1.

INTRO TO VERSION CONTROL

GIT & GITHUB SERIES

AGENDA

01

VERSION CONTROL

02

GIT

03

GITHUB

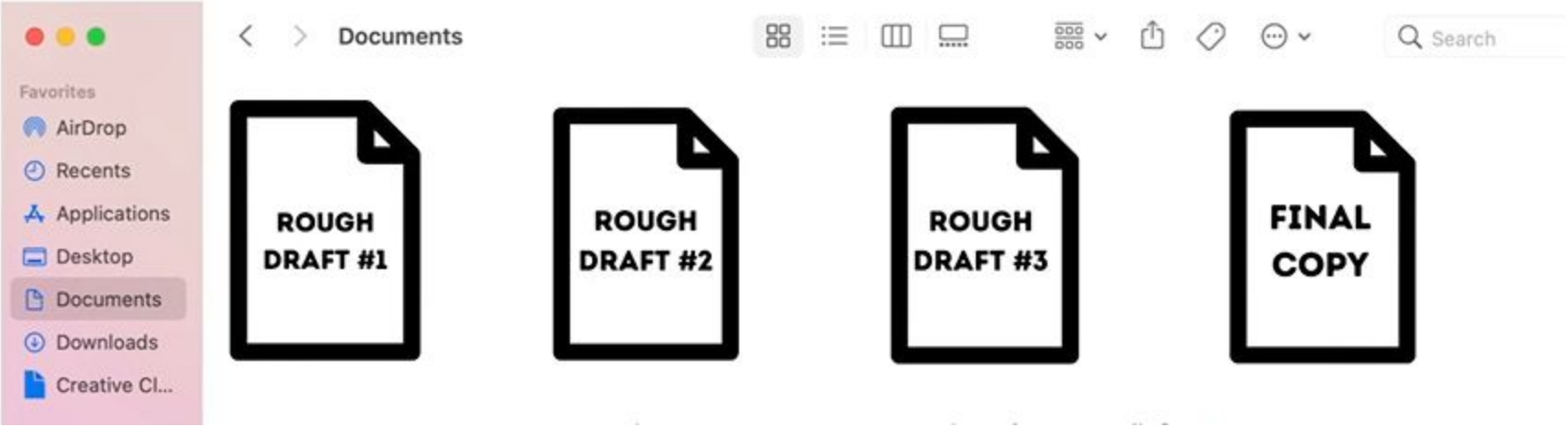
04

DISTRIBUTED COMPUTING

VERSION CONTROL

METHOD OF TRACKING AND MANAGING CHANGES TO FILES (OR CODE) ACROSS TIME

PRACTICAL EXAMPLE



PRACTICAL EXAMPLE

THIS IS WHERE GIT COMES INTO PLAY

WHAT IS GIT?

