Session 2: Python Essentials, Version Control, and Web Development Basics

Session 2: Version Control, Python 101, and Web Development Basics

- Version Control with Git and GitHub: Basic commands, branching, merging, and collaboration workflows.
- Python 101: Essential Python syntax and data structures relevant to AI programming.
- Web Development Introduction: Basics of HTML, CSS, and JavaScript to create simple web interfaces.
- Setting Up a Simple Web Server: Hosting applications locally to interact with AI models.

Introduction to GIT

Git (https://git-scm.com/) is a free and open-source distributed **version control system** designed to handle everything from small to very large projects with speed and efficiency.

A **Version Control System** is a software tool that helps developers manage and track changes to their codebase. With VCS, developers can easily collaborate on a project, share code, and maintain a history of their work.

As developers, we:

- create things,
- then save them,
- and subsequently at some other point in time:
  - edit them or
  - make changes or
  - make corrections or
  - make modifications based on some requests and
  - then save them again,
  - and subsequently at some other points in time:
    - edit them or
    - ....

Getting started with Git

How do we set it up?

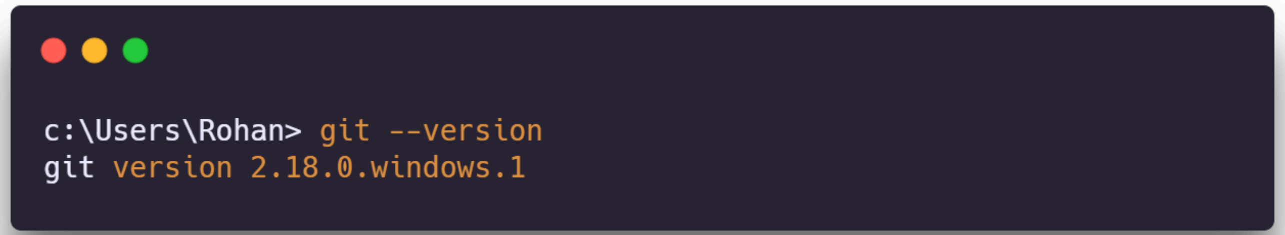
Git is primarily used via the command-line interface, which we can access with our system terminals. However, we first need to make sure that we have Git installed on our computers. Head over to https://git-scm.com/downloads (https://git-scm.com/downloads)

```
# Windows → Download the installer and install it

# MacOS
brew install git

# Ubuntu
apt-get install git
```

After installing it, start your terminal and type the following command to verify that Git is ready to be used on your computer.

```
git --version
```

If everything goes well, it should return the Git version that is installed on your computer.

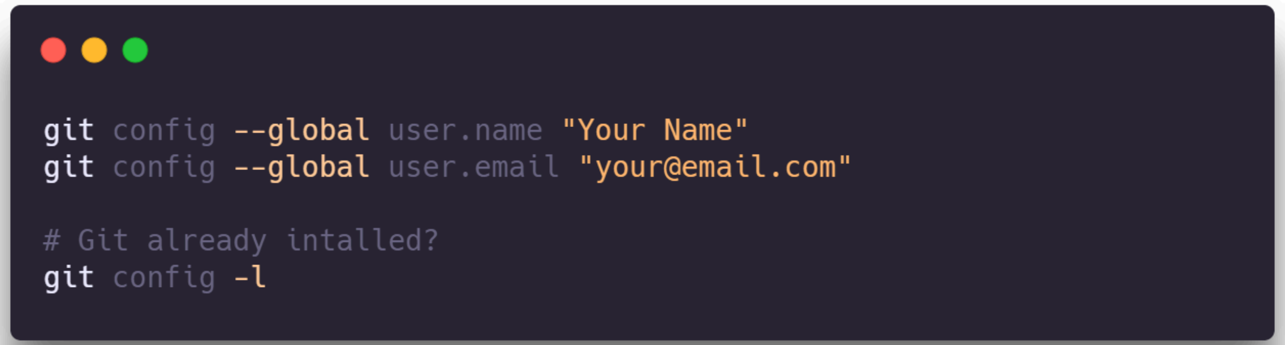


```
c:\Users\Rohan> git --version
git version 2.18.0.windows.1
```

Configuring your Name & Email

Git needs to know who you are and your email ID to track the changes you are going to make (imagine a company with 100s of developers).

