Git Introductory Tutorial

What is Git?

To put it simply, Git is a very powerful Version Control System. A VCS, as the name implies, is basically designed to help you keep track of what you are doing with your files. You can control what you add, what you omit, and go back and forth in different versions of your work. This is of course just "to put it simply"! To learn about the true power of Git, visit here: https://git-scm.com/book/en/v2/Getting-Started-Git-Basics, and read on!

How to setup Git?

This is probably the easiest part, just go to https://github.com/, sign up, and you will see very clear instruction to "Set Up Git" on the top of the home page: https://help.github.com/articles/set-up-git/
Follow the instructions based on your platform. Use the terminal for Git, not the GUI version, since there are certain limitations on the graphical one. The terminal version is much easier to work with in the long term. If you are using windows, it is better to use Git Bash, the specific terminal for Git, this is of course discussed in the installation guide on the website.

How to Authenticate?

Following the Git setup instructions, you see the part on the authentication methods. You can choose either one of these two authentication methods: HTTPS, SSH. They both serve the same purpose: to keep your projects safe, by securing your Git connection with GitHub. You can read detailed explanation about both methods on the website, but for now just choose one.
If you choose SSH, then you need to setup a SSH key on your computer, follow instructions here: https://help.github.com/articles/generating-ssh-keys/. Remember you have to setup a key for each system separately, meaning on each computer (or OS) you should have a unique SSH key connected with your GitHub account.
If you choose HTTPS, there is not much to do, just follow the instruction here: https://help.github.com/articles/caching-your-github-password-in-git/, if you want your Git to remember you and stop asking you the log-in info everytime.

How does it work?

Basically, what Git does is keeping snapshots of your whole project at different points, so you can later come back to them, or maybe merge some new features into old versions, etc. this is actually what version control is!

Each Git Project is consisted of three main sections: Working Directory, Staging Area, and Git Directory. The Git directory (.git folder) is where all the metadata, database and important files are stored by Git, this is basically the heart of Git.
The Working Directory is consisted of the single version of your project that you are working on currently (or in Git terminology: a checkout of your project). Whenever you checkout into a version of your project (we will talk about this in a moment), Git pulls out your related files from the database (which is located in Git directory) and puts it in your working directory. So to put it simply, working directory is actually the directory you can see while working on your project.
And last but not least, Staging Area. This is where the changes info about your project is stored, meaning the things you are going to commit into your new version. Staging area is a single file that stores data about what you have done since your last version, and keeps track of what to commit afterwards.
The basic Git workflow is like this:

1. You modify files in your Working Directory.
2. You stage the files, adding snapshots of them to your staging area.
3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git Directory, i.e. creating a new version.

If a particular version of a file is in the Git directory, it’s considered committed. If it has been modified and was added to the staging area, it is staged. And if it was changed since it was checked out but has not been staged, it is modified.

What is Branching? What is Checkout?

You can think of Git as a tree. When you start a Git project, you are on the master branch (let’s say the tree trunk), you can keep working there and modify, add, and commit and go on. But at some point you may want to create a new branch. Why? Let’s say you have a program that many people are using, an online game for example, but you want to start developing an update for it. What you can do is to create a new branch, let’s call it update1, and do your development on this update1 branch. So maybe while you are working on the branch, someone reports an important bug in your original version of the game, and you want to fix it right away. What you do is you just switch to your master branch (this is called checkout), and you fix the bug, and add, and commit. Then you come back to the update1 branch (i.e. checkout update1) and work on the rest of your update. When your update is complete, all you have to do is to merge the update1 branch with the master branch, and the new update will be added to the master branch, which already had the bug fixed.
Remember: when working with Git, there is something called HEAD, this is where you are. Where your HEAD is, is where your working directory is, to put it simply! So when I say move to another branch, I actually mean moving your HEAD to another branch. Thus what checkout does is moving your HEAD to another branch. So be careful where your HEAD is at all times!!!

How to stay sync with the online repository?

There are basically three things to know: Clone, Push, and Pull.

Clone is when you want to create a copy of a repository on your computer, this is when you want to start developing a project. It can be done via either SSH or HTTPS authentication.
Pull is when you want to get the newest changes from the online repository (called remote). For example many people may be working on a shared repository, and all of them want to have the newest updates developed by other people on the same repository. So they can each pull to update their working directory.
Push is the opposite of pull obviously! When you are ready to add your work from your local machine to the online repository (called remote), you push the data, for example if your update is ready and you want to send it online so other contributors can pull it.

Can we see an Example?