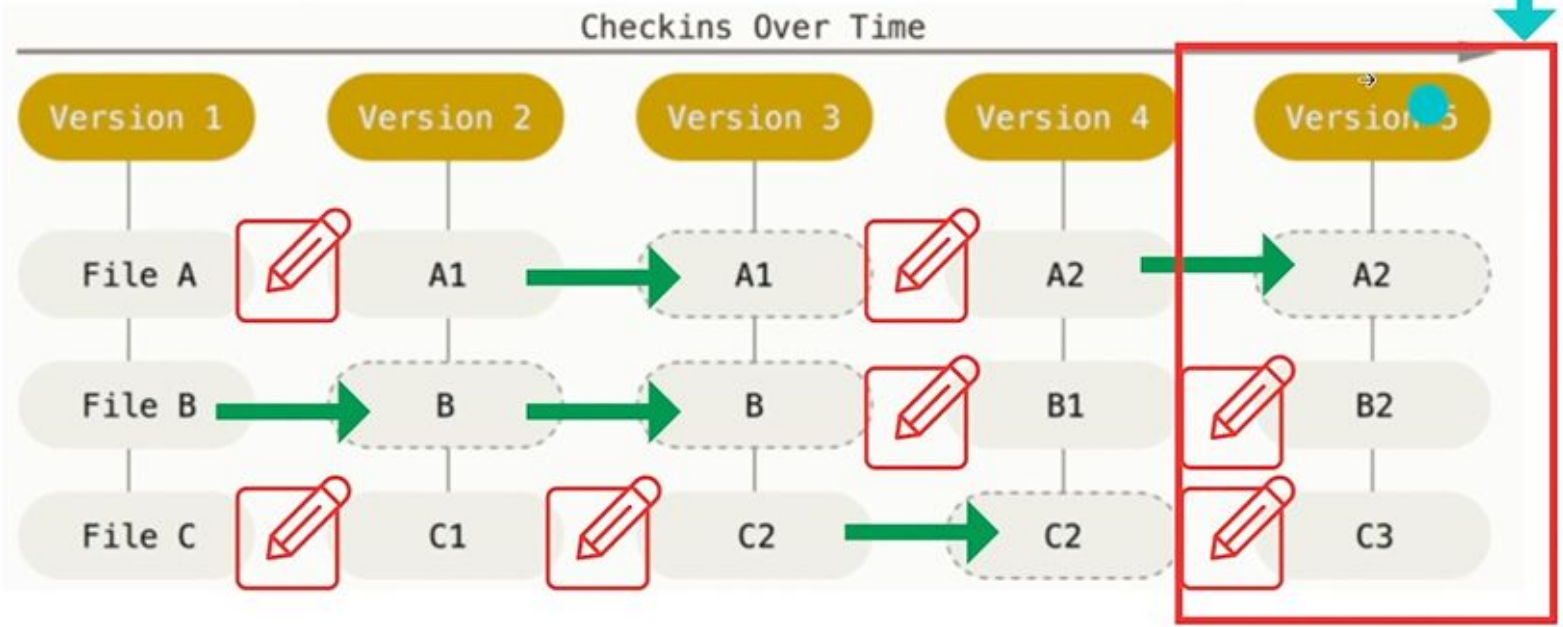
DOCUMENTATION: HTTPS://GIT-SCM.COM/

GIT

- OPEN SOURCE DISTRIBUTED VERSION CONTROL SYSTEM
- "CODING LANGUAGE" YOU USE ON YOUR LOCAL COMPUTER VIA THE TERMINAL
- TRACKS "SNAPSHOTS" OF YOUR FILE IN TIME (LOCALLY) AND SAVES THESE SNAPSHOTS INSIDE REPOSITORIES (ANY FOLDER CAN BE A GIT REPO!)
- SINCE IT IS LOCAL, ONLY YOU HAVE ACCESS TO YOUR GIT REPOS UNLESS YOU CONNECT IT TO A REMOTE PROVIDER
- IS NOT CONNECTED TO THE INTERNET

GIT

ALWAYS WORKING ON MOST RECENT VERSION BUT YOU CAN GO BACK IF NEEDED!



“

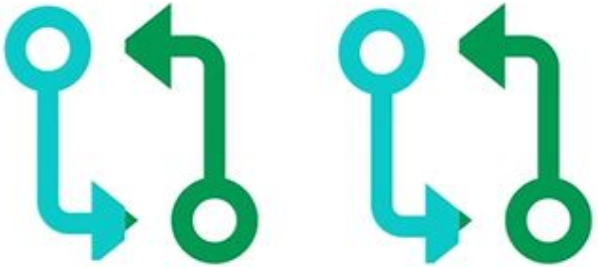
SO WHERE DOES GITHUB COME INTO PLAY?

SHARE FILES,
COLLABORATE,
PORTFOLIO PROJECTS

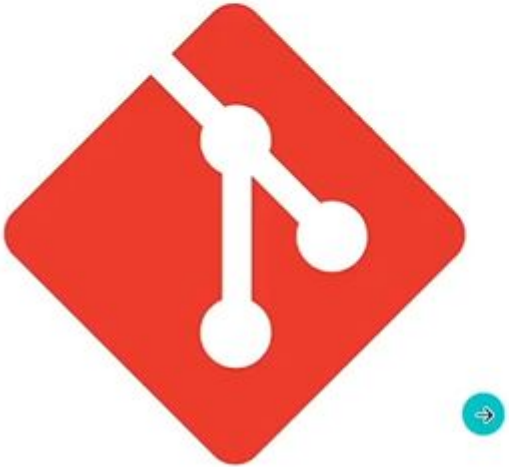
GITHUB

PROJECT 1

- REMOTE HOSTING SERVICE FOR VERSION CONTROL USING GIT
- IT'S REMOTE SO THAT MEANS ON THE INTERNET
- IF IT IS ON THE INTERNET THAT MEANS YOU CAN COLLABORATE/SHARE FILES (REPOS)
  - PUSH UP FILES/CODE (CHANGES TO FILES)
  - PULL DOWN FILES/CODE
- CONNECT YOUR LOCAL GIT REPOS TO A REMOTE REPO
- SOCIAL NETWORK FOR SOFTWARE DEVELOPERS AND CODERS
- CAN'T BE USED WITHOUT GIT!!!

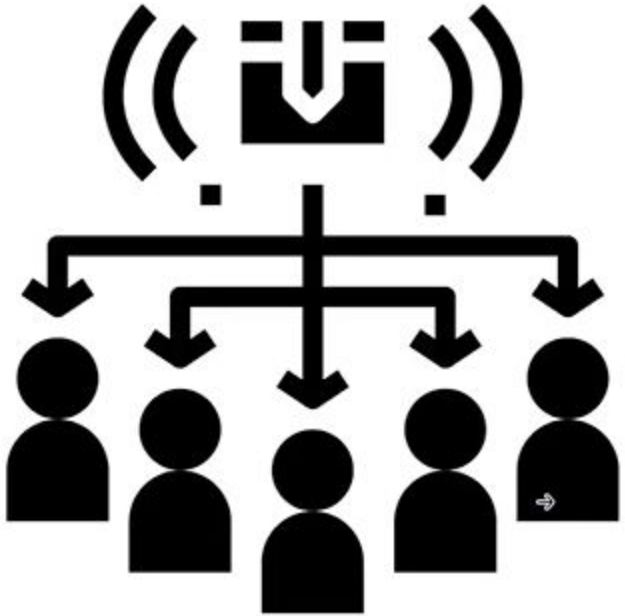


“WHATS THE DIFFERENCE BETWEEN THE TWO?