In your terminal, run the following commands to identify yourself with git (if you already have git installed, then check your configuration by **git config -l**):



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

# Git already intalled?
git config -l
```

Repositories

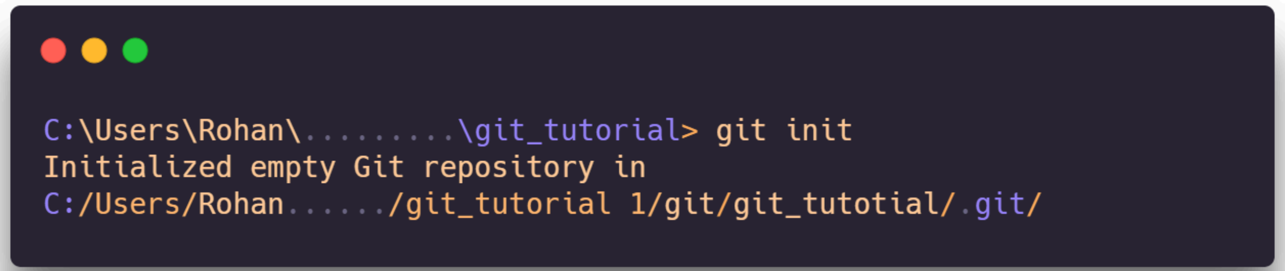
When working with Git, it is important to know that there are two types of repositories (a repository is a container for a project that is tracked by Git):

Local Repo: an isolated repo stored on your computer, where you can work on the local version of your project

Global repo: generally stored outside of your computer/local system, usually on a remote server (for example GitHub)

Initializing a Repository

To create a new repository and start tracking your project with Git, you need to navigate to the main folder of your project in the terminal and then type:



```
C:\Users\Rohan\.........\git_tutorial> git init
Initialized empty Git repository in
C:/Users/Rohan....../git_tutorial 1/git/git_tutotial/.git/
```

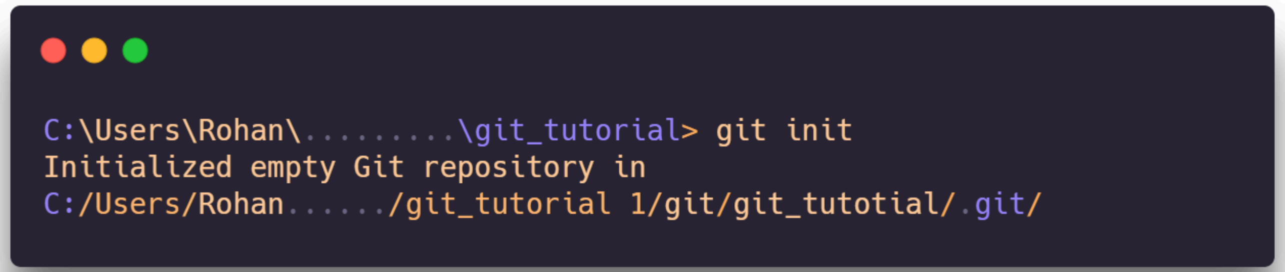
This command generated a hidden **.git** directory for your projects, where Git stores all internal tracking data for the current repo.

Staging

Committing is the process in which the changes are "officially" added to the Git repo. These are the "points" in history we can go back to for our review, even revert to this point.

Since "commitment" is a serious business, before we can actually commit, we need to place our changes inside the **staging area**.

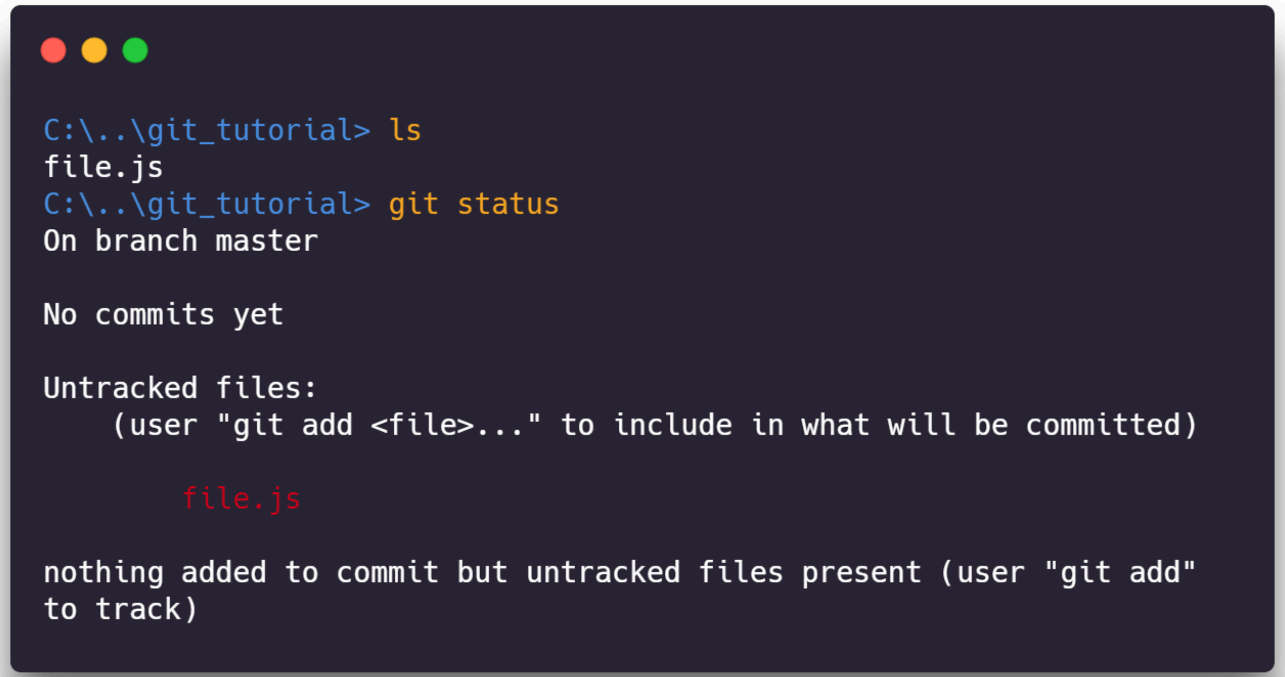
First, let's check the status of our folders/files:



```
C:\Users\Rohan\.........\git_tutorial> git init
Initialized empty Git repository in
C:/Users/Rohan....../git_tutorial 1/git/git_tutotial/.git/
```

The instructions are pretty clear above.

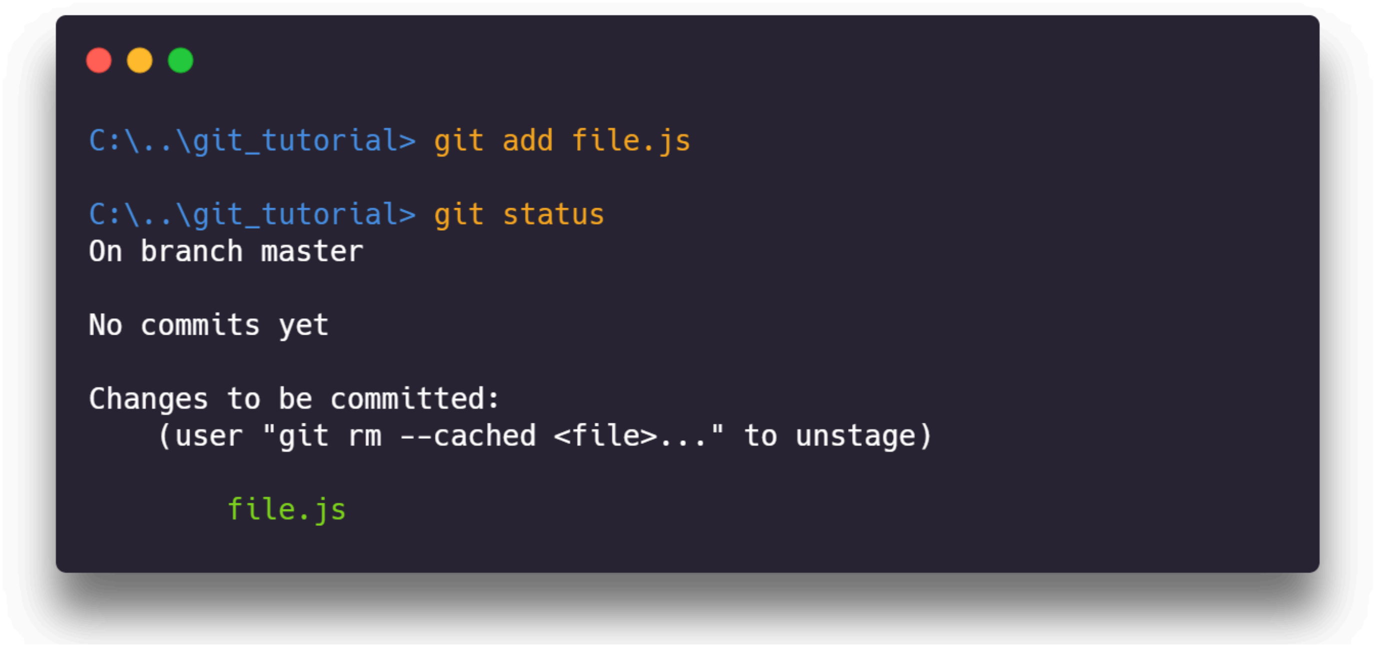
Let's add a file and check the status again:



```
C:\..\git_tutorial> ls
file.js
C:\..\git_tutorial> git status
On branch master

No commits yet

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

        file.js

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

We can use the command **git add** to add our files to the staging area, which allows them to be tracked

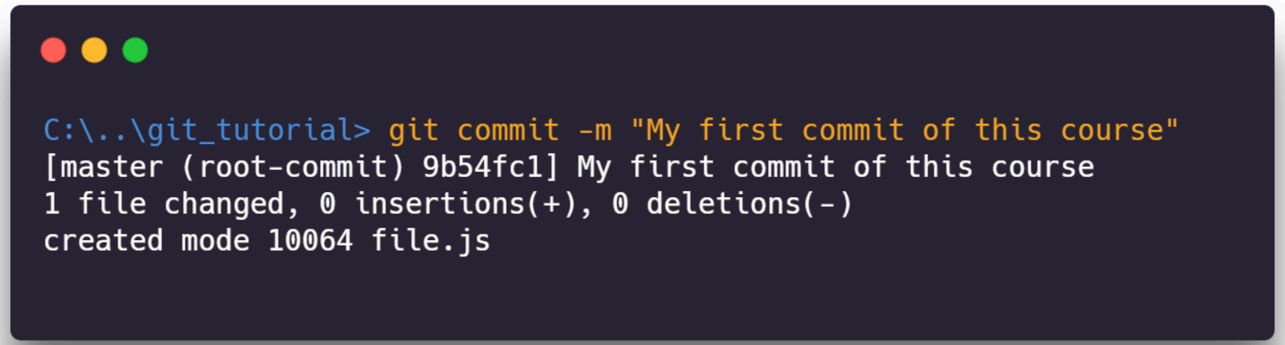
We can add specific files or all the files together.



```
C:\..\git_tutorial> git add file.js

C:\..\git_tutorial> git status
On branch master

No commits yet

Changes to be committed:
    (user "git rm --cached <file>..." to unstage)

        file.js
```

Making commits

A commit is a snapshot of our code at a particular time, which we are saving to the commit history.
After adding all the files that we want to track to the staging area with the **git add** command, we are ready to make a commit:



```
C:\..\git_tutorial> git commit -m "My first commit of this course"
[master (root-commit) 9b54fc1] My first commit of this course
1 file changed, 0 insertions(+), 0 deletions(-)
created mode 10064 file.js
```

