



# Introduction to git and github





# git




# GitHub



# Table of content

## Introduction

1. Git vs GitHub
  2. Why do we need Git or Github?
  3. GitHub essentials
  4. How to use git?
  5. Link local and online repository (Push, clone)
- 

A decorative graphic on the left side of the slide consists of a large cyan hexagon in the center. Surrounding it are several smaller hexagons of varying shades of blue and cyan. Some of these hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. There is also a network-like icon with a central node and several smaller nodes connected by lines.

# Introduction

Version control software (VCS)

# Introduction

- ◇ As an IT professional, you will come across these two words: **Git** and **Github**
- ◇ These are very popular tools used nowadays for **version control**
- ◇ The term **Version control** refers to **a system that records changes (versions) to a file or set of files over the time**
- ◇ In other words, these versions will help you **track the changes** in your codes/projects and if necessary, undo those changes as well.

1

# Git vs Github

What is the difference?

# Git vs Github

- ◇ **Git** is an **open-source, version control tool** created in 2005 by developers working on the **Linux operating system**;
- ◇ It is used to **manage versions of your code or files locally**, that is, **Git** is installed and maintained on your local computer system
- ◇ **GitHub** is a company founded in 2008
- ◇ It provides **a cloud-based hosting service** that lets you **store and manage Git repositories**.
- ◇ There are many other alternatives to **GitHub**, such as **GitLab, BitBucket..**

# Git vs Github

Git	GitHub
1. It is a software	1. It is a service
2. It is installed locally on the system	2. It is hosted on Web
3. It is a command line tool	3. It provides a graphical interface
4. It is a tool to manage different versions of edits, made to files in a git repository	4. It is a space to upload a copy of the Git repository
5. It provides functionalities like Version Control System Source Code Management	5. It provides functionalities of Git like VCS, Source Code Management as well as adding few of its own features



## 2

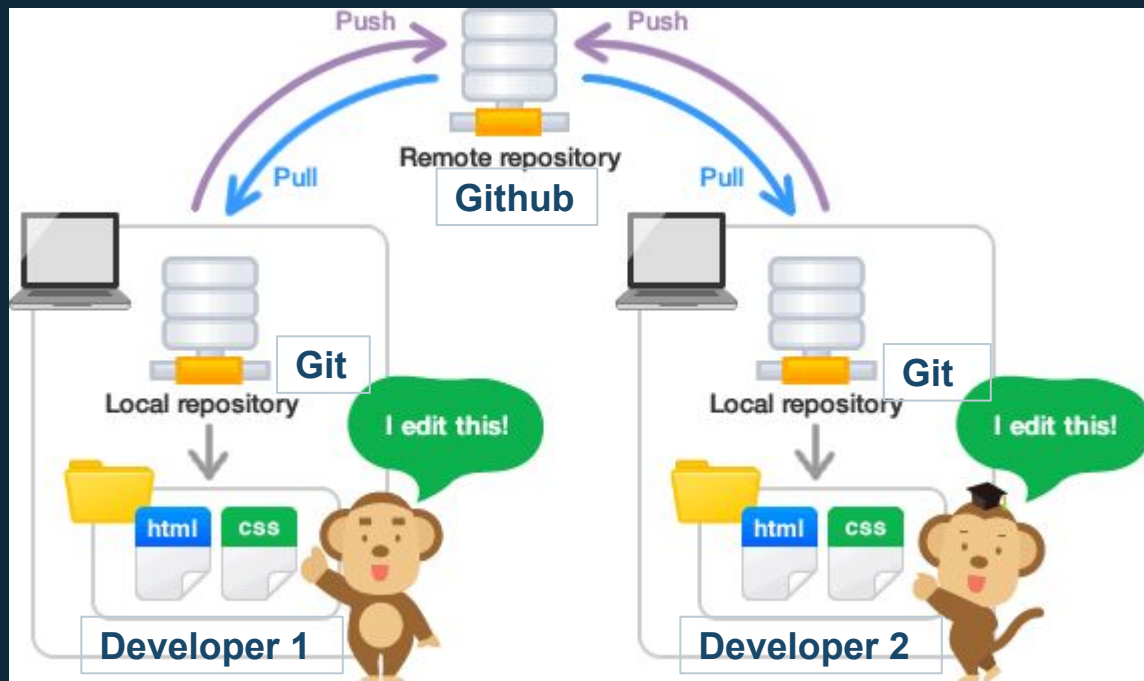
# Why do we need git or github

Collaboration on projects

# Why git or github?

- ◇ **GitHub** makes it easier to collaborate on projects using **git**.
- ◇ Multiple developers can work on a single project and see each others' edits in real-time in the cloud:
- ◇ This reduces the risk of duplicative or conflicting work, and can help decrease production time.
- ◇ With Git, developers can
  - **build** code, **push** it to **Github** to share with collaborators,
  - **track** changes, and **innovate** solutions to problems that might arise during the development process.

# Git vs Github



3

# GitHub essentials

GitHub language



# GitHub essentials

◇ While using **Git** and **Github**, there are some **keywords** you will often come across:

- **Repositories**
- **Branches**
- **Commits**
- **Pull requests**

◇ Let's briefly explain some of these concepts



# GitHub essentials

## Repository

- ◇ A **Repository** is like a directory that contains all the files and folders of a specific project
- ◇ It generally contains a **licence** and a **README** file about the project
- ◇ We can create **local repositories with git** or **online repositories in GitHub**
- ◇ A **GitHub** repository can also be used to store ideas or any resources that you want to share with your team or the world

# GitHub essentials

## Branch

- ◇ A **branch** is used to work with **different versions of a repository**. That is, a branch can contain a whole version of a project
- ◇ Each developer can work on a specific branch of the project at any time
- ◇ By default, a repository has a **main (master) branch** and any other branch is a copy of the main branch at a specific point in time
- ◇ Branches are generally created by developers or team members to work on **separate features** of the project before **merging it with the main branch**

# GitHub essentials

## Commits

- ◇ A **commit** is like a **change you do to the project**
- ◇ Each commit (change) has a **description (commit message)** explaining **why a change was made**, an **ID** to identify the commit, information about **the Author** (who made the change) and **the date** (when)