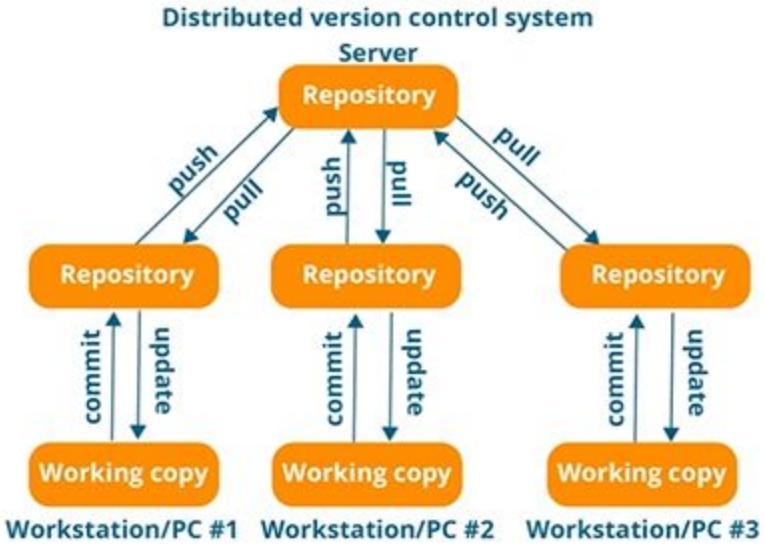
GIT	vs.	GITHUB
• Aides in Version Control		• Aides in Version Control
• Downloaded on local computer		• On the Internet
• Can be used without GitHub		• Can't be used without Git
• Great for independent projects		• Great for Collaboration
• Branching Model		• Branching Model
• Not a Social Network Feel		• Social Network Feel
• Local		• Remote

DISTRIBUTED COMPUTING



DISTRIBUTED VERSION CONTROL SYSTEM (DVCS)

- COPY OF THE COMPLETE REPOSITORY DISTRIBUTED TO EVERYONE ON THE TEAMS COMPUTER
- ALLOWS MEMBERS TO MAKE CHANGES AND WORK ON THEIR CODE LOCALLY THEN MERGE THOSE CHANGES IN REMOTELY
- MONITOR WHAT GETS MERGED IN AND CODE QUALITY



SOURCE: EUDREKA

2.

GIT & GITHUB SERIES

CREATING A LOCAL GIT REPO

AGENDA

01

INSTALLING GIT/SIGN UP FOR GITHUB

02

INSTALLING CODE EDITOR

03

GIT CONFIG

04

LOCAL REPO

INSTALLING GIT

- BUILT IN FOR MOST MAC COMPUTERS

- FOR PC, GO HERE AND FOLLOW THE PROMPTS: HTTPS://GIT-SCM.COM/DOWNLOAD/WIN

- TO CHECK WHAT VERSION OF GIT YOU HAVE, OPEN UP A TERMINAL AND TYPE:
  - GIT --VERSION (MAC)
  - $ SUDO DNF INSTALL GIT-ALL (LINUX)

git-scm.com/downloads

git --local-branching-on-the-cheap

Search entire site...

About

Documentation

Downloads

GUI Clients

Logos

Community

The entire Pro Git book written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

Downloads

macOS

Windows

Linux/Unix

Older releases are available and the Git source repository is on GitHub.

GUI Clients

Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience.

View GUI Clients →

Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

View Logos →

Git via Git

If you already have Git installed, you can get the latest development version via Git itself:

```
git clone https://github.com/git/git
```

You can also always browse the current contents of the git repository using the web interface.

INSTALLING GIT

- BUILT IN FOR MOST MAC COMPUTERS
- FOR PC, GO HERE AND FOLLOW THE PROMPTS: HTTPS://GIT-SCM.COM/DOWNLOAD/WIN
- TO CHECK WHAT VERSION OF GIT YOU HAVE, OPEN UP A TERMINAL AND TYPE:
  - GIT --VERSION (MAC)
  - $ SUDO DNF INSTALL GIT-ALL (LINUX)

SIGN-UP FOR GITHUB

- GITHUB RELIES ON GIT!
- GITHUB IS JUST A REMOTE PLACE TO PUSH UP YOUR REPOS OR PULL DOWN REPOS TO COLLABORATE (AND TO STORE YOUR PROJECTS)
- SIGNUP AT: WWW.GITHUB.COM
- NEED TO ALSO CREATE A PERSONAL ACCESS TOKEN USED FOR YOUR "PASSWORD" WHEN ASKED FOR ONE IN THE TERMINAL

GETTING A PERSONAL ACCESS TOKEN

1. CLICK YOUR FACE (OR ICON)
2. CLICK SETTINGS
3. SCROLL DOWN TO DEVELOPER SETTINGS (CLICK IT)
4. CLICK PERSONAL ACCESS TOKENS (CLASSIC)
5. GENERATE NEW TOKEN (CLASSIC)
6. ADD NOTE
7. SET EXPIRATION
8. CHECK REPO BOX
9. GENERATE TOKEN
10. COPY AND PASTE TOKEN IN SAFE SPACE

THE CODE EDITOR

INSTALLING VISUAL STUDIO CODE (VS CODE)

