**Lesson 1**

**Git Introduction**

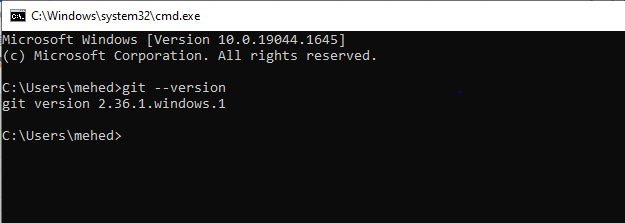
* Git is a version control system.
* Git use for
  + Tracking code changes
  + Tracking who made changes
  + Coding collaboration
* Git can
  + manage project with Repositories
  + Clone a project to work on a local copy
  + Control and track changes with **Staging** and **Committing**
  + **Branch** 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.
  + You can revert back to any previous commit.
  + Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!
* Why Git?
  + Over 70% of developers use Git!
  + Developers can work together from anywhere in the world.
  + Developers can see the full history of the project.
  + Developers can revert to earlier versions of a project.
* GitHub
  + Git is not the same as 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.

**Git Install**

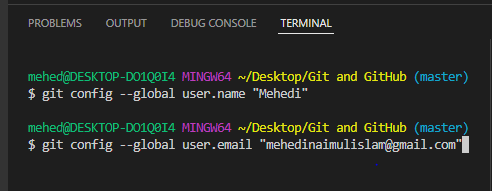
Download Git from Here (<https://git-scm.com/>)



**Using Git with Command Line**



**Configure Git**



Note: 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.*

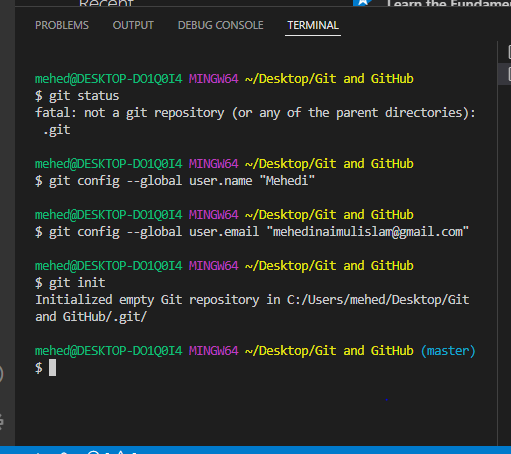
**Creating Git Folder**



* **mkdir** makes a new directory.
* **cd** changes the current working directory.

**Initialize Git**

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

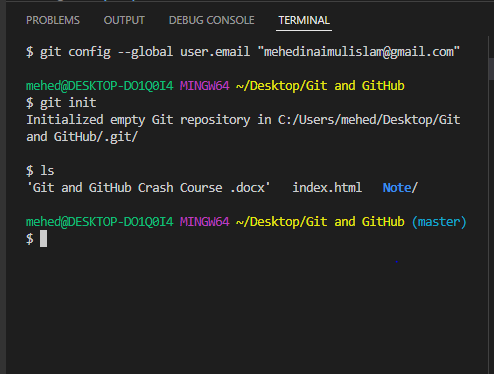


**Note:** Git now knows that it should watch the folder you initiated it on.

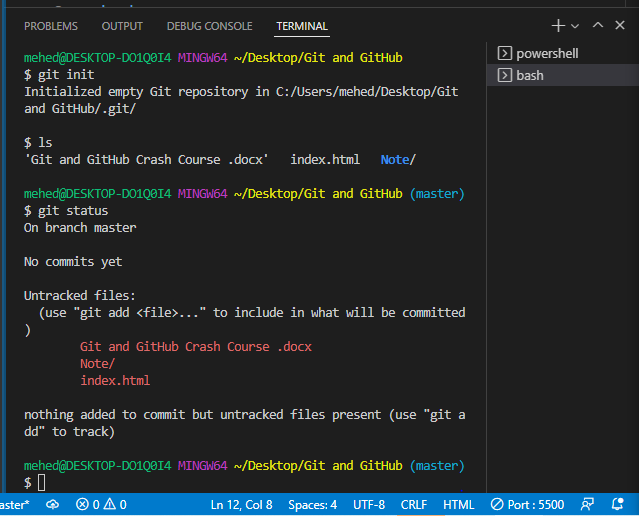
Git creates a hidden folder to keep track of changes.

