#### **Git and GitHub:**

Basic commands

* repo -> repository
* clone -> bring a repo down from the internet (remote repository like Github) to your local machine
* add -> track your files and changes with Git
* commit -> save your changes into Git
* push -> push your changes to your remote repo on Github (or another website)
* pull -> pull changes down from the remote repo to your local machine
* status -> check to see which files are being tracked or need to be commited
* init -> use this command inside of your project to turn it into a Git repository and start using Git with that codebase

### A quick aside: git and GitHub are ***not*** the same thing. Git is an open-source, version control tool created in 2005 by developers working on the Linux operating system; GitHub is a company founded in 2008 that makes tools which integrate with git. You do not need GitHub to use git, but you cannot use GitHub without using git. There are many other alternatives to GitHub, such as GitLab, BitBucket, and “host-your-own” solutions such as gogs and gittea. All of these are referred to in git-speak as “remotes”, and all are completely optional. You do not need to use a remote to use git, but it will make sharing your code with others easier.

## **Step 1: Create a local git repository**

### When creating a new project on your local machine using git, you'll first create a new [repository](https://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository) (or often, 'repo', for short).

### To use git we'll be using the terminal. If you don't have much experience with the terminal and basic commands, [check out this tutorial](https://ubuntu.com/tutorials/command-line-for-beginners#1-overview) (If you don’t want/ need a short history lesson, skip to step three.)

### To begin, open up a terminal and move to where you want to place the project on your local machine using the cd (change directory) command. For example, if you have a 'projects' folder on your desktop, you'd do something like:

### mnelson:Desktop mnelson$ cd ~/Desktop

### mnelson:Desktop mnelson$ mkdir myproject

### mnelson:Desktop mnelson$ cd myproject/

### [view raw](https://gist.github.com/cubeton/67a84eb876984f0b5785/raw/d4560016d742865c1fd68d97fcff1feb557d5e19/terminalcd.md)

### [terminalcd.md](https://gist.github.com/cubeton/67a84eb876984f0b5785#file-terminalcd-md) hosted with ❤ by [GitHub](https://github.com/)

### To initialize a git repository in the root of the folder, run the [git init](http://git-scm.com/docs/git-init) command:

### mnelson:myproject mnelson$ git init

### Initialized empty Git repository in /Users/mnelson/Desktop/myproject/.git/

### [view raw](https://gist.github.com/cubeton/89793ba1bc947f64658e/raw/f3dba1dd72fda5eeb98b761338aedfc310d29d54/gitinit.md)

### [gitinit.md](https://gist.github.com/cubeton/89793ba1bc947f64658e#file-gitinit-md) hosted with ❤ by [GitHub](https://github.com/)

## **Step 2: Add a new file to the repo**

### Go ahead and add a new file to the project, using any text editor you like or running a [touch](http://linux.die.net/man/1/touch) command. `touch newfile.txt` just creates and saves a blank file named newfile.txt.

### Once you've added or modified files in a folder containing a git repo, git will notice that the file exists inside the repo. But, git won't track the file unless you explicitly tell it to. Git only saves/manages changes to files that it *tracks*, so we’ll need to send a command to confirm that yes, we want git to track our new file.

### mnelson:myproject mnelson$ touch mnelson.txt

### mnelson:myproject mnelson$ ls

### mnelson.txt

### [view raw](https://gist.github.com/cubeton/2d8f224bede4c2dde86b/raw/b865e27cc4715b3a3a4a5839e77ab232ff1b31f9/addfile.md)

### [addfile.md](https://gist.github.com/cubeton/2d8f224bede4c2dde86b#file-addfile-md) hosted with ❤ by [GitHub](https://github.com/)

### After creating the new file, you can use the [git status](http://git-scm.com/docs/git-status) command to see which files git knows exist.

### mnelson:myproject mnelson$ git status

### On branch master

### 

### Initial commit

### 

### Untracked files:

### (use "git add <file>..." to include in what will be committed)

### 

### mnelson.txt

### 

### nothing added to commit but untracked files present (use "git add" to track)

### [view raw](https://gist.github.com/cubeton/02e849bbffcbea1e9a61/raw/71c93139666a8a4e06795f53c9aec5db95e6019a/gitstatus.md)

### [gitstatus.md](https://gist.github.com/cubeton/02e849bbffcbea1e9a61#file-gitstatus-md) hosted with ❤ by [GitHub](https://github.com/)

### What this basically says is, "Hey, we noticed you created a new file called mnelson.txt, but unless you use the 'git add' command we aren't going to do anything with it."

#### **An interlude: The staging environment, the commit, and you**

### One of the most confusing parts when you're first learning git is the concept of the staging environment and how it relates to a commit.

### A [commit](https://docs.github.com/en/free-pro-team@latest/github/getting-started-with-github/github-glossary#:~:text=the%20repository%20owner.-,commit,who%20made%20them%20and%20when.) is a record of what changes you have made since the last time you made a commit. Essentially, you make changes to your repo (for example, adding a file or modifying one) and then tell git to put those changes into a commit.

### Commits make up the essence of your project and allow you to jump to the state of a project at any other commit.

### So, how do you tell git which files to put into a commit? This is where the [staging environment or index](https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository) come in. As seen in Step 2, when you make changes to your repo, git notices that a file has changed but won't do anything with it (like adding it in a commit).