- VS CODE IS A CODE EDITOR THAT CAN BE USED AS A GUI FOR YOUR GITHUB REPOS
- HELPS YOU VISUALLY SEE AND TRACK CHANGES THAT YOU MAKE
- NOT REQUIRED BUT HIGHLY RECOMMENDED!
- DOWNLOAD HERE: HTTPS://CODE.VISUALSTUDIO.COM/DOWNLOAD

CONFIGURING GIT

- EVERY GIT REPO WILL HAVE A FOLDER THAT HOUSES CONFIGURATION SETTINGS

- YOU MUST CONFIGURE YOUR NAME AND EMAIL (SAME EMAIL USED FOR GITHUB) WHEN FIRST USING GIT . OPEN UP YOUR TERMINAL AND TYPE THE FOLLOWING:

```
william@williams MINGW64 ~ (main)
$ git config --global user.name "William Okomba"
(base)
william@williams MINGW64 ~ (main)
$ git config --global user.name
William Okomba
(base)
william@williams MINGW64 ~ (main)
$ git config --global user.email willokomba@gmail.com
```

TERMINAL COMMANDS

SOME OF MANY COMMANDS
WHEN WE SAY TERMINAL WE MEAN

- ZSH SHELL
- BASH
- POWERSHELL
- ETC.

COMMAND	ACTION
PWD	PRINT YOUR DIRECTORY (WHERE YOU ARE)
CD	CHANGE YOUR DIRECTORY
MKDIR	MAKE NEW DIRECTORY
LS	LIST FILES IN THAT DIRECTORY
TAB	HITTING TAB COMPLETES YOUR TYPING
CTRL + L	CLEAR YOUR TERMINAL


MAKE A
LOCAL REPO

MAKE A LOCAL REPO

1. Change your directory to your 'Documents' folder
2. Create a new folder in your documents called 'first_repo'
3. Change your directory so that you are inside the 'first_repo' folder
4. Change that folder to a git repo by typing `**git init**`

```
(base)
william@Williams MINGW64 ~ (main)
$ pwd
/c/Users/DELL
(base)
william@Williams MINGW64 ~ (main)
$ cd Documents/
(base)
william@Williams MINGW64 ~/Documents (main)
$ mkdir first_repo
(base)
william@Williams MINGW64 ~/Documents (main)
$ cd firt_repo
bash: cd: firt_repo: No such file or directory
(base)
william@Williams MINGW64 ~/Documents (main)
$ cd first_repo
(base)
william@Williams MINGW64 ~/Documents/first_repo (main)
$ git init
Initialized empty Git repository in C:/Users/DELL/Documents/first_repo/.git/
(base)
william@Williams MINGW64 ~/Documents/first_repo (main)
```

“SO WHAT IS GIT INIT?

GIT INIT "INITIALIZES A GIT REPO" AKA TURNS THE FOLDER INTO A GIT REPO THAT WE CAN USE FOR VERSION CONTROL!

CREATE A FILE INSIDE LOCAL REPO

Make sure you are in the 'first_repo' directory

1. Create a README.md file using the touch command
2. Type git status to check to see if the file was created (this is a command you should use A LOT to check the status of your repository)

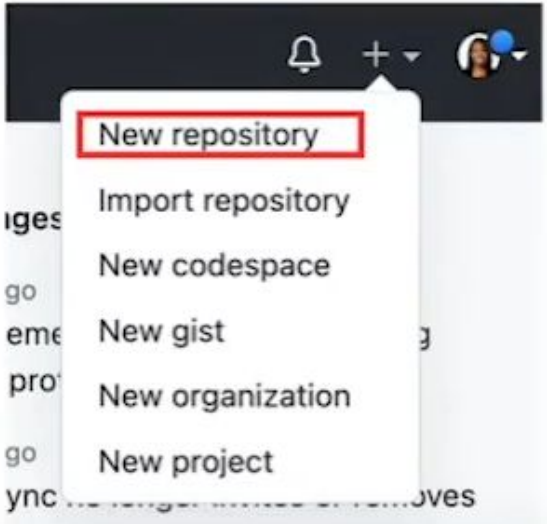
```
William@Williams MINGW64 ~/Documents/first_repo (main)
$ touch README.md
(base)
William@Williams MINGW64 ~/Documents/first_repo (main)
$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md

nothing added to commit but untracked files present (use "git add" to track)
(base)
```

GIT & GITHUB SERIES

CONNECTING A LOCAL REPO TO GITHUB

CREATE A NEW REPO ON GITHUB

- LOG INTO YOUR GITHUB ACCOUNT
- HIT THE '+' SIGN NEXT TO YOUR FACE AND CLICK ON "NEW REPOSITORY"
- GIVE THE REPO A NAME -- SOMETIMES I MAKE IT THE SAME NAME AS MY LOCAL VERSION
- MAKE IT PUBLIC (IF YOU WANT ANYONE TO SEE IT)
- CLICK 'CREATE REPOSITORY'



CONNECT OUR 'FIRST_REPO' TO GITHUB

- FOLLOW THE SECOND BOX OF INSTRUCTIONS IN YOUR TERMINAL (EXCEPT FOR LAST LINE)!
- BE SURE YOU ARE IN YOUR LOCAL REPO (AKA THE DIRECTORY IS THE REPO YOU CREATED)
- IF THERE IS AN ERROR, READ WHAT GIT IS TRYING TO SUGGEST FOR YOU TO DO