**Git Adding New Files**

Create and save index.html file and let’s go back to the terminal and list the files in our current working directory:



Check git *status*



Files in your Git repository 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

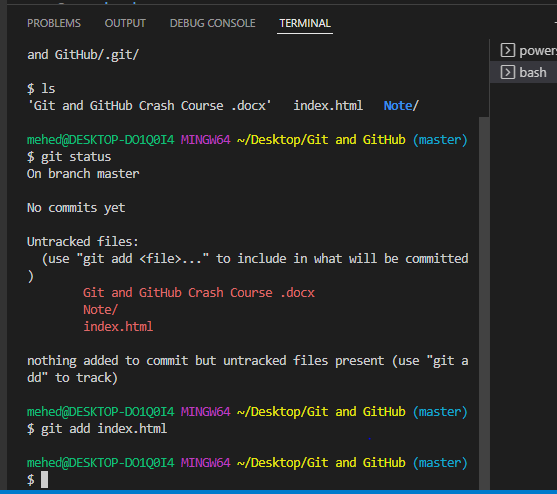
When you first add files to an empty repository, they are all untracked. To get Git to track them, you need to stage them, or add them to the staging environment.

**Git Staging Environment**

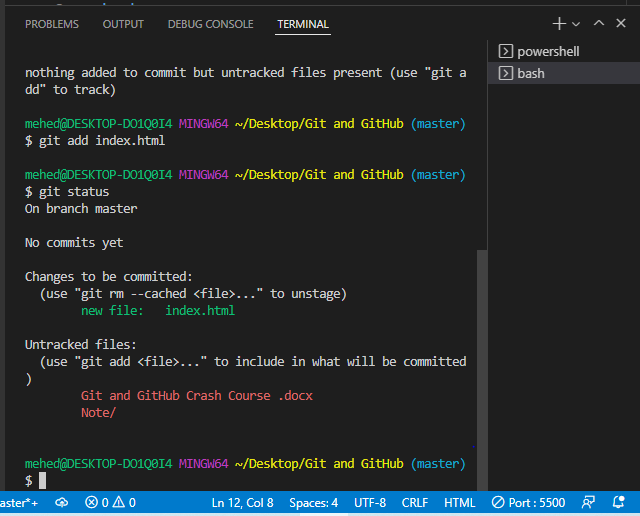
One of the core functions of Gits is the concepts fo the Staging Environment, and the Commit.

As you are working, you may be adding, editing and removing files. But whenever you hit a milestone or finish a part of the work, you should add the files to a Staging Environment.

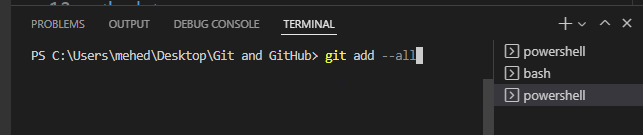
Staged files are files that are ready to be committed to the repository you are working on.



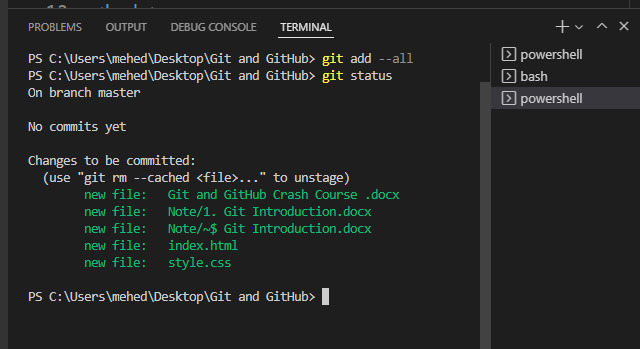
Lets’ check the status



**Git Add More than One File**



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



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

**Git Commit**

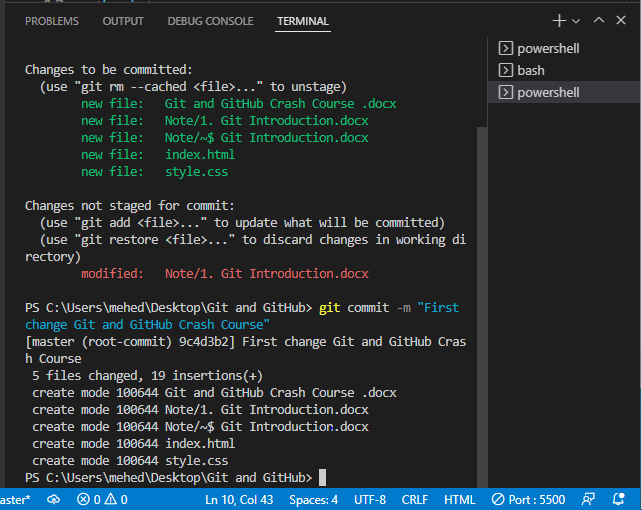
Since we have finished our work, we are ready move from *stage* to *commit* for our repo.