## Pull Requests

- ◇ With a **pull request** you are proposing that your changes should be **merged (pulled in)** with the **main branch (master)**
- ◇ As soon as you have a **commit**, you can open a **pull request**



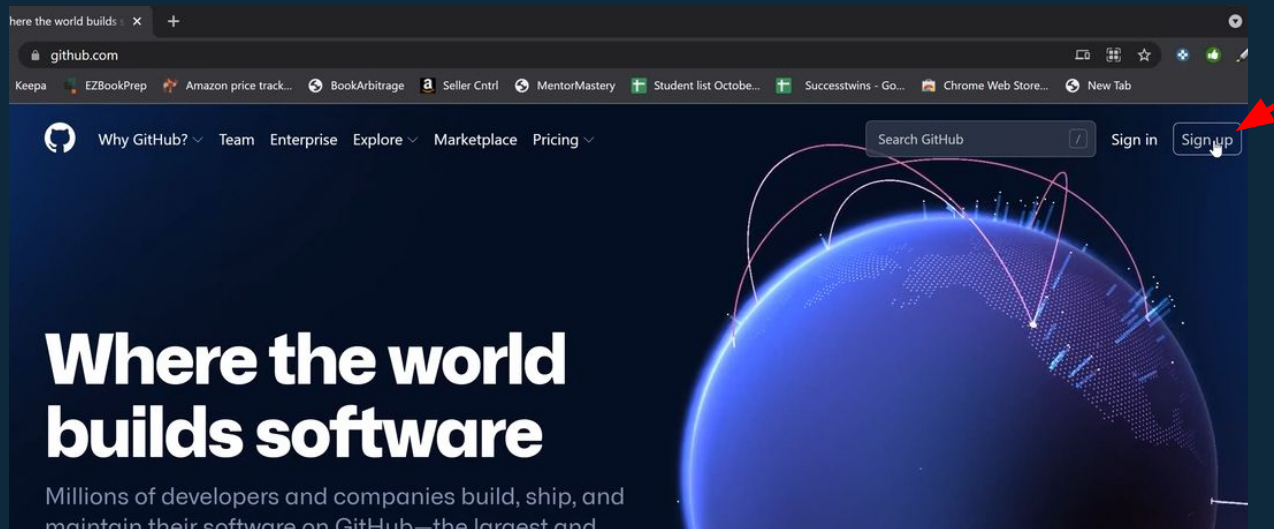


Sign up to GitHub



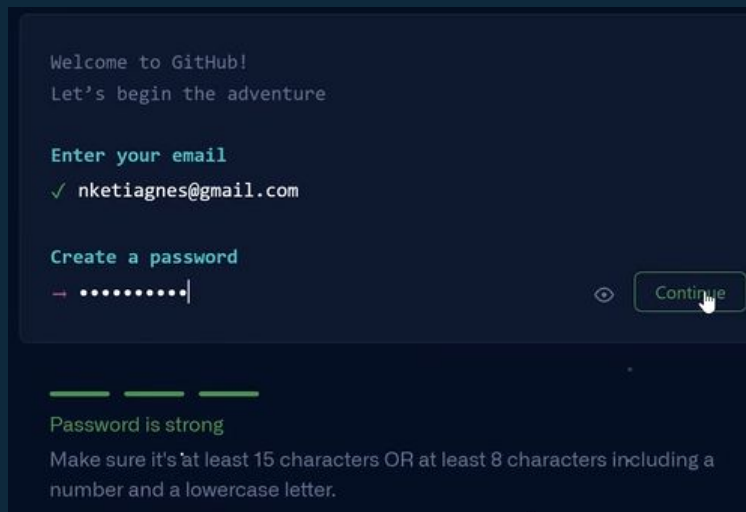
# Sign up to GitHub

- ◇ To use **GitHub**, we need to create an account on the **GitHub platform**
- ◇ Launch your google chrome browser and open [github.com](https://github.com)
- ◇ You will get the following page: Click on **Sign up**



# Sign up to GitHub

- ◇ Enter your **email address**
- ◇ This email must be valid
- ◇ Create a **strong password**
- ◇ Note your email and password and **carefully keep** that somewhere (We will use it later)
- ◇ Click on **Continue**



Welcome to GitHub!  
Let's begin the adventure

Enter your email  
✓ nketiagnes@gmail.com

Create a password  
→ .....|

Continue

Password is strong

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter.

# Sign up to GitHub

- ◇ Enter a valid or available **username** (Here I used **sre2021**)
- ◇ To the question Would you like to receive product update and announcements via email? Type **n** for **no** and **Continue**

Enter a username

✓ sre2021

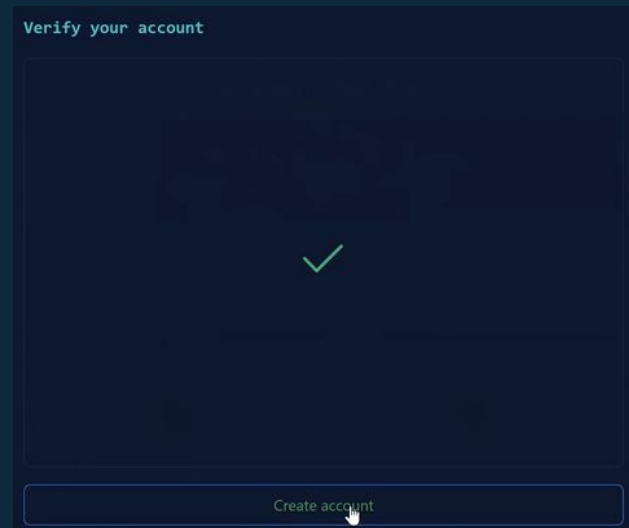
Would you like to receive product updates and announcements via email?

Type "y" for yes or "n" for no

✓ n

# Sign up to GitHub

- ◇ To verify your account, **read the message displayed** and do accordingly till the verification is **successful**
- ◇ Then click on **Create account**
- ◇ A **code** will be sent to the email address you provided.



# Sign up to GitHub

- ◇ Check it and type the code in the space provided: The following is mine, check yours and fill in

You're almost done!

We sent a launch code to `nketiagnes@gmail.com`

→ Enter code

1

2

5

0

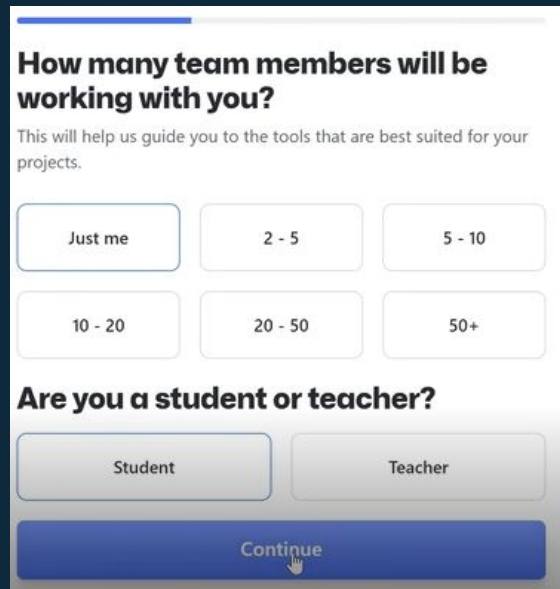
9

6

Didn't get your email? [Resend the code](#) or [update your email address](#).

# Sign up to GitHub

- ◇ How many team members
  - Choose **Just me**
- ◇ Are you a student or a teacher?
  - Choose **Student**
- ◇ Click on **Continue**



**How many team members will be working with you?**

This will help us guide you to the tools that are best suited for your projects.

Just me	2 - 5	5 - 10
10 - 20	20 - 50	50+

**Are you a student or teacher?**

Student	Teacher
---------	---------


**Continue**

# Sign up to GitHub

- ◇ Make sure you check the **Collaborative coding** and the **Automation and CI/CD** checkboxes
- ◇ Allow other features for now
- ◇ Scroll down and click **Continue**
- ◇ Scroll down the various plan page and click on **Skip personalization**
- ◇ After this, you will be redirected to your **GitHub account home page**

## What specific features are you interested in using?

Select all that apply so we can point you to the right GitHub plan.

- ☒  **Collaborative coding**  
Codespaces, Pull requests, Notifications, Code review, Code review assignments, Code owners, Draft pull requests, Protected branches, and more.
- ☒  **Automation and CI/CD**  
Actions, Packages, APIs, GitHub Pages, GitHub Marketplace, Webhooks, Hosted runners, Self-hosted runners, Secrets management, and more.



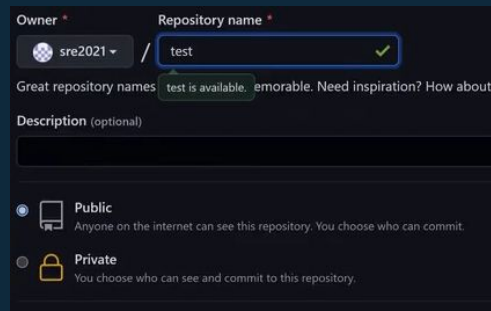
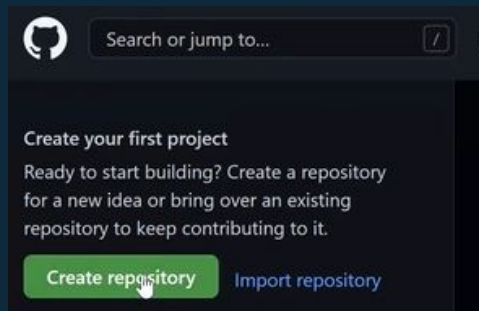


Create our first  
GitHub repository



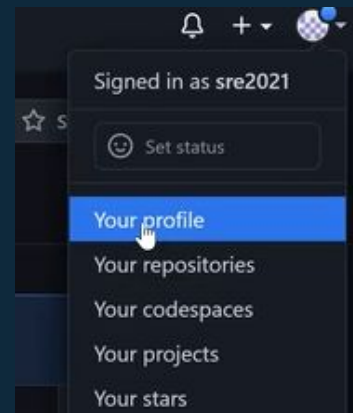
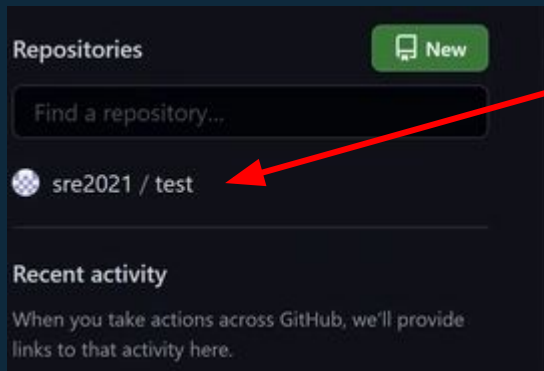
# Create GitHub Repo

- ◇ For now, our account is still empty
- ◇ To create our first repository in GitHub, just click on **Create repository**
- ◇ Repository name: **test**
- ◇ Allow it **Public** repository
- ◇ Scroll down and click on **Create repository**



# Create GitHub Repo

- ◇ Click on the **GitHub logo** at the top of the page to return to your account home page
- ◇ Click on the **repo name** to check its default content
- ◇ You can Also modify your profile information by clicking on the **dropdown**
- ◇ You can modify your **profile picture** and put a professional picture of you



## GitHub is a very useful and powerful tool

It can be used to search and utilize other developers or team projects

In the search bar: just type **vagrantfile** or **vagrant-docker**



Search or jump to...



You will get a whole bunch of repositories on these topics that may help you when working with docker!

## 4

# How to use git?

Local Installation and git commands



# Git installation



# Install git

- ◇ To use **Git** on our computer, we first need to install it
- ◇ To do that, launch your google chrome browser, and search for the term **git**
- ◇ Click on the first result from <https://git-scm.com>
- ◇ Click on **Download for Windows** (For mac computers you will see there **Download for MAC**)





# Install git

◇ If you get the following page, just click on [Click here to download](#)

## Download for Windows

[Click here to download](#) the latest (**2.35.0**) **64-bit** version of **Git for Windows**. This is the most recent **maintained build**. It was released **3 days ago**, on 2022-01-24.

