

Git

A Crash Course on Version Control and Git & GitHub



Ressources

- <https://towardsdatascience.com/a-crash-course-on-version-control-and-git-github-5d04e7933070>
- <https://git-scm.com/>
- <https://www.fullstackpython.com/git.html>
- <https://ohshitgit.com/>

Introduction

Git and GitHub is the de facto version control tool every programmer, data scientist, machine learning engineer, web developer full-stack developer, etc. uses.

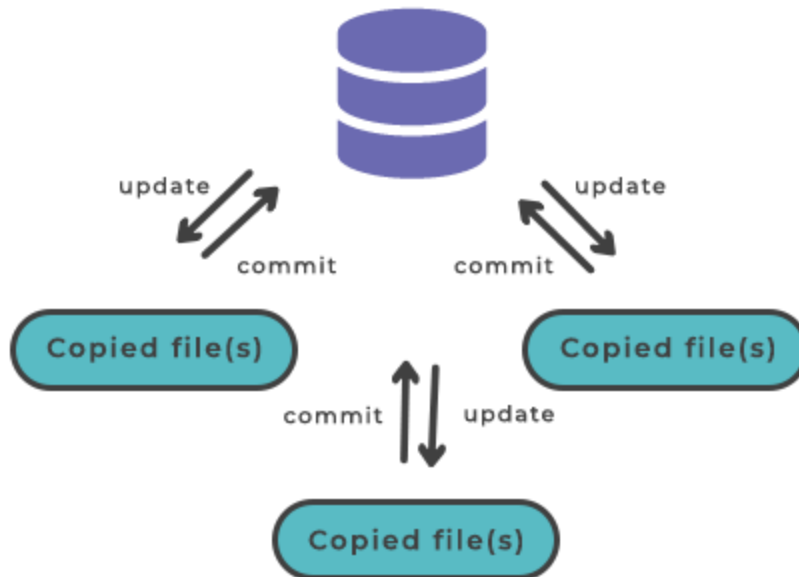
What can one do GitHub?

On GitHub, one can:

1. share code online, and continually update it as you improve your program, without restriction to your local machine,
2. create a rocking portfolio for your employers to marvel at your capabilities.
3. It's also a place filled with resources for learning, solutions, and notes to online courses, etc.
4. contribute to open source projects and build your skills.
5. collaborate with other developers on cool projects
6. fork cool projects and hack at it for fun
7. etc.

Version control system

Centralized Version Control



What is VCS ?

A version control system (VCS) is as intuitive as the name suggests, it's basically a collection of tools that help you **track changes** to your source code or files. It maintains the **history of changes** and facilitates **collaboration** with other developers.

You can think of what VCS does as taking **snapshots of changes** in your code and files, and stores information as messages which explains why the change was made and who made it.

Why use VCS

That said, VCS is incredibly helpful for three main reasons.

1. **Snapshot of history of your work.** — If you want to look back at old snapshots of your project to see why changes were made, simultaneously working as backup storage like Google Drive. It also allows you to search up your code or project from anywhere as long as you can sign in to your GitHub account.

2. **Allows for collaboration and handles conflicts** — Collaboration is huge when programming. One of the main hiccups when collaborating is conflicts, when two people are working on the same file, and upload their respective changes, only one change will be made and the other will be lost. Issues like this are handled by Git, and you can view all the issues and pull requests on GitHub's clean UI.
3. **Branching for experimentation** — There are times when you want to experiment on something new, so you'll want to make a copy of your code, change something, without affecting your original source code. This is called branching, and it's the same as apps releasing beta programs to the public, which are branches of the main program

Basic VC vocabulary

- **Repository** — projects folder or directory, all VC files are located here
- **Commit** — save edits and changes made (snapshots of files)
- **Push** — updating repository with your edits
- **Pull** — updating your local version of the repo to the current version
- **Staging** — preparing a file for commit
- **Branch** — Same file with two simultaneous copies, create a branch to make edits which are not shared with main repo yet
- **Merge** — where a branch merges back to the main repo, independent edits of the same file incorporated into a unified file
- **Conflict** — when two people made edits to the same line of code, conflict happens, either keep one or the other
- **Clone (only on the local machine)** — making a copy of repo with all of the tracked changes
- **Fork (remains in GitHub)** — a copy of repo of another person, edits are logged on your repo, not theirs





Git flows

So, a quick rundown of a basic flow of VCS is — files are hosted in repositories shared online with other coders. You clone your repo to have a local copy to edit it. After making changes, you stage the file and commit it. Then, you push to commit to the shared repo where the new file is online with messages explaining what changed, why, and by whom.