...or push an existing repository from the command line

```
git remote add origin https://github.com/aspratle/first_repo.git
git branch -M main
```

```
        README.md

nothing added to commit but untracked files present (use "git add" to track)
(base)
William@Williams MINGW64 ~/Documents/first_repo (main)
$ git remote add origin git@github.com:williamokomba/my_first_project.git
(base)
William@Williams MINGW64 ~/Documents/first_repo (main)
$ git branch -M main
(base)
William@Williams MINGW64 ~/Documents/first_repo (main)
$ git add README.md
(base)
```

```
illiam@Williams MINGW64 ~/Documents/first_repo (main)
 git push -u origin main
it@github.com: Permission denied (publickey).
atal: Could not read from remote repository.

lease make sure you have the correct access rights
nd the repository exists.
```

Note: in case it doesn't go through and show the error above, then add public key: [adding ssh public key tutorial](). Once done do git push again.

GIT COMMANDS

Common Git Commands

- $git config
- $git init
- $git clone <path>
- $git add <file_name>
- $git commit
- $git status
- $git remote
- $git checkout <branch_name>
- $git branch
- $git push
- $git pull
- $git merge <branch_name>
- $git diff
- $git reset
- $git revert
- $git tag
- $git log

SOURCE: DEV COMMUNITY

STEPS FOR STARTING LOCAL THEN REMOTE

1. CREATE A FOLDER ON YOUR COMPUTER FOR THE PROJECT
2. RUN A `GIT INIT` TO CHANGE IT TO A GIT REPO
3. USE THE `TOUCH` COMMAND TO MAKE A FILE
4. CREATE A REPO ON GITHUB
5. FOLLOW THE SECOND BOX OF CODE TO CONNECT THE REMOTE REPO TO YOUR LOCAL ONE
6. ANYTIME YOU CHANGE A FILE:
   A. GIT ADD FILENAME
   B. GIT COMMIT -M "MESSAGE"
   C. GIT PUSH

PUSHING CHANGES TO GITHUB REPO

GIT & GITHUB SERIES

AGENDA

01

VS CODE INTERFACE

02

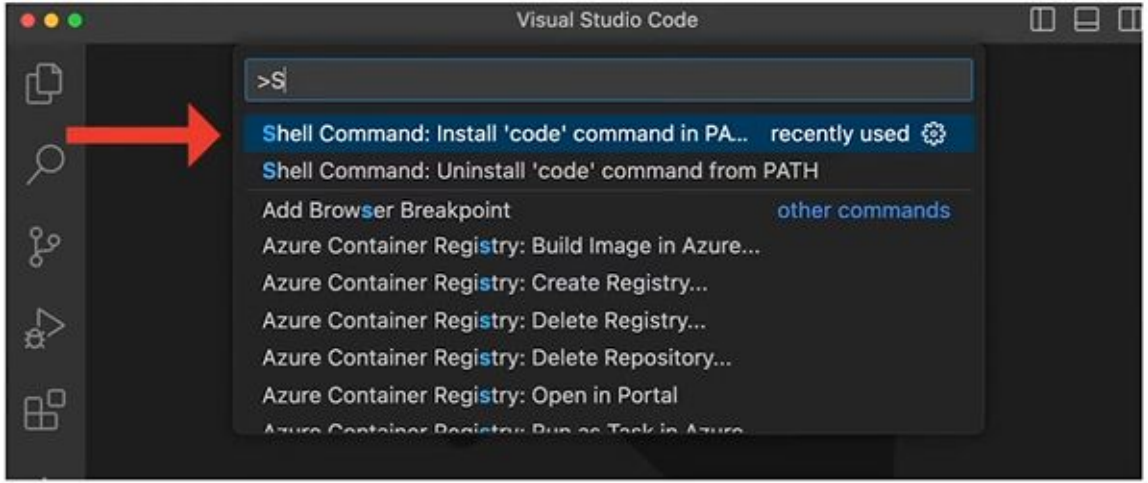
CHANGE A FILE

03

PUSH CHANGES

CONNECT VSCODE TO TERMINAL

- OPEN UP VS CODE
- PRESS 'COMMAND + UP ARROW + P'
- IN THE SEARCH BAR TYPE
  - > SHELL COMMAND
- CLICK THE "INSTALL 'CODE' COMMAND IN PATH
- <u>THIS ALLOWS US TO USE 'KEYWORD' CODE IN OUR TERMINAL TO OPEN UP VS CODE</u>



IF IT DOES NOT WORK: TRY 'UNINSTALLING' THE SHELL COMMAND THEN INSTALLING IT (CLICK THE SECOND CHOICE ABOVE THEN THE FIRST)

CHANGE A FILE

RED COLOR IS WHAT YOU TYPE IN THE TERMINAL!