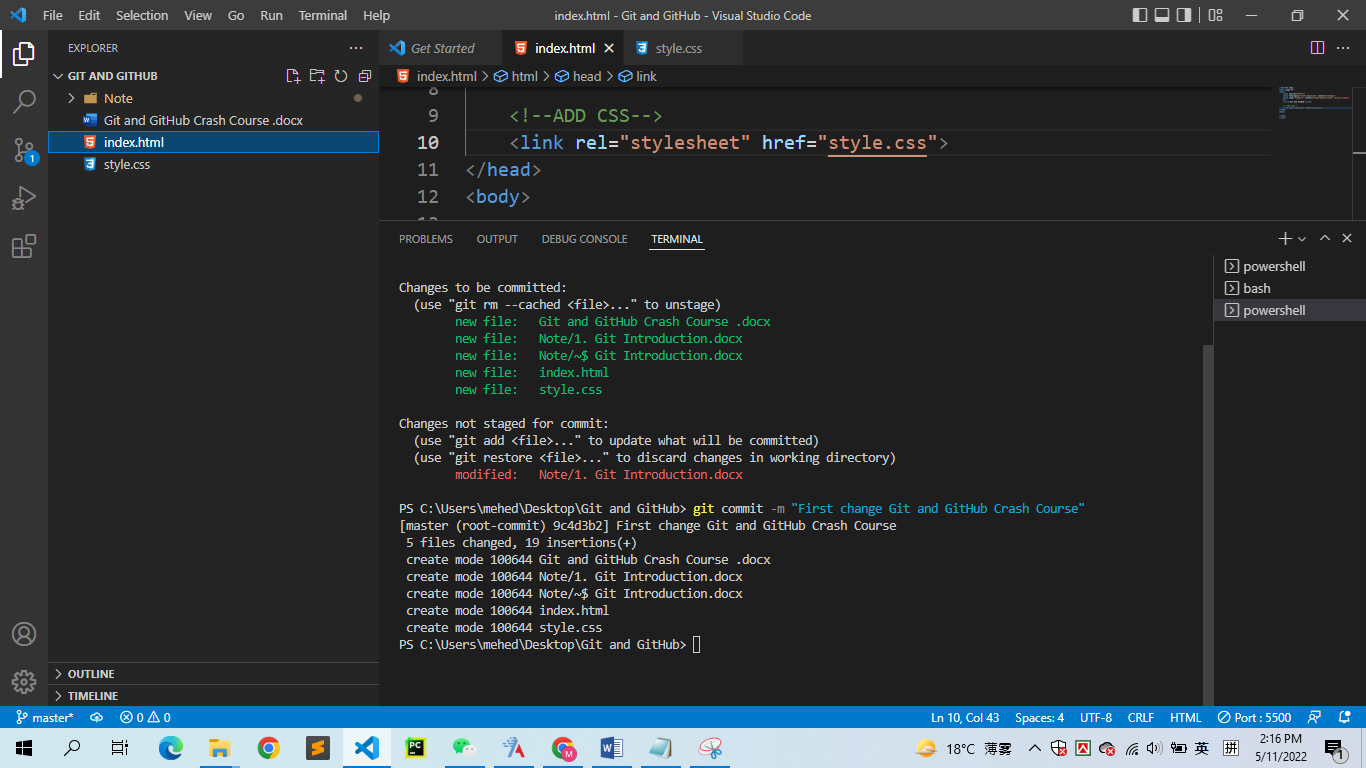
Adding commits keep track of our progress and changes as we work. Git considers each *commit* change point or “save point”. It is a point in the project you can go back to if you find a bug, or want to make a change.

When we *commit,* we should always include a message.

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.





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

**Git Commit without Stage**

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.

We, will use the --short option to see the changes in a more compact way:

