**Professional Software Development**

**Bash Fundamentals**

**Introduction to Bash**

A black and white text on a black background

Description automatically generated



A black background with white text

Description automatically generated

A black and white text on a black background

Description automatically generated

**Fundamental Commands**

pwd : print working directory. This commands allows us to view the directory we are currently inside of. We can also see that location on the right of the green and purple. In this case, we get a tilde.

A screenshot of a computer

Description automatically generated

The tilde indicates that we are inside of the user’s home directory. Every single user on an operating system has a home folder or home directory.

After typing pwd, we can see we are inside lisac directory.

Cd : Used to change the directory we are in. we can go inside of the parent’s directory or change directories all together.

Cd .. : Used to go to the parent’s directory. We need to put a space between cd and .. otherwise it won’t work.

Cd ../.. : used to go to two levels up in the parent’s directory. We only need to put a space between cd and the first dots.

Cd (name of the directory) : if we want to change the directory and specify which directory we want to be in.

Clear : clears the screen, the terminal

Ls : this will give us a list of all the directories and files that are inside of the current directory that we are in.

One letter + TAB : that allows us to do autocomplete. If we type T then tab, we’ll get all the files that start with the letter T

The up arrow key : will cycle through all of the previous commands we just typed in the terminal. We can do the opposite with the down arrow key

Mkdir + name of the file : To create a directory (ex : mkdir test)

Touch + name of the file : To create a file (ex : touch my\_file.txt)

Mv + the file we want to move + the destination location (ex: mv my\_files.txt files)

Cp : copy a file. We specify the file that we want to copy then the location where we want it copied (ex : cp my\_files.txt files)



If we want to copy the file but want to specified that it’s the copied file, we add it in the path, like shown above.

To copy a directory, we need to copy the directory itself but everything it contains as well. To do so, we need to include the hyphen r (written like this -r ), because it will tell the computer to look through every single subdirectory and subfile and copy those as well.

We’ll type cp -r

The hyphen is known as a flag or an option

Rm + name of the file : remove, delete a file. We also need to include the -r. so we’ll type

Rm -r new\_files/ (new-files being the name of the file)

**Bash Editors**

Nano : most friendly editor inside of BASH.

We type in nano then the name of the file that we want to edit.

A black screen with text

Description automatically generated

Here we wanted to create a file named test.js. we write this down and the screen below is what’s gonna appear and where we can write our javascript code.



Once we’re done typing our code we need to save it (CTRL S) and if we want to exit we type in CTRL X.

If we want to edit a file, we type nano then the name of the file.

To copy paste something into the bash terminal, we’ll copy whatever we need to copy (from outside of it, CTRL C), then once inside Bash we just right click

VIM :

Mouse not needed for that editor. To start using that editor, we type in vim inside bash. By default it doesn’t let us type anything. Why . Vim has two modes (a command mode and an insertion mode). To be in the insertion mode, we need to type i

Once we’re done typing we need to hit the ESCAPE button. That will allows us to hit CTRL S to save our file and to exit Vim.

To exit Vim without keeping the changes we made we type in :q !

Q stands for quit. If we don’t type in the !, it won’t let us exit the editor.

To save a file :wq

Head test.js : will print eh first 10 lines from the fil test.js

Tail test.js : will print the last 10 lines from the file test.js

**IDEs**

**VSCode Editor**

IDE is a code editor and stands for integrate development environment. Typically it features a debugger, built in code editor, IntelliSense, find and replace etc.

(VS code doesn’t have all the features than an IDE would have so technically it’s not an IDE)

VS Code Features

CTRL + shift + p : to open the command palette

To search for a file, we can use the search bar at the top.

To search through the features, we can use the magnifying glass on the top left . we have three options for that search.



The first on the left is match case Aa. This will show the results that match the capital/regular letters in the word we’re looking up. The second option is to match the spacing. The third one is use regular expression. Underneath we have the replace option where we replace the word we just searched by another one. Ex: replace square by cube and the search will show us all the cube words.

That search engine is useful if we’re doing a global search because we don’t remember in while file we need to look into.