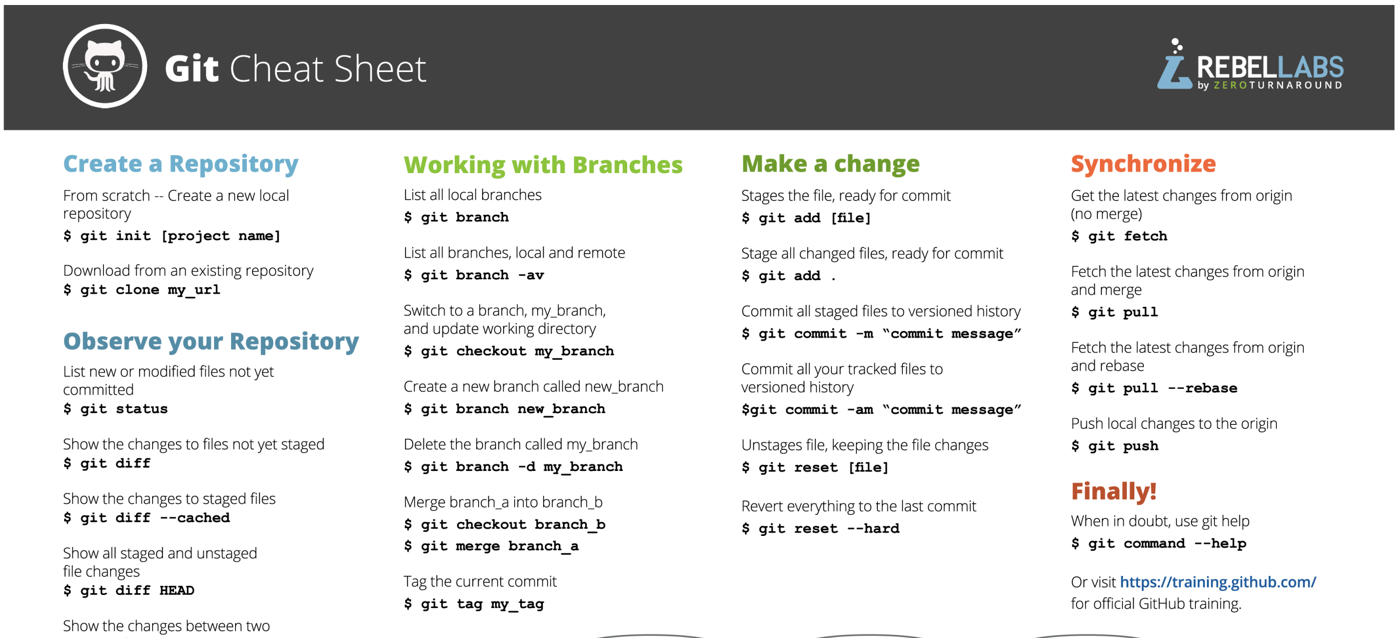
Commit History

To see all the commits that were made for our project, you can use the command **git log**.

Now, let's add some text to our file, check **git status**, then **git commit**. Oh! we get an error, so **git add .** then **git commit -m "message"**, then **git log**.

Copy the hash of the old commit, and then **git checkout *HASH***. Isn't it wonderful!

Here is a Git cheat sheet that you should get used to in the time to come.



Git Cheat Sheet

REBELLABS by ZEROTURNAROUND

Create a Repository

From scratch -- Create a new local repository
```
$ git init [project name]
```

Download from an existing repository
```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed
```
$ git status
```

Show the changes to files not yet staged
```
$ git diff
```

Show the changes to staged files
```
$ git diff --cached
```

Show all staged and unstaged file changes
```
$ git diff HEAD
```

Show the changes between two commit ids
```
$ git diff commit1 commit2
```

List the change dates and authors for a file
```
$ git blame [file]
```

Show the file changes for a commit id and/or file
```
$ git show [commit]:[file]
```

Show full change history
```
$ git log
```

Show change history for file/directory including diffs
```
$ git log -p [file/directory]
```

Working with Branches

List all local branches
```
$ git branch
```

List all branches, local and remote
```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory
```
$ git checkout my_branch
```

Create a new branch called new_branch
```
$ git branch new_branch
```

Delete the branch called my_branch
```
$ git branch -d my_branch
```

Merge branch_a into branch_b
```
$ git checkout branch_b
$ git merge branch_a
```

Tag the current commit
```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit
```
$ git add [file]
```

Stage all changed files, ready for commit
```
$ git add .
```

Commit all staged files to versioned history
```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history
```
$git commit -am "commit message"
```

Unstages file, keeping the file changes
```
$ git reset [file]
```

Revert everything to the last commit
```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)
```
$ git fetch
```

Fetch the latest changes from origin and merge
```
$ git pull
```

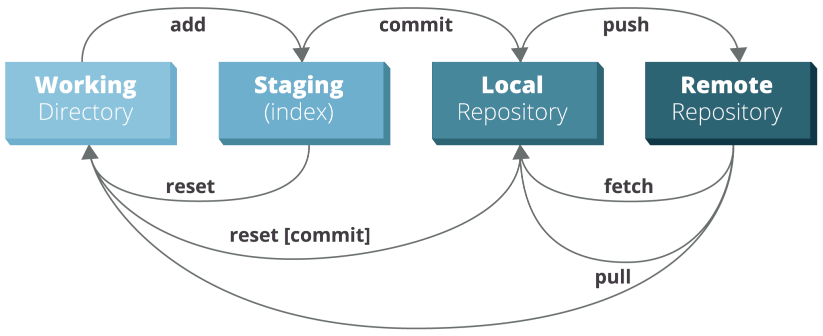
Fetch the latest changes from origin and rebase
```
$ git pull --rebase
```

Push local changes to the origin
```
$ git push
```

Finally!

When in doubt, use git help
```
$ git command --help
```

Or visit https://training.github.com/ for official GitHub training.



Let's look at a hands-on!

Python 101

For Python 101, let's jump to code directly. Here is the link to the [Python 101 code (https://canvas.instructure.com/courses/12597696/files/308578133?wrap=1)](https://canvas.instructure.com/courses/12597696/files/308578133?wrap=1) [(https://canvas.instructure.com/courses/12597696/files/308578133/download?download_frd=1)](https://canvas.instructure.com/courses/12597696/files/308578133/download?download_frd=1) that we covered in the class

Web Development Introduction

Here we'll go through full hands-on. Here are your instructions (https://canvas.instructure.com/courses/12597696/files/308578174?wrap=1) (https://canvas.instructure.com/courses/12597696/files/308578174/download?download_frd=1) .

Assignment

You have the code for S2_Class_demo. Do it once before attempting this assignment.

1. Use Cursor/GhatGPT for all of this
2. Create a simple creative AI-related application (you'll ask ChatGPT or Gemini to code, not Cursor, save it), here are some examples:
    1. Image Filter Demo. Write your custom 3x3 numbers, and convolve it on top of image and show results
    2. Upload 4-5 images, calculate mean of those images, apply mean and see how those images loog
    3. Extend my example, and show total parameters as well
    4. Word to One-Hot vectors (write 10-n words, and get one-hot vectors for them)
    5. Token length checker (enter a paragraph, app shows token counr (just split by spaces for simplicity)
3. Test it locally. Keep it very simple, 2-3 max edits with ChatGPT should do.
4. Upload on GitHub

5. Setup EC2, and move it there.
6. Play with it
7. Create a video and share the YouTube link
8. Share the link to GitHub
9. Share the link to your AWS Public URL (you can close it after 1 hour of your submission)

VIDEOS

STUDIO (Transcript (https://canvas.instructure.com/courses/12597696/files/309177762?wrap=1) (https://canvas.instructure.com/courses/12597696/files/309177762/download?download_frd=1))

GMeet

Support