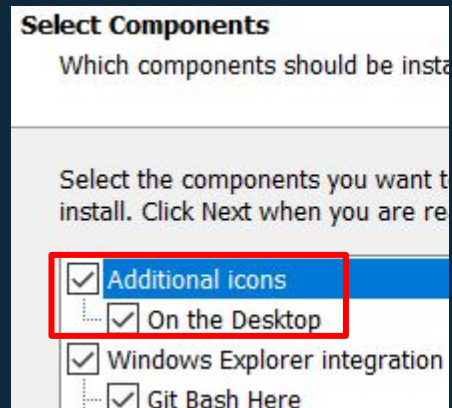
**Other Git for Windows downloads**





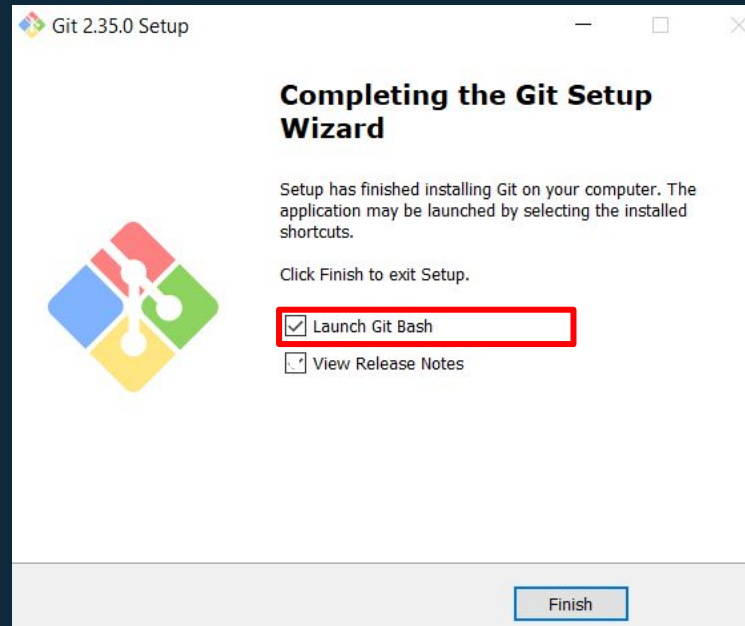
# Install git

- ◇ When it is done downloading, click on it to start the installation process
- ◇ Click **Yes** if you are asked to **Allow the app to get installed**
- ◇ Click on **Next** to accept the GNU Licence
- ◇ Click on **Next** to select the default location
- ◇ Check the **Additional icons** box then click on **Next**
- ◇ Now, click on **Next** and **install** until the process is complete



# Install git

- ◇ Check the **Launch Git Bash** box and **Uncheck the View Release Notes** box
- ◇ Click on **Finish**
- ◇ The installation is complete and the **Git Bash window** will open





On Mac, it might look a little bit different. At the end of the process it might open your regular Terminal. Just run the command **git version** to make sure git is installed

# Install git

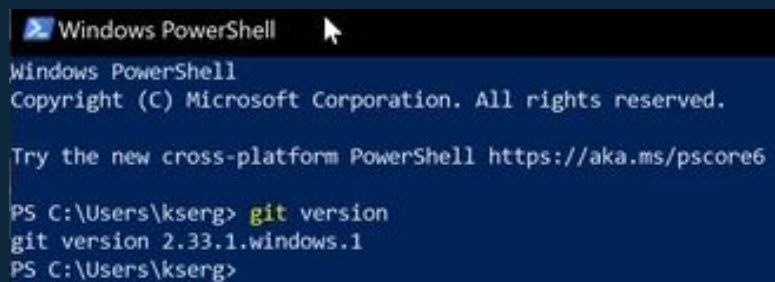
- ◇ To check the version that was installed, run the command \$ **git version**

```
MINGW64 ~/C:/Users/kserg
kserg@DESKTOP-023NQ4M MINGW64 ~
$ git version
git version 2.33.1.windows.1
```

- ◇ You can use **Ctrl +** or **Ctrl -** to **adjust the font-size**
- ◇ If you close all, you can see the icon of **Git Bash** on your windows Desktop
- ◇ You can easily launch the **Git Bash** from there
- ◇ You can also use the Windows search bar by typing **bash** to get the **Git Bash app**

# Install git

- ◇ You can also use **Git** from the **Powershell** on your windows computer
- ◇ Search for **Powershell** in your Windows search bar and launch the app
- ◇ Check the git version installed with the command \$ **git version**

A screenshot of a Windows PowerShell terminal window. The title bar says "Windows PowerShell". The text inside shows the standard PowerShell startup messages: "Windows PowerShell", "Copyright (C) Microsoft Corporation. All rights reserved.", and "Try the new cross-platform PowerShell https://aka.ms/pscore6". Below this, the user has entered the command "git version" and the output is "git version 2.33.1.windows.1". The prompt "PS C:\Users\kseng>" is visible at the bottom.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

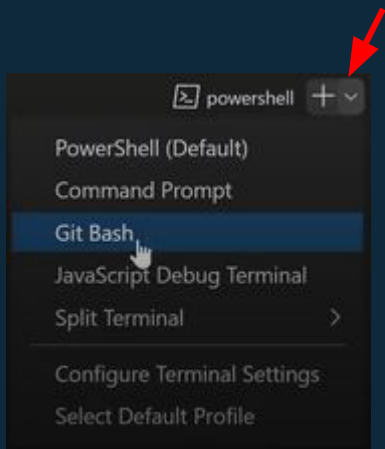
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\kseng> git version
git version 2.33.1.windows.1
PS C:\Users\kseng>
```

**Note:** For macs, just use your regular **mac terminal**

# Install git

- ◇ Another way to access the git bash terminal is to use the **Visual studio code**
- ◇ Click on the dropdown and choose **Git Bash** to open a **git bash terminal**
- ◇ **Git Bash** accept some commands that Powershell don't. Example: id
- ◇ You can switch between **Git Bash** and **Powershell terminal**





# Basic Git Commands



# Git commands

- ◇ To create and manage projects or repositories, we use **git commands** in the terminal
- ◇ Open your **Visual Studio code** and launch a **Git Bash terminal**
- ◇ Let's create a folder on our **Desktop** for our **git repository**
  - \$ **cd Desktop**
  - \$ **mkdir tes-repo** then \$ **cd tes-repo**
- ◇ Now, we will use the **git init command** to initialize a repository in there:
- ◇ Just run the command: \$ **git init**



# Git commands

- ◇ For windows users, you will see (**master**) at the end of the line. But for mac users you won't have that