To look inside a specific file, we open the file and then hit CTRL + F

CTRL + H : stands for find and replace and will open the replace window

CTRL + / CTRL - : to zoom in or out

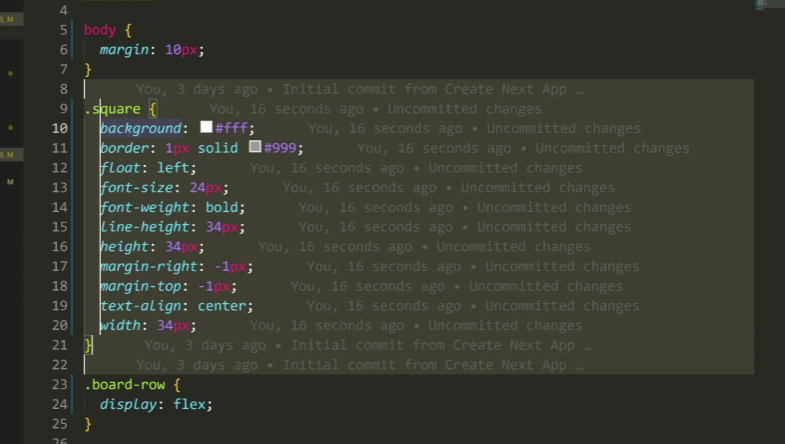
A screenshot of a computer program

Description automatically generated

If we are in a file and we want to split the screen to view another one, we right click on the file we are in (in this case global.css) then click on split right

To close the sidebar, CTRL + B

To create multiple terminals : we press the + sign on the right side of the terminal



To write on several lines at the same time, we hit CTRL + ALT and drag the mouse along all the lines we want to modify. Once we’re done typing, we hit ESCAPE to go back to the single curseur

If we want to find where a function is defined, we can right click on the name of the function and hit go to definition.

**VSCode Customization**

Open settings (or hit CTRL + , ). The use settings is what’s gonna apply globally to every single VS code windows we open. The workspace is gonna be stored inside the current project we’re working on.

**VSCode Debugger**

A debugger is an advanced tool that allows us to pause the execution of our code and step through that code viewing the values of different variables, function calls etc.

To start the debugger, we need to open up our React app.

So we’re inside of a directory that contains our next js / React app and we’re inside of our Javascript file.

Once we are inside the Javascript fie, we hit F or A screenshot of a computer

Description automatically generated

And then press run and debug. Once we do that, the search bar will show us a couple options

A screenshot of a computer

Description automatically generated

In this case, we’ll click on the chrome option and this will allow the computer to do a launch with chrome and tell us where to use the debugger.

Once we’re running the debugger, we can use a breakpoint. The breakpoint allows us to specify where in our program we want to stop and start stepping through and viewing the different code/. We can have as many breakpoints as we want.

A computer screen shot of a code

Description automatically generated

**Git & GitHub**

**Introduction to Git and GitHub**

They allow us to keep track of the changes that have been made to our code. They also allow us to go back to different steps in time so that we can revert changes or move things that might be breaking.

Let’s say we have a developer that’s creating a file or repository.

The developer will need to create a commit. Every once in a while, during the creation of that file, there’s gonna be some checkpoints, some moments in time where the code is frozen and stored somewhere. It will be stored inside a commit.

The first commit will generally be named first commit. The developer will use Git to denote that he has made the first commit that’s kinda initialized the GitHub repository.

If the developer wants to make a few changes to the code, he’ll make another commit after the changes have been made in his code.

Summary : we have a repository, which is all of the code we’re working on. Then inside of this repository, we have a few commits that we create along the way. The original one, the first version of our code, then the other commits with the changes made to the code after the first commit. The other commit focus on the changes made to the code.

A blackboard with white text

Description automatically generated

Thanks to those commits, we now have a log of all the previous changes that we made since creating the initial code. If the code was working fine in a previous version but now has an issue, we can find the issue more easily by focusing on the last commit we made. Or we can delete that last commit all together and revert to a previous version of our code. Github and Git allow us to track the different versions of our code and revert to another one if needed.

Git is a tool that works for a single computer, one single developer.

