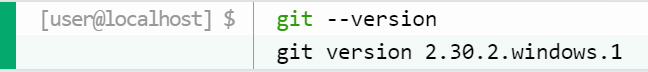
Git is a version control system.

Git helps you keep track of code changes.

Git is used to collaborate on code.

Example



What is Git?

It is used for:

* Tracking code changes
* Tracking who made changes
* Coding collaboration

What does git do:

* Manage projects with Repositories
* Clone a project to work on a local copy
* Control and track changes with Staging and Committing
* Branch and merge to allow for work on different parts and versions of a project
* Pull the latest version of the project to a local copy
* Push local updates to the main project

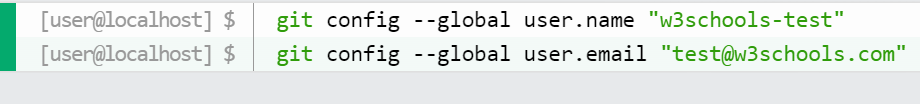
Working with Git

* Initialize Git on a folder, making it a Repository
* Git now creates a hidden folder to keep track of changes in that folder
* When a file is changed, added, or deleted, it is considered modified
* You select the modified files you want to Stage
* The staged files are committed, which prompts Git to store a permanent snapshot of the files
* Git allows you to see the full history of every commit
* Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit !

What is GitHub?

* GitHub makes tools that use Git
* GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018

If we already have the installation change the username and e-mail address to your own. You will probably also want to use this when registering to GitHub with the next command



Use global to set the username and e-mail for every repository on your computer.

If you want to set the username/e-mail for just the current repo, you can remove global.

mkdir makes a new directory.

cd changes the current working directory.

Now that we are in the correct directory. We can start by initializing Git!

Note: If you already have a folder/directory you would like to use for Git:

Navigate to it in command line, or open it in your file explorer, right-click and select "Git Bash here"

*INITIALIZE GIT*

Once you have navigated to the correct folder, you can initialized Git on that folder:



*GIT NEW FILES*

Create a new file using a text editor. Save it to the git folder and at the terminal collocate this:

Graphical user interface, application

Description automatically generated with medium confidence

And to check the status:

Graphical user interface, text, application

Description automatically generated

Files in git repository folder can be in one of 2 states:

* Tracked – files that Git knows about and are added to the repository
* Untracked – files that are in your working directory, but not added to the repository

When you first add files to an empty repository, they are all untracked. To get Git to track them, you need to stage them, or add them to the staging environment.

*GIT STAGING EVIROMENT*

One of the core functions of Git is the concepts of the Staging environment, and the commit.

Staged files are files that are ready to be committed to the repository you are working on.

For add a file to the Staging environment use:



So check the status:

Graphical user interface, text

Description automatically generated

We can also stage more than one file at a time.

It’s recommended to add a README.md file that describes the repository.

A basic external style sheet of css.

To add all files in the current directory to the staging environment:



Using **--all** instead of individual filenames will **stage** all changes (new, modified, and deleted) files.

The shorthand command for **git add –all** is **git add -A**

*GIT COMMIT*

Adding commits keep track of our progress and changes as we work. Git considers each commit change point or “save point”. It is a point in the project you can go back to if you find a bug, or want to make a change.

When we commit, we should always include a message.

By adding clear messages to each commit, it is easy for yourself (and others) to see what has changed and when.

Text

Description automatically generated

The **commit** command performs a commit, and the **-m “message”** adds a message.

The staging environment has been committed to our repo, with the message:

“First release of hello world!”.

Sometimes, when you make small changes, using the staging environment seems like a waste of time. It is possible to commit changes directly, skipping the staging environment. The **-a** option will automatically stage every changed, already tracked file.

If we make a change to a file that we already have, and then we check the status of our repository using the –short option, we will see the changes in a more compact way like this:

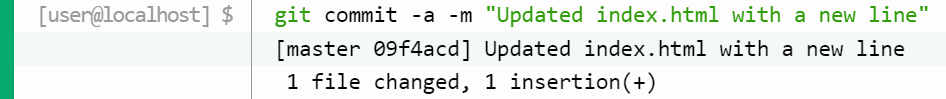
Graphical user interface, application

Description automatically generated with medium confidence

Note: Short status flags are:

* ?? - Untracked files
* A - Files added to stage
* M - Modified files
* D - Deleted files

Once we see our file modified, we proceed to commit it directly:



To view the history of commits for a repository, you can use the **long** command:

Text

Description automatically generated with medium confidence

*GIT BRANCHES*

In Git, a **branch** is a new/separate version of the main repository.

If we had a large project and are interested in updating the design of it, in Git, that would work like:

* With a new branch called new-design, edit the code directly without impacting the main branch
* EMERGENCY! There is an unrelated error somewhere else in the project that needs to be fixed ASAP!
* Create a new branch from the main project called small-error-fix
* Fix the unrelated error and merge the small-error-fix branch with the main branch
* You go back to the new-design branch, and finish the work there
* Merge the new-design branch with main (getting alerted to the small error fix that you were missing)

Branches allow to work on different parts of a project without impacting the main branch.

When the work is complete, a branch can be merged with the main project.

You can even switch between branches and work on different projects without them interfering with each other.

Branching in Git is very lightweight and fast!

New branch, imagine we are working in our local repository, and we do not want to disturb or possibly wreck the main project.

So, we create a new branch:



Now that created a new branch called “hello-world-images”

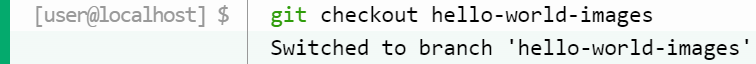
Let’s confirm that we have created a new branch:

Table

Description automatically generated with low confidence

We can see the new branch with the name “hello-world-images”, but the \* beside master specifies that we are currently on that branch.

**Checkout** is the command used to check out a branch. Moving us from the current branch, to the one specified at the end of the command:



Now we have moved our current workspace from the master branch to the new branch.

If one file is edited and add an image to the working folder and a line of code in the **.html** file.

Then, after changing the file and added a new file in the working directory (same directory as the main branch).

If then check the status of the current branch:

Graphical user interface, text, application

Description automatically generated

***BEST PRACTICES***

Skipping the staging environment is not generally recommended.

Skipping the stage step can sometimes make you include unwanted changes