```
kserg@DESKTOP-023NQ4M MINGW64 ~/Desktop/tes-repo
$ git init
Initialized empty Git repository in C:/Users/kserg/Desktop/tes-repo/.git/

kserg@DESKTOP-023NQ4M MINGW64 ~/Desktop/tes-repo (master) ←
```

- ◇ As previously mentioned, the **master** here represents the main branch of our project

# Git commands

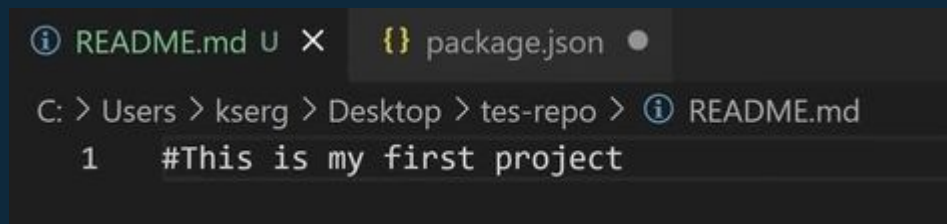
- ◇ To check the various branches we have created in a project, we run the commands: \$ **git branch**

```
$ git branch  
  
kserg@DESKTOP-023NQ4M MINGW64 ~/Desktop/tes-repo (master)  
$ ls  
  
kserg@DESKTOP-023NQ4M MINGW64 ~/Desktop/tes-repo (master)
```

- ◇ Check the content of the master branch with \$ **ls**: Nothing in there for now

# Git commands

- ◇ Let's create a file called **README.md** in this main branch:
  - `$ code README.md`
- ◇ The file will **open in a tab** of your Visual Studio code (reduce the terminal if necessary)
- ◇ Write a simple comment in there: **#This is my first project**
- ◇ Press **Ctrl** then **s** to save the changes (Ctrl-s)



```
❯ README.md U X {} package.json ●
C: > Users > kserg > Desktop > tes-repo > ⓘ README.md
1  #This is my first project
```

# Git commands

- ◇ In the Git Bash terminal, run `$ ls` to check the content of the branch
- ◇ To check the status of the files in our project, we run the command:
  - `$ git status`
- ◇ Here we can see an Untracked file: **README.md**

```
$ git status
On branch master

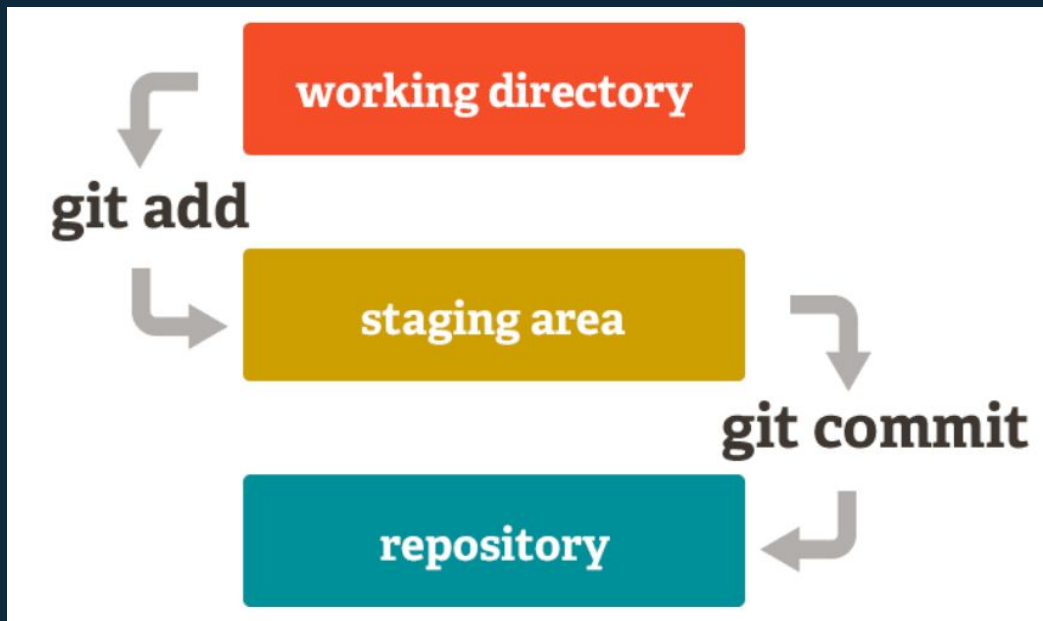
No commits yet

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)
```

# Git commands

◇ In git, we use the following workflow



# Git commands

- ◇ When a **file is created** for the project, it stays in the **working directory** in an **untracked state**.
- ◇ For the file to go to a **tracked state**, we must send it to the **Staging area** using the command **git add filename**
- ◇ The command **\$ git add .** will add **all modified and new (untracked) files** in the current directory and all subdirectories to the **staging area**
- ◇ After adding the file to the staging area, we need to use the command **git commit** to commit the changes to our local repository
- ◇ Let's practice that on our **README.md** file

# Git commands

- ◇ Still in your terminal, run the commands
  - \$ **git add .**
  - \$ **git commit -m "new repo"**
- ◇ After running this command, it **will throw an error**
- ◇ This because, before committing any change to our local repository, we need to **configure the git username and git email address**
- ◇ In fact, every **git commit** will use these configuration details to identify you as the **Author**

# Git commands

- ◇ The command use for that is **git config**:
  - \$ **git config --global user.email "your email here"**
- ◇ Use the email with which you created your github account
- ◇ Example I used: \$ **git config --global user.email "nketiagnes@gmail.com"**
  - \$ **git config --global user.name "your name here"**
- ◇ Example I used: \$ **git config --global user.name "Serge"**
- ◇ After this, recall the commit command: \$ **git commit -m "new repo"**



# Git commands

- ◇ Now, make a new change to the **README.md** file and commit:
- ◇ Add this line in the file : **#This is the second line** and **save** the file (Ctrl - s)

- ◇ In the terminal:

- **\$ git add .**
- **\$ git status**
- **\$ git commit -m "new change"**
- **\$ git status**

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
```