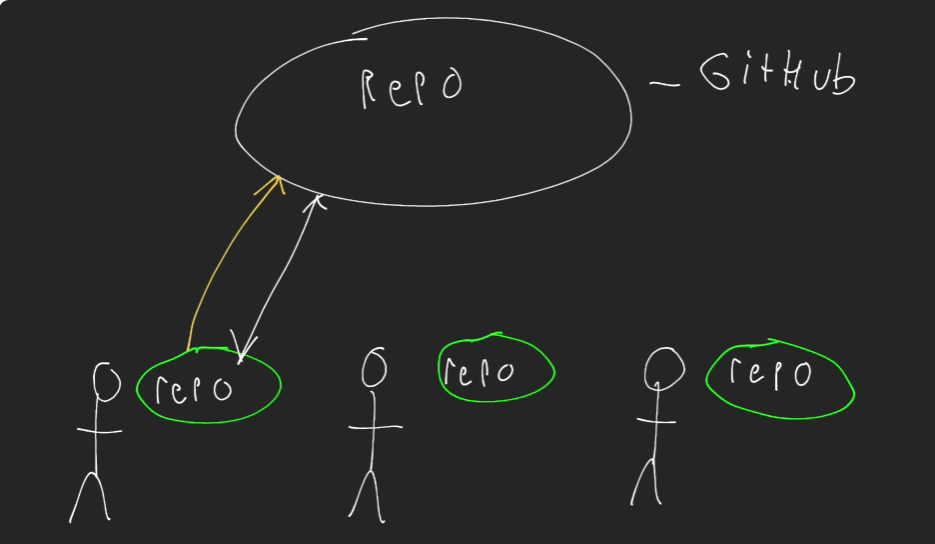
What happens when we’re working with a team of developers ?

All of the developers will work on the same code and will use Git on their individual computer. They all have a repository on their local computer. If they’re working on the same code, at one point in time, they will need to synchronize the changes that they each made to the code. To avoid having to send the code back and forth to one another, they need a centralized location where all of the main code is going to live. That’s where we need a remote repository and where GitHub comes in.

A group of stick figures with green ovals

Description automatically generated

GitHub provides a bunch of features for developers to be able to synchronize their code



If developer A is making changes to the code on his local computer, the common github repository will get those changes as well since they’re connected.

**Fundamental Git Commands**

How to set up a Git repository :

A screenshot of a computer program

Description automatically generated

When we are inside of Vs code, we’re gonna type git init inside the terminal. It will initialize a git repository. It’s gonna create a .git folder. This folder is invisible by default. That means we have an underlying .git folder that is hidden from us and that contains all of the information about the current git repository. That’s what’s tracking all of the commits and all of the changes that have been made.

The staging area :

If we type in git status, it will give us some basic information about the current git directory.

A screenshot of a computer program

Description automatically generated

Before we make a commit, we need to tell Git what files we want included inside the commit. To do that, we add them to the staging area. We type in git add then the name of the directory that we want to include in the staging area. In this case it’s vscode then we enter the request.

A black background with white text

Description automatically generated

After that, we type in git status

There is a short cut for the git add request. If we don’t want to manually type in all the files present in the directory, we can type **git add .**  and by default all the files present in the current directory will be included.



How to make a commit :

We type in git commit -m (m stands for message) then we will provide a message that will describe what we just did. In this case, we’re just going a simple repository. Typically we need to be as descriptive as possible while staying brief. In this case we’ll type



Result :

A screen shot of a computer program

Description automatically generated

Master is the branch that the commit is on then the message that we wrote. Then how many files have been changes and we created the 3 files

To see a log of the commits : Git log

A computer screen with white text

Description automatically generated

The long series of number is the ID of the commit

When we’re working with Git, we have branches. Branches are different versions of our code base, that have different commits on them. Master is the master branch or the main branch. The main branch is where all the code that’s fully tested, production ready , is going to live. When a developer will take a look at the final code, he will refer to that main branch. The HEAD is indicating to us where we are and here we are inside the master branch.

The Git ignore file :

If we don’t want to include some files inside a git repository, we can create a git ignore file

A screenshot of a computer

Description automatically generated

In this file we can include all of the paths of other files or directories that we don’t want to include in our git repository. If we had a directory called env and a file inside called secret.env

A screenshot of a computer

Description automatically generated

Env stands for environment variable file. These typically contain passwords, private keys, logging credentials and we might need them for our application but don’t want to include them in our Github because we don’t want to push them out to all the other developers.

We go to the gitingore file and type in /env

A screenshot of a computer

Description automatically generated

If we type in /env the env file is turning grey. Once that is done, we can make a commit that’s gonna ignore the env file

We type in git status and the terminal will highlight that gitignore was not included in the repository

A screenshot of a computer program

Description automatically generated

We then type git add . this will only include the getignore file

We then type git commit -m “add gitignore file”

A screen shot of a computer

Description automatically generated