Version control with Git and VS Code

What is version control?

Version control (also known as source control) is a system that tracks changes to the file a developer is currently working on. As a Python developer you will be required to work on several code files and to make changes to those files (writing new coding, improving old code, etc.).

A Version Control System (VCS) allows you to track the changes that you make to a file. It also allows you to revert files to a previous state, revert an entire project back to a previous state, compare changes to a file over time, shows you who made changes to a file and when, and helps you to resolve issues.

In a professional setting, when working with several developers, or on client projects knowing how to use a Version Control System is a must.

What is Git?

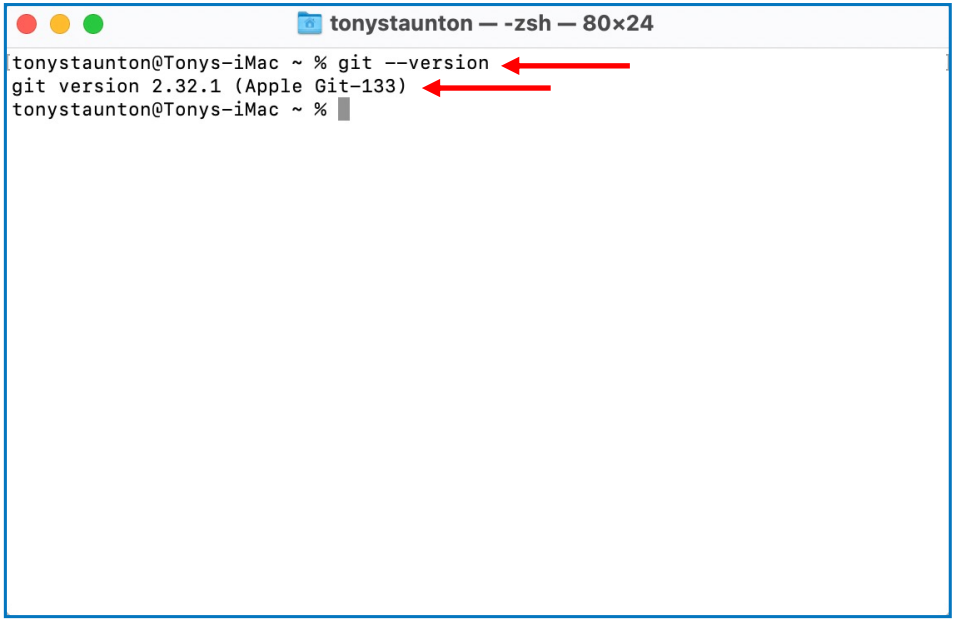
Git is the most popular Version Control System used by developers today. When using Git, every time you make a change to a file, Git takes a snapshot of that file and stores a reference to it. If a file has not changed, Git doesn't store the file again, just a link to the previous identical file it already stored.

As a Python developer, Git will help you track your work without having to worry about mistakes and help you to collaborate with other developers and clients.

Git is open source, free, and works across all operating systems. In this video we are gong to learn how to install Git for macOS and Windows and configure Git in VS Code.