# Git commands

- ◇ To display all the commit informations of changes done in the project, we run the command: \$ **git log**

```
$ git log
commit 5af120731b169cf6c50d35e85cfc98c16f2c3fab (HEAD -> master)
Author: Serge <nketiagnes@gmail.com>
Date:   Fri Nov 12 15:37:28 2021 -0600

    new change

commit 8f1332b842656d5ec5c73162f1f2916e55669c21
Author: Serge <nketiagnes@gmail.com>
Date:   Fri Nov 12 15:33:07 2021 -0600

    New repo
```

Commit ID



Create a new  
branch



# Create new branch

To create a new branch identical to our **master branch**, we use the command `$ git branch branchName`

- ◇ Let's create a new branch called **feature1**
  - `$ git branch feature1`
- ◇ Now the master and feature1 branch have the exactly the same content
- ◇ To display all the branches of the project, we can simply run the command: `$ git branch`

```
ksereg@DESKTOP-023NQ4M MINGW64 ~/Desktop/tes-repo (master)
$ git branch
  feature1
* master
```

# Create new branch

- ◇ To move to the new branch we created, we use the **git checkout** command: `$ git checkout feature1`
- ◇ Verify with `$ git branch` (the star \* is now showing on feature1)
- ◇ Let's create 2 new files on the **feature1** branch:

- `$ touch file`

- `$ touch script.sh`

- `$ ls`

```
kseng@DESKTOP-023NQ4M MINGW64 ~/Desktop/tes-repo (feature1)
$ ls
file  README.md  script.sh
```

# Create new branch



Let's check, add and commit our new change

- `$ git status`
- `$ git add .`
- `$ git status`
- `$ git commit -m "new branch"`

```
$ git commit -m "new branch"
[feature1 f8e2605] new branch
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 file
create mode 100644 script.sh
```



# Create new branch

Let's modify the **README.md** file on the **feature1** branch:

- `$ code README.md`
- Add the line: **#This change is done on the feature branch** and save the changes Ctrl - s

◇ In the Terminal:

- `$ git status`

```
$ git commit -m "change to readme"
[feature1 27af9b9] change to readme
1 file changed, 2 insertions(+), 1 deletion(-)
```

- `$ git add .`

- `$ git commit -m "change to readme"`

◇ You can check the new content of **README.md** with: `$ cat README.md`

# Create new branch

- ◇ Now, let's switch back to the master branch:

- `$ git checkout master`

```
$ git checkout master  
Switched to branch 'master'
```

- `$ cat README.md`

```
kserg@DESKTOP-023NQ4M MINGW64 ~/Desktop/tes-repo (master)  
$ cat README.md  
#This is my first project  
#This is a second
```

- ◇ We can see that we have only 2 lines in the **README.md** file that is on the **master branch**
- ◇ There is an operation we will often do to **merge the content** of a specific branch with the content of the **master branch**



## 5

# Link local and online repository

Push to code to GitHub, clone the code from GitHub



# Local to Online

Push code to GitHub repository for others to access it at any time



# Local to online

- ◇ For better collaboration with the team on a project, we need to store and share the code in **GitHub**
- ◇ We need to **link the local repository to a GitHub repository** where others will have access to the code we developed locally
- ◇ When creating a new repository in **GitHub** (like the test repo we created earlier), it will provide the steps we can use to push an existing local repository to **GitHub** from the command line
- ◇ You can click on the **test repo** in your **GitHub account home page** and **scroll down to get the steps**

# Local to online

- ◇ Being in the **test** repo online, you will get the command needed and the link to the repository
- ◇ Now, let's push our local repository online:
  - \$ **git remote add origin paste the link here**
  - For me: \$ **git remote add origin <https://github.com/sre2021/test.git>**
  - Use the command \$ **git remote -v** to verify that the remote site was added to the list
  - Now, run: \$ **git push -u origin master**

**Note: We are not using **main** here!**

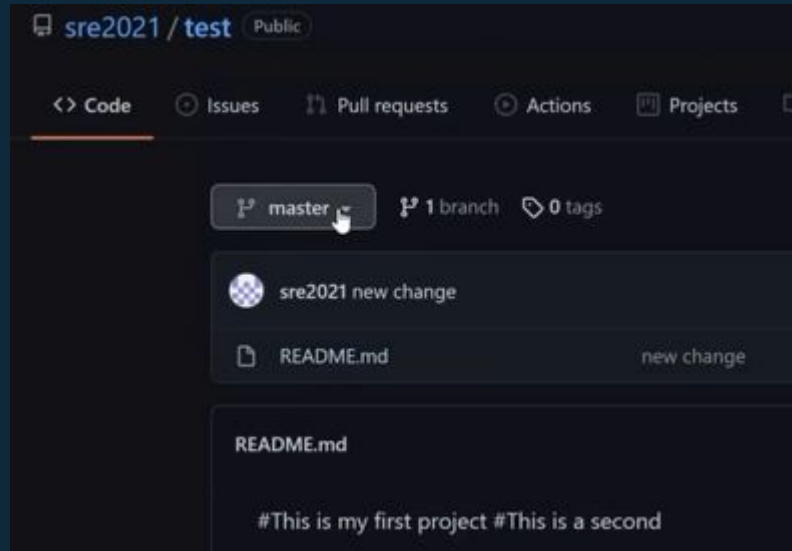
# Local to online

- ◇ The previous command might ask you to **authenticate**:
- ◇ Click on **Sign in with your browser**. This will open a tab in your browser
- ◇ Click on **Authorize GitCredentialManager** and make sure the **authentication succeeded**



# Local to online

- Now refresh your GitHub account page, you will see that a new branch (**master**) was added to the online repository

