Dylan Lawson

Tech With Tim

GitHub

18th May, 2020

GitHub Version Control | Tutorial for Beginners

* YouTube Link: <https://www.youtube.com/watch?v=PWqS4NBhEY8>
* Please see GitHub\_Notes.odt for 1st hand information and troubleshooting
* You can follow the numbered headlines to know how to create a local repository and upload it to GitHub
* All other headers contain helpful information that do not need to be followed in a specific sequence

**Common Commands:**

* **cd** - allows you to change into a different directory
* **ls** - lists the files/directories within a directory
* **mkdir [directory name]** - allows you to create a directory
* **touch [file name.extension]**- allows you to create a file like "text.txt"
* **history** - shows you the history of commands you used
* **clear** - clears the command terminal of any commands or information
* **git show [file name]** - shows the content in a file that you created
* **git status** - shows the working tree status
* This command will show you what files you have yet to add to the “master” branch
* **git branch** - shows all branches created along with the master branch
* **git branch -av** - displays what branches exist in your repository both locally and remotely
* **git branch -m [branch to be renamed] [new name for branch]** - renames a specifc branch

**1) Initializing a Local Repository**

* Open Git Bash
* Navigate to the directory that you would like to add to GitHub with “cd [directoryName]”
* Type **git init**
* This command will turn your folder into a local repository
* This will also create a **.git** folder for the folder that you just initialized
* This **.git** folder will allow us to run Git specific commands
* Use **git init** inside of a folder that contains the files you want to push to a remote respository
* If the **git init** command is used outside of the folder that contains all the files you want to push to a remote repository, the **.git** folder will keep track of all unrelated folders/files that is on the same heirarchy of the folder that contains all the files you want to push to a remote repository

**2) Adding files to your Local Repository**

* Once you have created new files in the newly created repository, you have to manually add them into the repository. **You must have files in your repository to in order to push it.**
* Still in the same directory, type **git status**
* This command will show you what files you have yet to add to the “master” branch
* Still in the same directory, type **git add .**
* This adds all of the newly created files into the repository into a staging area
* If you type **git status** again, the command will show you the new files that have been added to the repository
* You can type: **git add [fileName.txt]** to add a singular file to the staging area
* Keep in mind that at this point, you have only added the files and not committed them.
* Committing will make a permanent change in the repository

**3) Committing your Changes**

* The next command you can type is: **git commit -m “[Your reason for committing your changes]”**
* A good first commit would be **git commit -m "Initial Commit - Adding all files to local/remote repository"**

**Removing files from Local Repository**

* Still in the same directory of Git Bash, type: **git rm –cached [fileName.txt]**
* This will delete the specified file named in the brackets above

**Branching from Your Main (Master) Repository**

* Branches basically make a copy of your Local Repository (Master Branch).
* This allows you to make changes and edits on the Branch so that you won’t mess up the Master Branch
* Once you are ready, type: **git branch [branch name]**
* You’ll notice that Git Bash still says that you are in the “master” branch
* To switch or select your new branch, type **git checkout [branch name]**

**Merging Branches Together**

* Once you are done making changes in the child branch, you can merge the changes that you made back into the master branch.
* To do this, first take note of what branch you are in.
* If you are in the local branch, type: **git merge master**
* If you are in the master branch, type: **git merge [branch name]**
* This will take the changes from either branch and overwrite either branch depending on which branch has more updated files/code
* **git show [file name]** - shows the content in a file that you created
* **git branch** - shows all branches created along with the master branch
* **git branch -av** - displays what branches exist in your repository both locally and remotely

**4) Creating a Remote Repository**

* Creating a Remote Repository on [https://github.com](https://github.com/) will enable you to have a repository on the cloud
* Log on to GitHub – Create New Repository – Name it – Give a description (Optional) – Private/Public – Initialize with a README.md file (Optional)
* If you opt to add a README.md file, a **main** branch will be created
* If this happens you'll need to use the **main** branch and not the **master** branch for your remote repository
* So, when you use commands like **git push -u origin master**, be aware that this will create another branch called **master**
* Using **git push -u origin main** may cause errors
* If this happens, you can do 1 of 2 things:
* 1) You can opt out of creating a README.md file when you initially create your repo in GitHub
* Afterwards, you can manually create a README.md file in your local repo and push it to your remote repository
* 2) After creating your repo with a README.md file, you can set the default branch of your repository from **main** to **master**
* This can be done by going to Settings - Default Branch - Switch main to master
* This will allow you to use **git push -u origin master** and upload to your remote repository
* You can then delete the **main** branch from your new repository