1. NAVIGATE TO YOUR LOCAL REPO 'FIRST_REPO'
2. OPEN UP README IN VSCODE (CODE README.MD)
3. CHANGE THE FILE
4. SAVE CHANGES (COMMAND+S OR CTRL + S)
5. ADD THE CHANGED FILE TO STAGING (GIT ADD)
6. WRITE A COMMIT MESSAGE ABOUT THE CHANGE (GIT COMMIT -M "MESSAGE")
7. GIT PUSH TO PUSH CHANGES UP

AGENDA

01

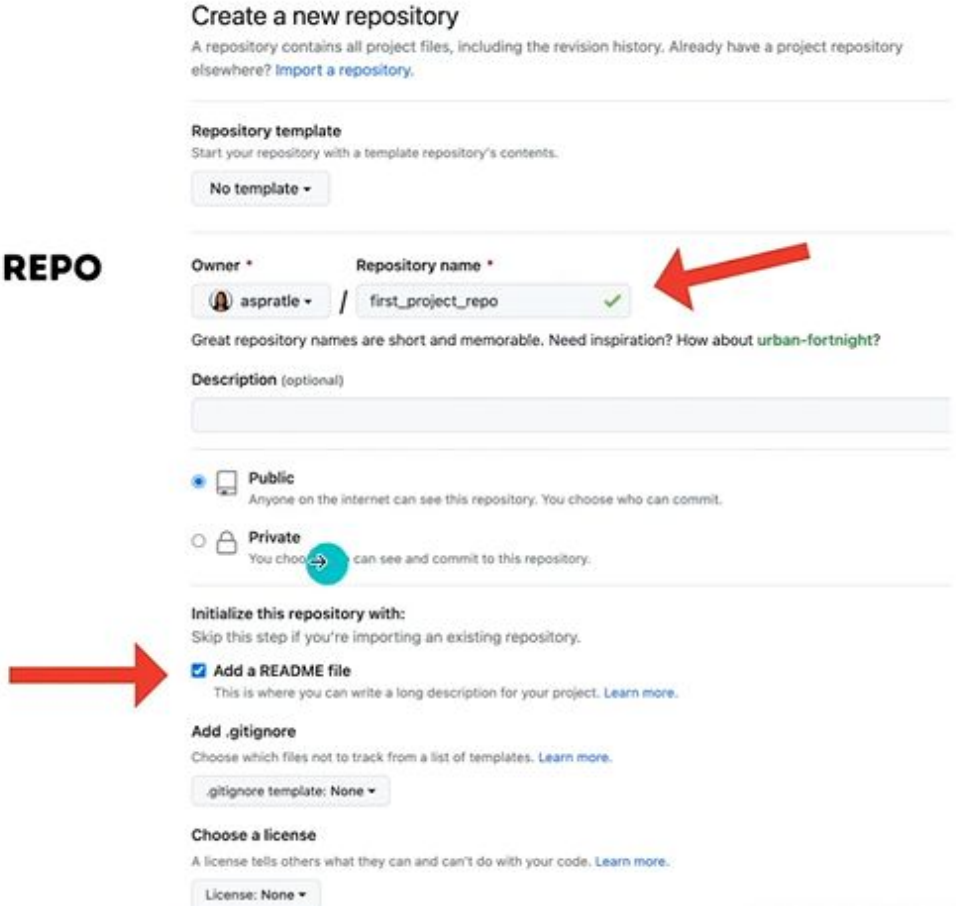
CREATE A REPO IN GITHUB

02

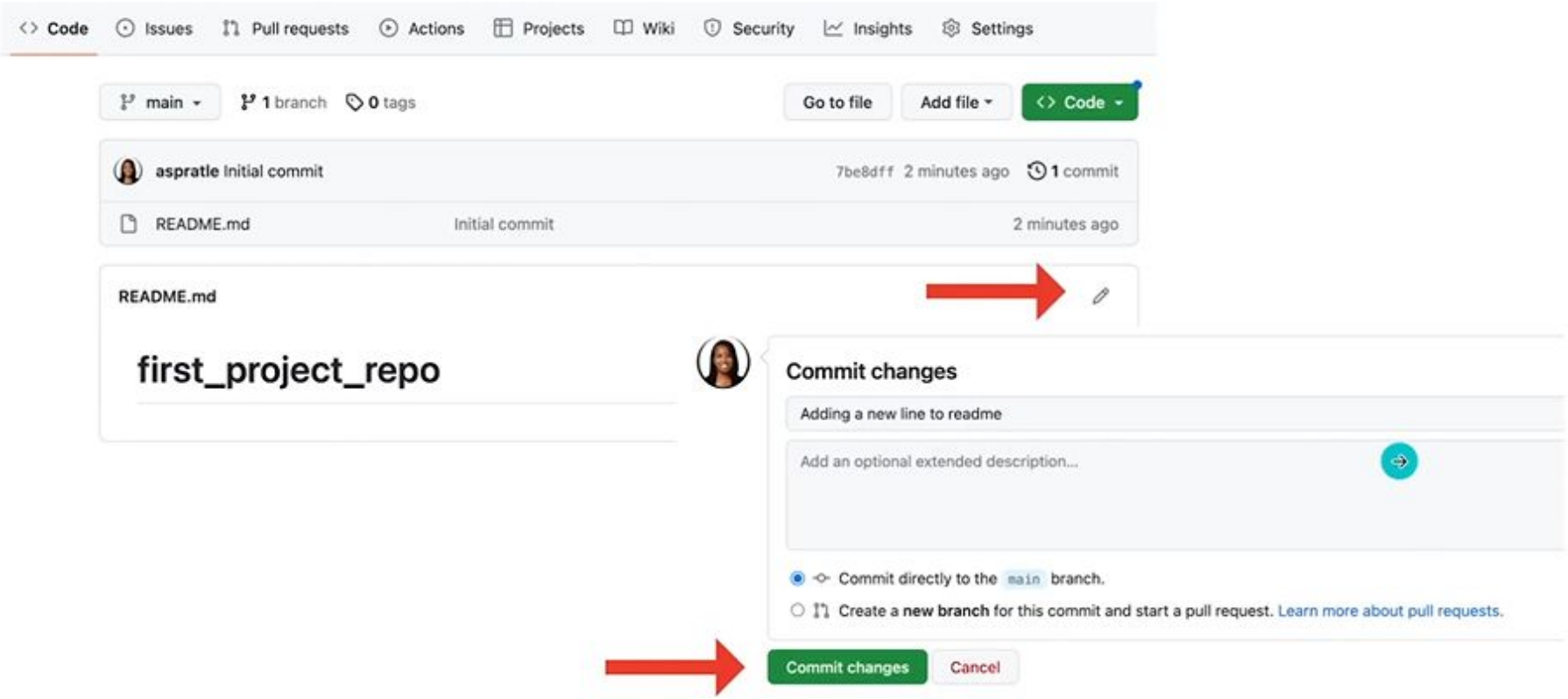
MAKE CHANGES RIGHT IN GITHUB

CREATE A GITHUB REPO

- HIT THE '+' SIGN NEXT TO YOUR FACE