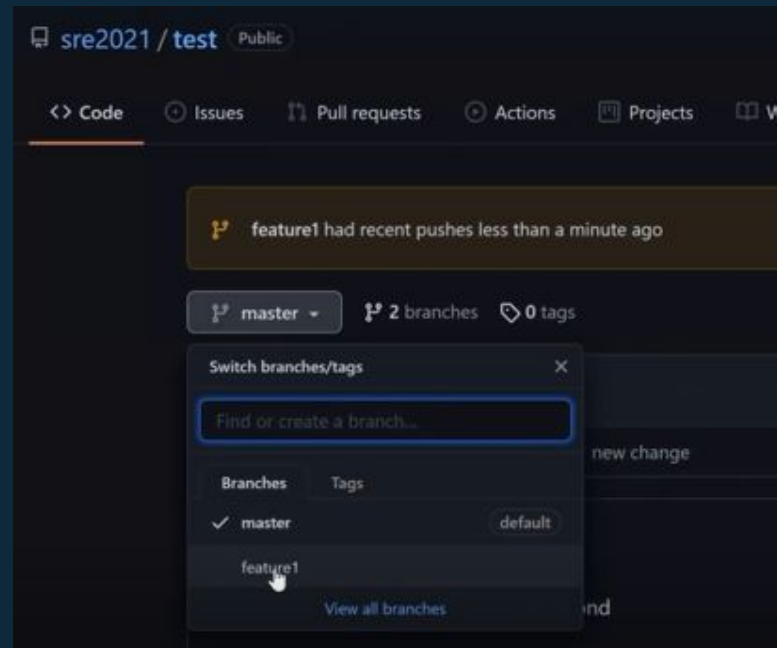


# Local to online

◇ Let's also push the **feature1** branch

- From your terminal, switch to the branch:
  - \$ **git checkout feature1**
- Push the branch online:
  - \$ **git push origin feature1**

◇ Refresh your GitHub account page:  
we have **2 branches** in the project





# Online to Local

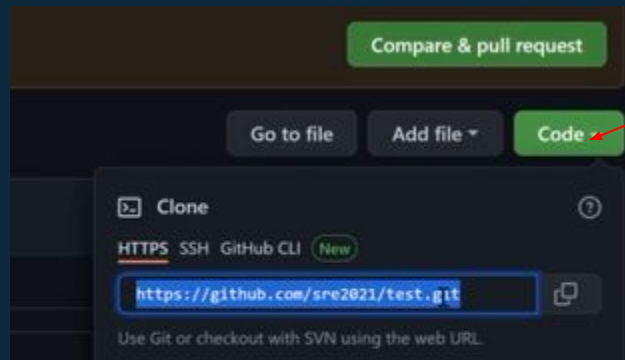
How can others copy the project and work with it locally on their own computer?





# Online to Local

- ◇ When the project has been pushed online, team members can access it and can also make a copy of it on their local machine
- ◇ To do that, they need to **clone** the project
- ◇ To clone the project, click on **Code** then **copy the link**



# Online to Local

- ◇ Let's pretend to be another team member by **opening a new git bash terminal**
- ◇ Let's create a folder on the local machine for our project:
  - \$ **cd Desktop** then \$ **mkdir repo1** then \$ **cd repo1**
- ◇ Now, let's use the **git clone** command to clone the online repo locally
  - \$ **git clone paste the URL here**
  - For me: \$ **git clone <https://github.com/sre2021/test.git>**
- ◇ This will clone the master branch of our Github repository

# Online to Local



You can check the content of the repo:

- `$ ls`
- `$ cd test`
- `$ ls`

```
$ ls
test/

kserg@DESKTOP-023NQ4M MINGW64 ~/repo1
$ cd test/

kserg@DESKTOP-023NQ4M MINGW64 ~/repo1/test (master)
$ ls
README.md
```

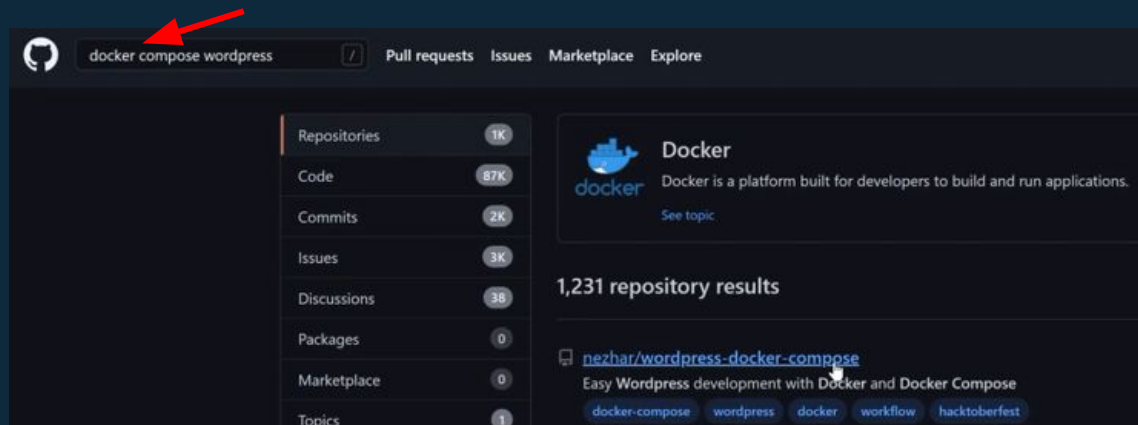


Fork a repo



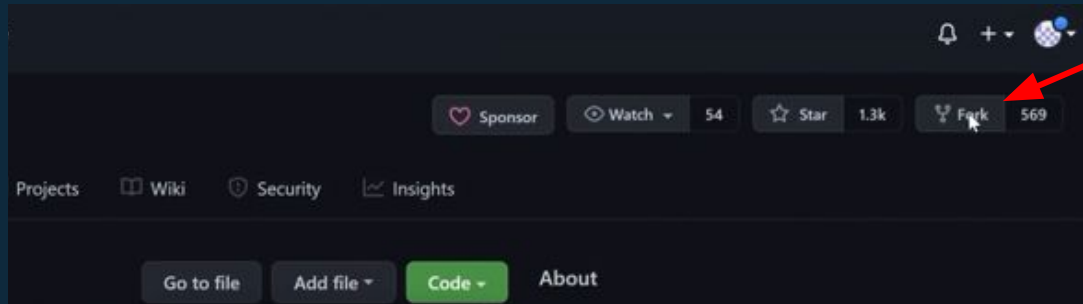
# Online to Local

- ◇ You can also **Fork** any repo online and put it in your GitHub account before working on it
- ◇ Let's practise that on a **docker-compose wordpress** repo online
- ◇ In your search bar, enter **docker-compose wordpress** and **validate**



# Online to Local

- ◇ Click on the first result displayed
- ◇ To **fork** this repository, click on the button **Fork**



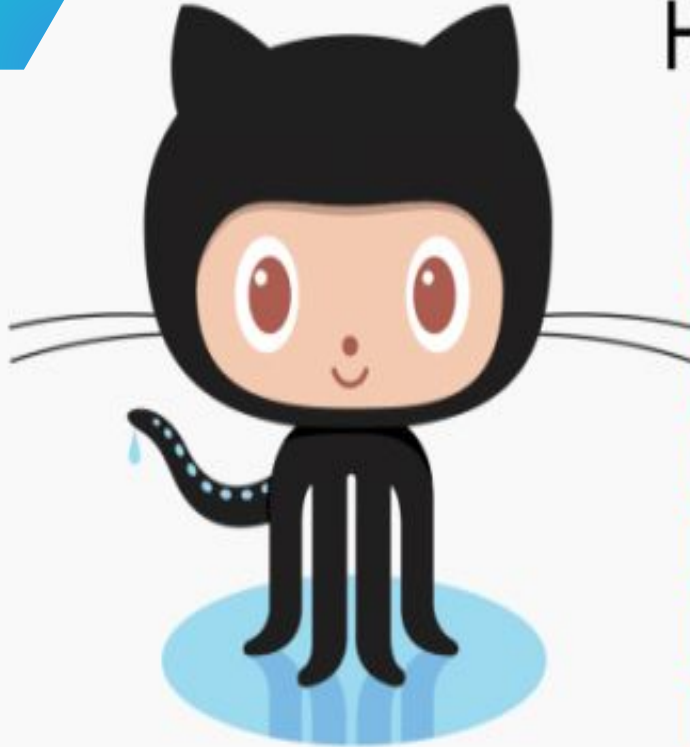
- ◇ Now, click on the **Github logo** to return to your home page
- ◇ Check the list of the repositories you have now: **It has been added!**



Work on this and follow all the steps to make sure you understand the concept. We will use this for **DevOps**

See you guys in the next lesson

# Summary



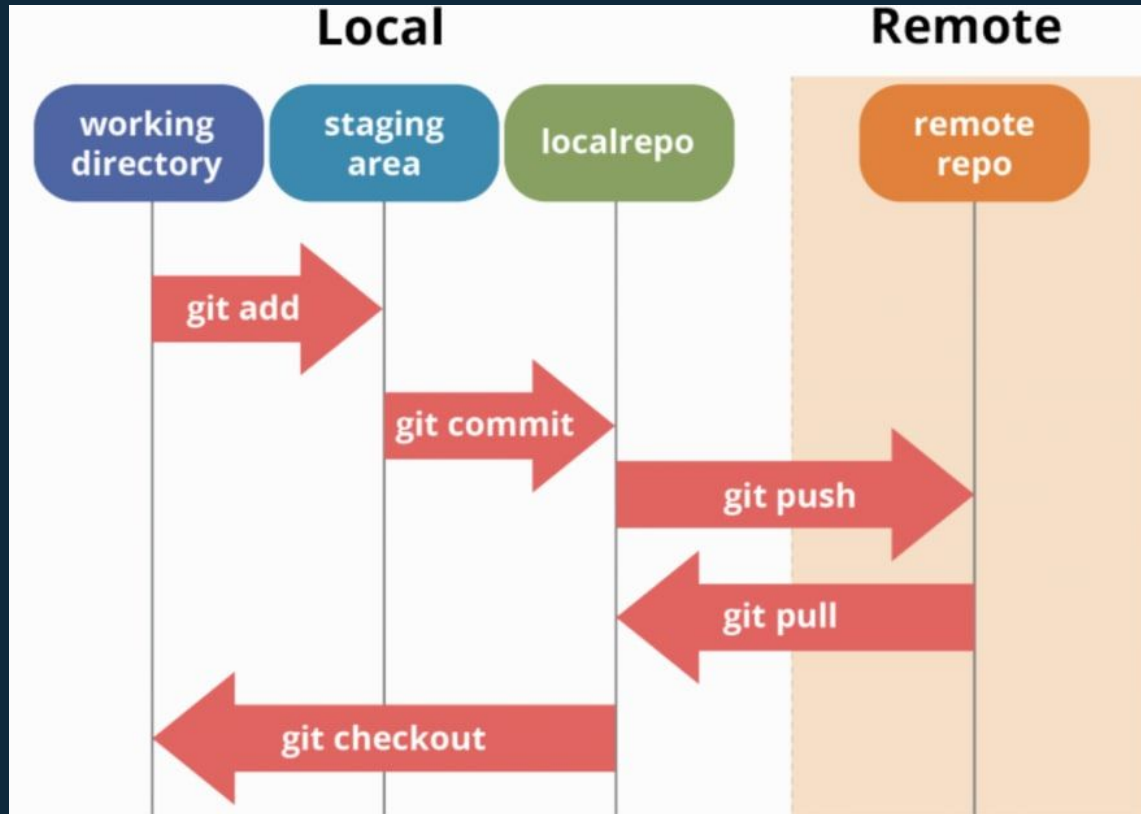
## How to use GitHub

- Step 1 ➤ Sign up for GitHub
- Step 2 ➤ Create repository
- Step 3 ➤ Install and set up Git
- Step 4 ➤ Clone the remote repository
- Step 5 ➤ Make changes to files
- Step 6 ➤ Add changes to the staging area
- Step 7 ➤ Commit changes
- Step 8 ➤ Push changes to the remote





# Basic GitHub commands





# Thanks!

## Any questions?

You can find us at:

**website:** <http://utrains.org/>

**Phone:** +1 (302) 689 3440

**Email:** [contact@utrains.org](mailto:contact@utrains.org)





Click on the link below to  
contact the support team  
for any issue!

[utrans.org/support/](https://utrans.org/support/)

Create a ticket for your problem and we will get back to you soon!