**5) Upload to your Remote (Online) Repository**

* Here, you are essentially creating a Remote Repository on your local machine.
* Once you’ve created your remote repository on GitHub, it will show you the commands needed to push information into the remote repository
* To start, go to the directory that you would like to push into the cloud and create the local repository by typing: **git remote add [repo name] + [repo hyperlink]**
* An example of this is: **git remote add origin** [**https://github.com/username/repoName.git**](https://github.com/username/repoName.git)
* "origin" is your repo name. You can type a different name for your repo instead of "origin"
* Then, we push that remote repository into the cloud by typing: **git push -u [repo name] master**
* An example is: **git push -u origin master**
* You can also push branches by typing: **git push -u [repo name] + [branch name]**
* If you get an error here, look at your GitHub\_Notes\_Updated document. Your repo may have conflicting README.md and gitignore files that are stopping you from uploading your local repo
* If you receive a "There isn't anything to compare. Nothing to compare, branches are entirely different commit histories", review your GitHub\_Notes\_Updated document. There are 4 commands you can use to resolve this situation
* This happens when you opt to add a README.md file and use **git push -u origin master** in Git Bash
* Review the 2 steps in the **4) Creating a Remote Repository** section to solve this issue

**Cloning a Remote (Online) Repository**

* Cloning is essentially pulling a repository off from GitHub onto your local machine
* To start, make a directory for where you want to house the new repository with the **mkdir** command in Git Bash
* This just makes a folder in whatever directory you are in
* Go ahead and go into the newly created folder by typing: **cd [newFolder]**
* Then go to GitHub, find the repository that you want, click “Clone or Download”, copy the link, go back to Git Bash and type: **git init**
* Remember, this command allows you to utilize git commands
* Now, you type: **git pull [copied link]**
* Now you have the repository on your local machine

**Pulling information from a remote repo when both your local and remote repos are out of sync (diverged)**

* Review your Git - Resolving Out-of-sync Repos conversation in your ChatGPT account
* Sceanrio: You've made changes to certain files that are more up to date than you remote repo; however, your remote repo has a file that is more up to date than your local repo
* Local Repo - repo on your computer
* Remove Repo - repo on GitHub
* To handle this situation, you can follow the steps below to ensure that both repos are back in sync:
* 1) Use the command **git fetch origin** in Git Bash
* This will tell you:
* If your local or remote branch is ahead/behind/diverged
* If you're only **ahead** → no pull needed, remote is just behind.
* If you're **behind** → git pull will bring changes into your local repo.
* If you're **diverged** → you may want to resolve manually (merge or rebase).
* 2) If your local and remote repos are diverged, you can then run **git pull origin master** in Git Bash (or main if main is your default branch)
* Your local repo should stay exactly the same
* No files on your local repo should be updated besides the file that is behind compared to your remote repo
* For example, let's say a change to your README.md file in GitHub. No other files were changed in GitHub
* On you computer you may have made many code changes, but didn't touch the local README.md file
* If you used the git pull origin master command, the only file that would be updated on your local computer would be the README.md file.
* The files that are behind on your remote repo would not overwrite the more up-to-date files on your local computer.
* 3) Once you run the git pull origin master command, a Vim text editor will appear in Git Bash
* Vim will ask if you want to add a comment stating why the remote and local repos are being merged together
* When ready, you can:
* 1) Press Esc to enter command mode
* 2) Enter :wq to "write" and "quit"
* This will:
* 1) Git will finish the merge.
* 2) The merge commit will be created.
* 3) Your local branch will now include both your changes from your local repo and the remote repo changes.