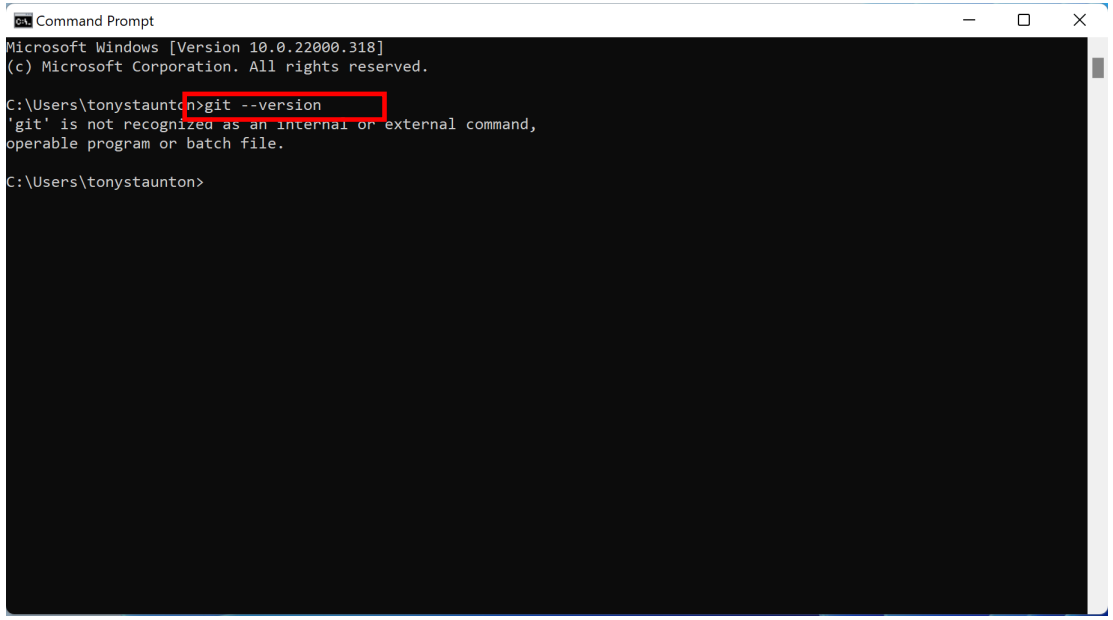
Installing Git on macOS

In your terminal, enter the command: *git --version*



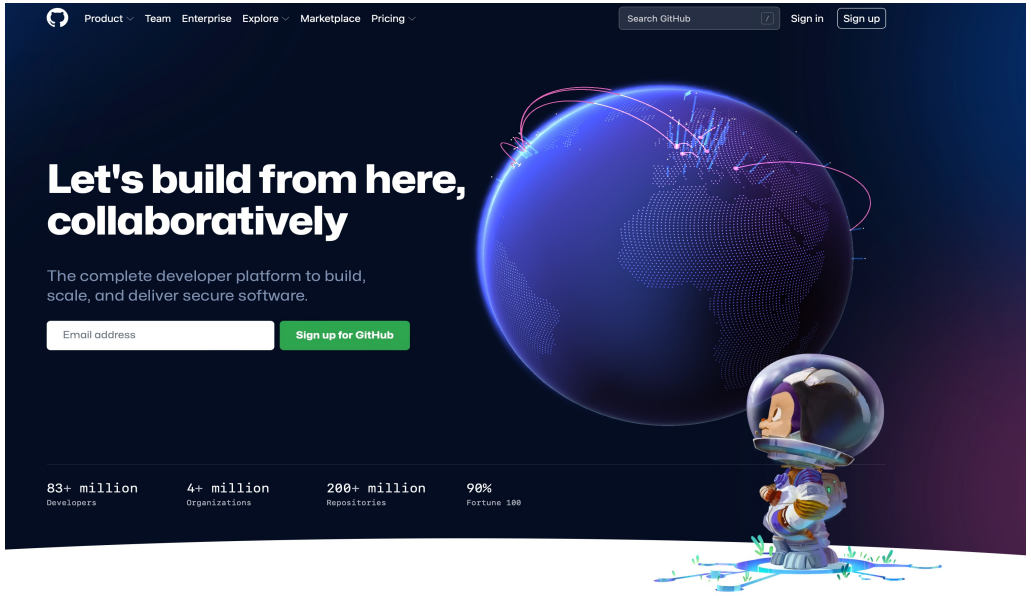
Installing Git on Windows

Go to: https://git-scm.com/



Setup GitHub

Go to www.github.com and create a GitHub account



Configuring Git

As mentioned, Git tracks changes that you make to a file. To do this you need to provide Git with a username and email address. Make sure to enter the same username and email address you used to create a GitHub account

In your terminal enter the commands:

git config --global user.name “your_username”
git config --global user.email “username@emailaddress.com”

Check you configuration

Use the following command to make sure that everything is configured as expected:

git config --global --list

Enable Git in VS Code

Open VS Code and go to *Settings*

In the search bar type “*Git: Enabled*”

Make sure that the check box is ticked

Now that Git is installed and enabled in VS Code, you can access Source Control from the left-hand menu.

Working with Git

To begin create a project folder that you can easily access. I am going to place my project folder on my desktop.

Using your code editor, create a new Python file called *hello_world.py* and save it to your project folder.

Inside the file *hello_world.py* add one line of code:

print(“hello world!)

Ignoring files

As you write Python programs you will start to notice files with the extension *.pyc* being generated automatically from your *.py* files. These files are stored in a folder called *__pycache__*. We don't need Git to track them, so we tell Git to ignore them.