- ADD AN OWNER (YOUR ACCOUNT) AND REPO NAME
- CLICK PUBLIC
- CHECK 'ADD A README FILE'
- "CREATE REPOSITORY"

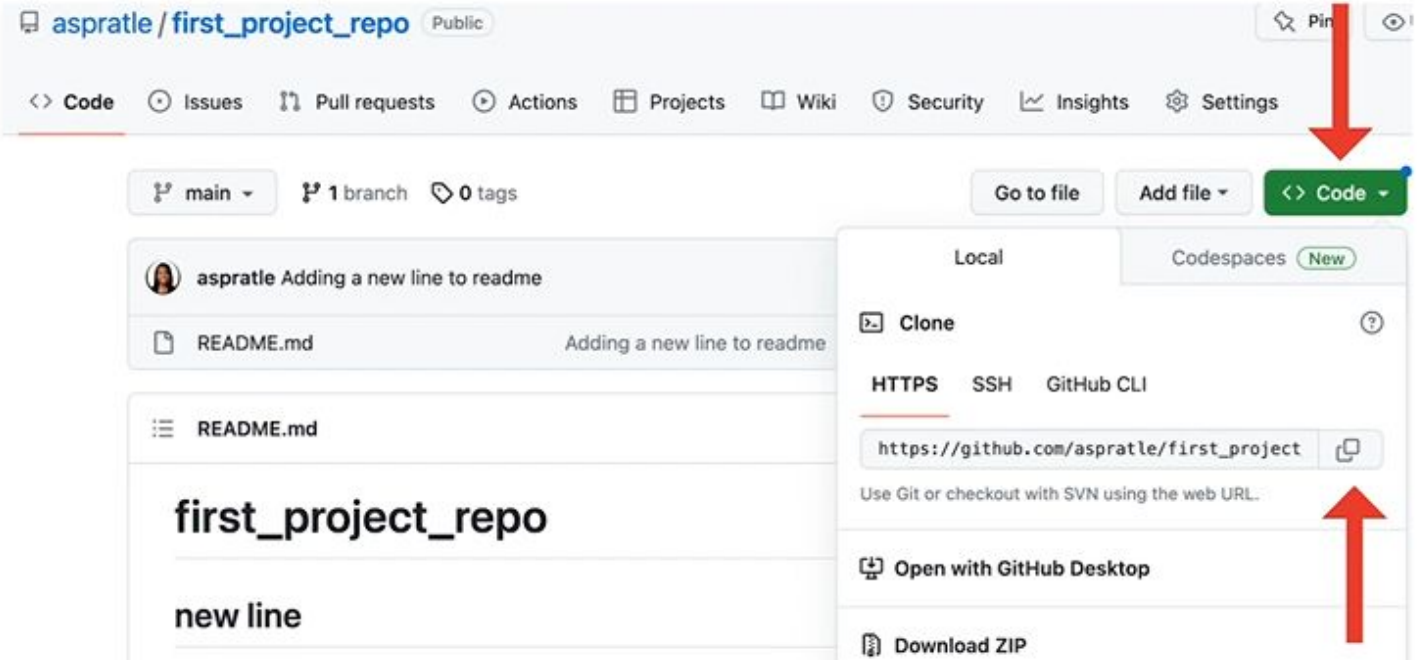


EDIT THE README RIGHT WITHIN GITHUB



CLONE THE REPO DOWN TO YOUR LOCAL COMPUTER

- **COPY THE URL OF THE REPO**



Then do the following

```
git clone “url link”

#move to the directory

cd “directory name”

#then create a new file

touch “file name”

#then add
git add

#then commit
git commit -m “mesage”

#then push
git push
```