Git commands

There are a couple of basic git commands you should know by heart if you're to be a master programmer one day.

Copy of Git basic commands

 Title	 Git task	 Notes	 Git commands
<u>Untitled</u>	<u>Tell Git who you are</u>	Configure the author name and email address to be used with your commits. Note that Git <u>strips some characters</u> (for example trailing periods) from <code>user.name</code> .	<pre>git config --global user.name "Sam Smith" git config -- global user.email sam@example.com</pre>
<u>Untitled</u>	<u>Create a new local repository</u>		<pre>git init</pre>
<u>Untitled</u>	<u>Check out a repository</u>	Create a working copy of a local repository:	<pre>git clone /path/to/repository</pre>
<u>Untitled</u>	For a remote server, use:	<pre>git clone username@host:/path/to/repository</pre>	
<u>Untitled</u>	<u>Add files</u>	Add one or more files to staging (index):	<pre>git add <filename> git add *</pre>
<u>Untitled</u>	<u>Commit</u>	Commit changes to head (but not yet to the remote repository):	<pre>git commit -m "Commit message"</pre>
<u>Untitled</u>	Commit any files you've added with <code>git add</code> , and also commit any files you've changed since then:	<pre>git commit -a</pre>	
<u>Untitled</u>	<u>Push</u>	Send changes to the master branch of your remote repository:	<pre>git push origin master</pre>

Aa Title	≡ Git task	≡ Notes	≡ Git commands
<u>Untitled</u>	Status	List the files you've changed and those you still need to add or commit:	<code>git status</code>
<u>Untitled</u>	Connect to a remote repository	If you haven't connected your local repository to a remote server, add the server to be able to push to it:	<code>git remote add origin <server></code>
<u>Untitled</u>	List all currently configured remote repositories:	<code>git remote -v</code>	
<u>Untitled</u>	Branches	Create a new branch and switch to it:	<code>git checkout -b <branchname></code>
<u>Untitled</u>	Switch from one branch to another:	<code>git checkout <branchname></code>	
<u>Untitled</u>	List all the branches in your repo, and also tell you what branch you're currently in:	<code>git branch</code>	
<u>Untitled</u>	Delete the feature branch:	<code>git branch -d <branchname></code>	
<u>Untitled</u>	Push the branch to your remote repository, so others can use it:	<code>git push origin <branchname></code>	
<u>Untitled</u>	Push all branches to your remote repository:	<code>git push --all origin</code>	
<u>Untitled</u>	Delete a branch on your remote repository:	<code>git push origin :<branchname></code>	
<u>Untitled</u>	Update from the remote repository	Fetch and merge changes on the remote server to your working directory:	<code>git pull</code>
<u>Untitled</u>	To merge a different branch into your active branch:	<code>git merge <branchname></code>	
<u>Untitled</u>	View all the merge conflicts:View the conflicts against the base file: Preview changes, before merging:	<code>git diff</code> <code>git diff --base <filename></code> <code>git diff <sourcebranch></code> <code><targetbranch></code>	

 Title	 Git task	 Notes	 Git commands
<u>Untitled</u>	After you have manually resolved any conflicts, you mark the changed file:	<code>git add <filename></code>	
<u>Untitled</u>	Tags	You can use tagging to mark a significant changeset, such as a release:	<code>git tag 1.0.0 <commitID></code>
<u>Untitled</u>	CommitID is the leading characters of the changeset ID, up to 10, but must be unique. Get the ID using:	<code>git log</code>	
<u>Untitled</u>	Push all tags to remote repository:	<code>git push --tags origin</code>	
<u>Untitled</u>	<u>Undo local changes</u>	If you mess up, you can replace the changes in your working tree with the last content in head: Changes already added to the index, as well as new files, will be kept.	<code>git checkout -- <filename></code>
<u>Untitled</u>	Instead, to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it, do this:	<code>git fetch origin git reset --hard origin/master</code>	
<u>Untitled</u>	Search	Search the working directory for <code>foo()</code> :	<code>git grep "foo()"</code>

Best Practices For VCS

When you're just starting out using VCS, you may be confused when to commit your code, or how best to use VCS, so here are some helpful tips.

1. **Make purposeful commits** — Your commits should be purposeful and not trivial, it should be making substantial changes to your code. For example, typing one new line of code and committing it isn't effective and can impede your productivity.

2. **Informative messages on commits** — Your commits should have messages that fully explain why a certain change was made so that in the future you can go back and refer to it. It also helps others collaborate to understand the changes.
3. **Pull and push often** — Pulling is important when working on a public repo so that you're always up to date on changes, and you should be pushing often as well to have your changes updated on GitHub.
4. More tips [here](#).