To ignore the *__pycache__* folder, create a new file inside your project folder called *.gitignore,* make sure to include the dot(.) at the start of the filename, and don't include a file extension. Open this new file and add the following line:

_ _pycache_ _/

This file will tell Git to ignore all files in the *__pycache__* folder.

Initializing a repository

A git repository is the location where changes made to your files are tracked. By tracking the file changes in a repository, a history is built up over time. A Git repository is the *.git/* folder inside your project folder. If you delete this folder, then you delete your project history.

To initialize a repository, open your terminal and go to your project folder we just created. Enter the following command:

git init

The result should confirm that Git has initialized an empty repository.

Checking the status

In your terminal and still in your project folder, enter the following command:

git status

Adding files to a repository

Let’s add the file that we have been working on to our repository. In your terminal, and still in your project folder, enter the following command:

git add .

Now, check the git status again:

git status

The command *git add .* adds all the files in a project folder that are not currently being tracked to the repository. It doesn’t commit the files, simply adds them.

Our first commit

In the terminal, and still in the project folder, enter the following command:

git commit -m “Hello World project”

This command takes a snapshot of the project. Using *–m* tells Git to attach the message that follows in the project’s log.

Now, check the status again:

git status

Checking the log

In the terminal, and still in the project folder, enter the following command:

git log

Every time a commit is made, Git creates a unique 40-character reference ID. It also records who made the commit, when it was made, and the message attached. If you don’t always need this much information you can use the command:

git log --pretty=oneline

This command tells Git to output online one line of information, the reference ID and the attached message.

Our second commit

Let’s modify our hello_world.py program by adding a new line:

print(“Hello Tony!”)

Now, check the status of the project folder, and Git should have noticed the file change:

git status

Now, let’s commit the new changes:

git commit –am “Added name to message.”

To attach the new message, we’ve used the *–am* flag. The -a flag tells Git to add all modified files in the repository to the commit.

Reverting a change

Let’s make another change to our code file, *hello_world.py by adding the line:*

print(“Hello Frank!”)

Now, check the project status:

git status

Git knows that we have updated *hello_world.py. Instead of committing this change lets revert to the last commit we made. We do this with the following command:*

git checkout .
git status

Reverting a change

git checkout . Allows you to work with any previous commit.

git checkout . Will abandon any changes made since the last commit and restore your file to the last committed state.

Now, go back to your code editor and your code should have revert to its original state. You may need to refresh your code editor to view the changes.

Working with previous commits

Git allows you to check out any commit from your log, it doesn't have to be just the last one. You can do this by including the first six characters of the reference ID instead of a dot (.).

This feature allows you to review previous commits or go back to a previous working commit.

git checkout abcdef

Checking out a previous commit means that you are leaving the master branch of your project. When this happens, Git refers to it as a *detached HEAD state*. HEAD is the current committed state of your project. The reason it is referred to as *detached* is because you have left a named branch (master).

Working with previous commits

To return to the master branch you must check it out using:

git checkout master

With this command you will now be returned to the master branch.

Best practice tip: Try not to make any changes to your project when you've checked out an old commit. If you want to go back to a previous state, reset your project to a previous commit. You can do this from the master branch with the following commands:

Working with previous commits

git status

git log

git reset –hard abcdef

git status

git log

Working with previous commits

With these commands, we are first checking the status to make sure that we are working on the master branch.

The *git reset --hard* command tells Git that we want to revert to this commit permanently.

Again, we check the status of the log to make sure that we are on master and that there is nothing to commit.

Finally, we check the log again to make sure that we are working with the new/previous commit.

Deleting a repository

There will be times when mess up and get so so confused with your projects history that you will just want to delete everything and start again with the code you have. When you reach this point, you can do the following, remove the projects history by deleting the .git directory. This will delete all commits. Next you will need to start again with a fresh repository:

git status

*rm -rf .git (*Windows use *rmdir /s .git)*

git init

git status

Deleting a repository

git add .

git commit -m “Fresh repository”

git status

Git dictionary

Repository	The location where changes made to your files are tracked. This location is located inside the project folder and is called *.git/*
Branch	A version of the project that you are working on
Commit	A snapshot of the project at a particular point in time
Master/Main Branch	The default branch of your project
Head	The current committed state of the project

Where to get more information

Git website: https://git-scm.com

Pro Git book: https://git-scm.com/book/en/v2

Git cheat sheet: https://education.github.com/git-cheat-sheet-education.pdf