FORKING AND CLONING GITHUB REPOS

GIT & GITHUB SERIES

FORKING A REPO

- FORKING A REPO IS GOOD WHEN YOU WANT TO ADD A COPY OF SOMEONE ELSE'S REPO TO YOUR ACCOUNT (FOR YOUR OWN PROJECT OR USE)

- TRY FORKING A REPO (ANY REPO ON GITHUB)

GIT & GITHUB SERIES

USING GIT FETCH AND GIT MERGE

GIT FETCH/MERGE

- SOMETIMES YOU WANT TO MERGE THE ORIGINAL OWNER'S CONTENT INTO YOUR FORK

- THIS IS SO YOU HAVE THE MOST RECENT CHANGES ON YOUR REPO THAT THE OWNER HAS MADE TO THE ORIGINAL REPO

- FIRST: SET THE ORIGINAL OWNERS REPO URL AS THE UPSTREAM (GIT REMOTE ADD UPSTREAM URL)
- GIT FETCH UPSTREAM
- GIT MERGE UPSTREAM/MAIN -M "YOUR MESSAGE"

GIT & GITHUB SERIES

ADDING A .GITIGNORE FILE

.GITIGNORE

- A .GITIGNORE FILE IS USED WHEN YOU DON'T WANT GIT TO TRACK CERTAIN FILES

- IT IS COMMON TO ADD CODING NOTEBOOK 'CHECKPOINTS' SOMETIMES SEEN AS ".IPYNB_CHECKPOINTS" TO THE .GITIGNORE FILE IF CREATING JUPYTER NOTEBOOKS

- IT IS IMPORTANT TO ADD THIS FILE IN THE BEGINNING (IF ADDED AFTER YOU PUSH A CODE NOTEBOOK, IT WON'T TRACK FUTURE CHECKPOINTS BUT WILL STILL KEEP THE OLD ONES)

.GITIGNORE

- NAVIGATE INSIDE YOUR REPO
- IN TERMINAL TYPE THE FOLLOWING:
  - TOUCH .GITIGNORE (DON'T FORGET THE '.')
  - CODE (OPENS IN VSCODE)
    - EDIT THE FILE BY COPYING AND PASTING:
    - .IPYNB_CHECKPOINTS
    - */.IPYNB_CHECKPOINTS/*
  - GIT ADD .
  - GIT COMMIT -M "COMMIT MESSAGE"
  - GIT PUSH

.gitignore (Untracked)

.gitignore

```
1   1  .ipynb_checkpoints
2   2  */.ipynb_checkpoints/*
```

REVERTING BACK TO A DIFFERENT VERSION

GIT & GITHUB SERIES

AGENDA

01

GIT LOG

02

GOING BACK TO
AN OLD VERSION

GIT LOG

- GIT LOG WILL PRINT OUT A LOG OF ALL COMMITS INCLUDING THE COMMIT "ID"
- YOU CAN TYPE GIT LOG --ONELINE TO PRINT OFF A MORE READABLE SUMMARY OF EACH COMMIT
- TO EXIT OUT OF THE LOG PRESS 'Q'

- To remove any commit

git revert “provide commit id from the git log”

CREATING BRANCHES IN GIT/GITHUB

GIT & GITHUB SERIES

AGENDA

01

DEFINE BRANCHES

02

CREATE A BRANCH

BRANCHES

- BRANCHES ARE USED WHEN COLLABORATION IS INVOLVED (NOT NECESSARY FOR INDIVIDUAL PROJECTS)
- MOST OF THE TIME, BRANCHES ARE FOR A SPECIFIC SPRINT OR FEATURE
- BRANCHES CAN BE MERGED INTO THE MAIN BRANCH VIA PULL REQUESTS
- ALWAYS MAKE SURE YOU ARE PUSHING TO THE RIGHT BRANCH USING GIT BRANCH!

CREATE A BRANCH

- USE GIT BRANCH NAMEOFBRANCH TO CREATE A NEW BRANCH
- USE GIT CHECKOUT NAMEOFBRANCH TO SWITCH TO THAT BRANCH
- ONCE ON THAT BRANCH YOU CAN CREATE YOUR OWN FILES AND PUSH THEM TO THAT BRANCH (BRANCH WILL NOT SHOW UNTIL YOU ADD A FILE TO IT)

RESOLVING MERGE CONFLICTS IN GIT/GITHUB

GIT & GITHUB SERIES

AGENDA

01

MERGE CONFLICTS

MERGE CONFLICTS

- DUE TO THE COLLABORATIVE NATURE OF GIT, THERE COULD BE MULTIPLE PEOPLE CHANGING THE SAME THINGS AT THE SAME TIME AND CAUSE CONFLICT
- GIT TRIES TO DO AN AUTO-MERGE BY DEFAULT
- IF IT CAN'T AUTO-MERGE, IT WILL RAISE A CONFLICT
- YOU THEN HAVE TO GO IN AND DECIDE WHAT CHANGES TO ACCEPT
- MERGE CONFLICTS CAN BE VERY COMPLEX


```
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)
```

END