If you feel overwhelmed and confused, that's alright, things will get much more clear in a moment. So let's start doing some basic work with Git:

1. Go to GitHub homepage, sign in, and from the top right corner click on the plus sign (+) and then click New Repository. Provide a name, a description, make it public (you should pay for private, your choice!) and initialize with a README. Click create and you have your repository ready.
2. Now it is time to clone your repository into your computer. On the right side of your repository page find the URL bar. You can select HTTPS or SSH based on your previous decision (remember, you must setup SSH key if you want to use SSH authentication!). Copy the URL. Now open your terminal (or Git Bash in windows), go to the drive where you want to store your project (repository), and type:
   ```
   git clone <URL>
   ```
   That's it, you should see a folder by the name of your repository appear in the drive! This is called **"Cloning"**! you can clone any repository from anyone as long as it is public.
3. Now type:
   ```
   git status
   ```
   You will see that everything is updated. Now add a file to your project folder, change the contents, and save. Type git status again and see the "Modified" alert!
4. Now let's add the changed files to the staging area:
   ```
   git add --all
   ```
   or you could add files one by one:
   ```
   git add <filename>
   ```
5. Check the status again, see how things are green now!
6. Go for committing now, remember, when you commit, you create a new checkpoint in your project, meaning something is done, so try to commit only when something has been added that is actually WORKING!
   ```
   git commit -m '<your commit message>'
   ```
   Note that commit command only commits the changes on the staging area. There is a way to do staging and commit all at once, and this is it:
   ```
   git commit -a -m '<your commit message>'
   ```

7. Done! You have successfully committed and are ready to start working on your files again, and then repeat all the steps from the top when you are ready for your next commit! You can use Git log to see history:
   ```
   git log
   ```
   Type :q to exit!
8. You can use push and pull commands to push and pull as was described before (do this only after you have a successful commit!):
   ```
   git pull <remote> --all
   git push <remote> --all
   ```
   Default remote name (your online repository), is set to origin.
   Note that you must have write access to a repository in order to be able to push!
   Ignore the --all command if you want to push/pull a specific branch to another branch, here is how:
   ```
   git pull <remote> <branch name on remote>:<branch name on local>
   git push <remote> <branch name on local>:<branch name on remote>
   ```
9. If you want to branch here is how to create a new branch:
   ```
   git branch <branch name>
   ```
   This create the branch by the name given. Now note that your HEAD is still at the master branch (or what ever branch you were on when branching). To move your HEAD to the new branch, you need to checkout to that branch:
   ```
   git checkout <branch name>
   ```
   You can checkout back and forth between different branches (and the master branch of course), and if you are ever lost, use this to view your local branches:
   ```
   git branch
   ```
   And this if you want to see the branches on the remote:
   ```
   git branch --all
   ```
   There is also a way to create a branch and checkout to it at the same time:
   ```
   git checkout -b <branch name>
   ```
   And when you want to merge a branch back into another branch, go to the branch that is your to keep branch, meaning the branch you want to merge the other one into, and type:
   ```
   git merge <the branch to be merged name>
   ```
   You may get an Conflict error, meaning Git could not solve the merge, in which case you have to open the files mentioned in the error by yourself, and then add and commit (see here: https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging)
   And when you are done with a branch, delete it this way:
   ```
   git branch -d <branch name>
   ```
10. **Last but definitely not the least, this is the most important part. If you want help with a command, while connected to the Internet type:**
    ```
    git help <command name>
    ```
    And you will be taken to the Git help webpage for that command. This is technically how you will learn Git. The best command ever is undoubtedly this "help". So enjoy!

Git Etiquette

Just a few pointers before you wield the power of Git. With great power comes great responsibility.

- Give verbose commit messages. Not only will it help other collaborators on the repository understand what changes you committed, but also it will help you keep a track of changes even if you forget it.
- Before committing your changes, make sure your code works. Before pushing your commits, make sure you properly merge with the repository and your code works after the merge. Do not push broken code.
- When you are collaborating with people, be careful when you merge your code with other people's changes. Make sure you do not discard someone else's change while pushing your own commits.
- If you think the merge was incorrect and messed things up (say created a new head or created conflicts), first make sure you keep a backup of your changed files before doing anything drastic to the repository. You do not want to revert the repository and lose all your changes. Do not force your commits unless you are absolutely sure that is what you want to do.
- If someone else has broken your code through their commits, coordinate with them to figure out the fix. There is a fix for almost everything.
- Keep pulling changes and merging frequently so that you do not have to deal with a massive merge with a million changes that other people have pushed to the repository.
- If the number of collaborators of the repository is small, one easy way to avoid issues is to not work on the same file simultaneously. This might not be always possible but it lets you avoid painful merges.

Want to know more?

Here is the source used for writing this document, and where you can find anything and everything about Git. Perfect place to continue reading and to look at whenever you have a problem:
https://git-scm.com/book/en/v2/
You can also always google any error that pops up, and you will most likely find the answer online.

Good Luck Everyone!