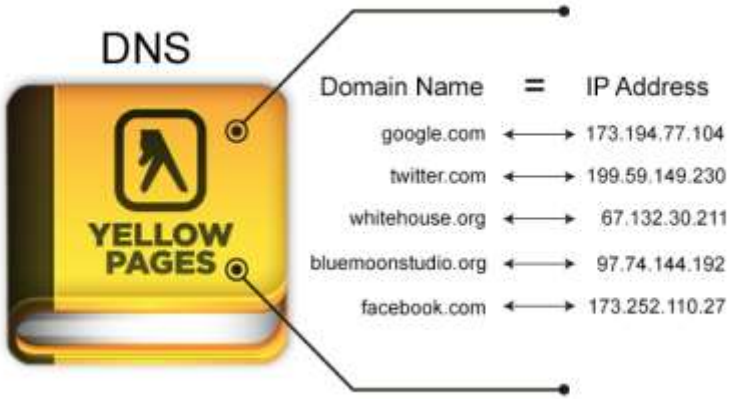
https://www.youtube.com/watch?v=7_LPdttKXPc&feature=youtu.be

VI : Publishing

How the internet works?

IP
(curl ifconfig.me)

ISP
Internet Service Provider



DNS
Domain Name Server

https://ipinfo.info/html/ip_checker.php

STEP - I www.google.com — ***request*** →

STEP - II ← ***response*** *IP of nearest Google Data Center*

STEP - III 173.194.77.104 — ***request*** →

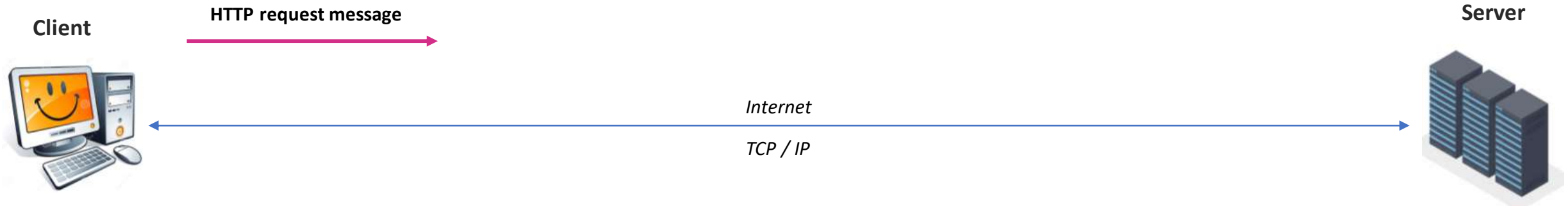
forward to Google →

response *from Google- the relevant files to load the google.com webpage - in smaller pieces called* ***data packets***

Google Data Center
Server

VI : Publishing

How the internet works?



TCP/IP

Transmission Control Protocol / Internet Protocol

These are communication protocols that define how data **should travel** across the internet.

DNS

Domain Name Servers

These are like an address book for websites.

HTTP

Hypertext Transfer Protocol

This is an application protocol that defines a **language** for clients and servers to speak to each other.

Component files

code files + assets

https://www.avast.com/c-what-is-tcp-ip

VI : Publishing

How the internet works?

SSH, HTTPS PROTOCOLS

Basic common Marks & Signs

Sign	Explanation
&	ampersand , and
‘	apostrophe , single quotation mark
*	asterisk
@	at
\	backslash (shift + alt + 7)
.	dot
:	colon
,	comma
{ }	left/rigt curly brackets , brace
$	dollar sign
“	double quotation mark
…	ellipsis (alt + .)
-	hyphen
–	dash
~	tilde , swung dash (alt + N)

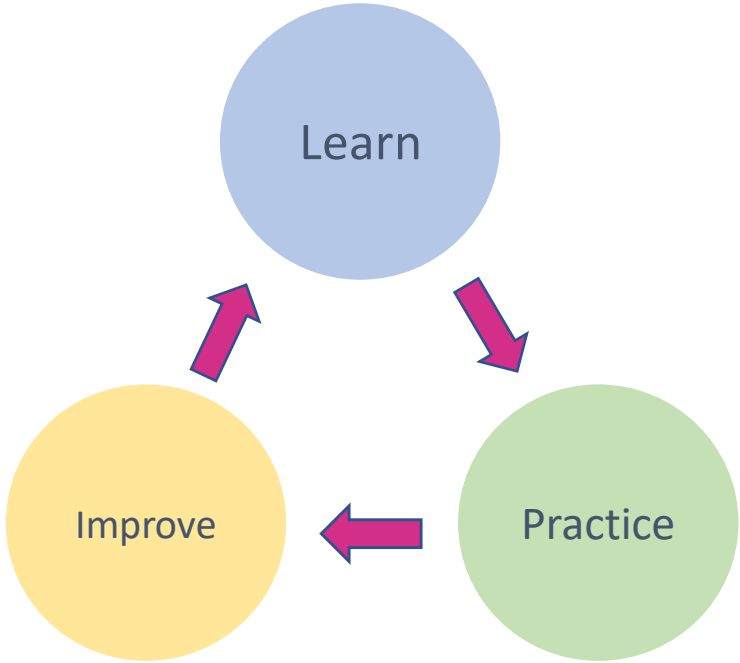
Sign	Explanation
=	equal
!	exclamation mark
``	Backticks , grave accent
>	greater than sign
<	less than sign
#	hash
( )	left/right parenthesis
%	percent
?	question mark
;	semicolon
/	slash
[ ]	left/right square brackets
_	underscore
|	pipe, vertical line (alt + 7)

The Roadmap

		DIFFICULTY	WAY TO LEARN
FE	HTML5	★☆☆☆☆	HTML Element tags
	CSS3	★★☆☆☆	CSS properties and values
	BOOTSTRAP4	★☆☆☆☆	Bootstrap class names
	JAVASCRIPT	★★★☆☆	understand the algorithm logic
	JQUERY	★★☆☆☆	requires good knowledge of Javascript
	REACT	★★★★☆	Javascript knowledge, concept and logic
BE	NODEJS	★★★★☆	Javascript knowledge and logic
	EXPRESS JS	★★★★☆	Javascript knowledge and logic
	MONGO DB	★★★★☆	understand the concept and logic

Introduction

- Who am I?
- My motivation?
- How to manage this course?
  - Theory + Practice + Repitation + Effort
  - What ? Why ? Where?
- Your aim?
- Algorithm logic? (Video)
- Theory + Practice (to improve your horizon)
  - Youtube channels
  - Udemy/Corsera/Udacity courses
- You are problem solver, not problem finder!
- Flow of my lessons (articles, videos, web pages, homeworks)
- Some more advice



code .

- Launch VS Code.
- Open the **Command Palette** (⇧⌘P) and type 'shell command' to find the **Shell Command: Install 'code' command in PATH** command.

- Restart the terminal for the new `$PATH` value to take effect. You'll be able to type 'code .' in any folder to start editing files in that folder.