### To add a file to a commit, you first need to add it to the staging environment. To do this, you can use the [git add](http://git-scm.com/docs/git-add) <filename> command (see Step 3 below).

### Once you've used the git add command to add all the files you want to the staging environment, you can then tell git to package them into a commit using the [git commit](http://git-scm.com/docs/git-commit) command.

### Note: The staging environment, also called 'staging', is the new preferred term for this, but you can also see it referred to as the 'index'.

## **Step 3: Add a file to the staging environment**

### Add a file to the staging environment using the git add command.

### If you rerun the git status command, you'll see that git has added the file to the staging environment (notice the "Changes to be committed" line).

### mnelson:myproject mnelson$ git status

### On branch master

### 

### Initial commit

### 

### Changes to be committed:

### (use "git rm --cached <file>..." to unstage)

### 

### new file: mnelson.txt

### [view raw](https://gist.github.com/cubeton/28f7bea3b232f67e031c/raw/875157cd78d75c23f3f0e29bf0c97989e3d52937/addtostaging.md)

### [addtostaging.md](https://gist.github.com/cubeton/28f7bea3b232f67e031c#file-addtostaging-md) hosted with ❤ by [GitHub](https://github.com/)

### To reiterate, the file has **not** yet been added to a commit, but it's about to be.

## **Step 4: Create a commit**

### It's time to create your first commit!

### Run the command git commit -m "Your message about the commit"

### mnelson:myproject mnelson$ git commit -m "This is my first commit!"

### [master (root-commit) b345d9a] This is my first commit!

### 1 file changed, 1 insertion(+)

### create mode 100644 mnelson.txt

### [view raw](https://gist.github.com/cubeton/1068d965d147b4039e4d/raw/5c3262c3f6e3c28328ba57ea33c512dbab149fcf/commit.md)

### [commit.md](https://gist.github.com/cubeton/1068d965d147b4039e4d#file-commit-md) hosted with ❤ by [GitHub](https://github.com/)

### The message at the end of the commit should be something related to what the commit contains - maybe it's a new feature, maybe it's a bug fix, maybe it's just fixing a typo. Don't put a message like "asdfadsf" or "foobar". That makes the other people who see your commit sad. Very, very, sad. Commits live forever in a repository (technically you *can* delete them if you really, really need to but it’s messy), so if you leave a clear explanation of your changes it can be extremely helpful for future programmers (perhaps future you!) who are trying to figure out why some change was made years later.

## **Step 5: Create a new branch**

### Now that you've made a new commit, let's try something a little more advanced.

### Say you want to make a new feature but are worried about making changes to the main project while developing the feature. This is where [git branches](https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell) come in.

### Branches allow you to move back and forth between 'states' of a project. Official git docs describe branches this way: ‘A branch in Git is simply a lightweight movable pointer to one of these commits.’ For instance, if you want to add a new page to your website you can create a new branch just for that page without affecting the main part of the project. Once you're done with the page, you can [merge](https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging) your changes from your branch into the primary branch. When you create a new branch, Git keeps track of which commit your branch 'branched' off of, so it knows the history behind all the files.

### Let's say you are on the primary branch and want to create a new branch to develop your web page. Here's what you'll do: Run [git checkout -b <my branch name>](http://git-scm.com/docs/git-checkout). This command will automatically create a new branch and then 'check you out' on it, meaning git will move you to that branch, off of the primary branch.

### After running the above command, you can use the [git branch](http://git-scm.com/docs/git-branch) command to confirm that your branch was created:

### mnelson:myproject mnelson$ git branch

### master

### \* my-new-branch

### [view raw](https://gist.github.com/cubeton/fa25a25f322a2cd5f405/raw/81033788d288adeffe260bd724ab2699b29e3e35/gitbranch.md)

### [gitbranch.md](https://gist.github.com/cubeton/fa25a25f322a2cd5f405#file-gitbranch-md) hosted with ❤ by [GitHub](https://github.com/)

### The branch name with the asterisk next to it indicates which branch you're on at that given time.

#### **A note on branch names**

### By default, every git repository’s first branch is named `master` (and is typically used as the primary branch in the project). As part of the tech industry’s general anti-racism work, some groups have begun to use alternate names for the default branch (we are using “primary” in this tutorial, for example). In other documentation and discussions, you may see “master”, or other terms, used to refer to the primary branch. Regardless of the name, just keep in mind that nearly every repository has a primary branch that can be thought of as the official version of the repository. If it’s a website, then the primary branch is the version that users see. If it’s an application, then the primary branch is the version that users download. This isn’t *technically* necessary (git doesn’t treat any branches differently from other branches), but it’s how git is traditionally used in a project.

### If you are curious about the decision to use different default branch names, GitHub has an explanation of *their* change here: <https://github.com/github/renaming>

### Now, if you switch back to the primary branch and make some more commits, your new branch won't see any of those changes until you [merge](http://git-scm.com/docs/git-merge) those changes onto your new branch.

## **Step 6: Create a new repository on GitHub**

### If you only want to keep track of your code locally, you don't need to use GitHub. But if you want to work with a team, you can use GitHub to collaboratively modify the project's code.

### To create a new repo on GitHub, log in and go to the GitHub home page. You can find the “New repository” option under the “+” sign next to your profile picture, in the top right corner of the navbar:

### Git_1

### 

### After clicking the button, GitHub will ask you to name your repo and provide a brief description:

### Git_2

### When you're done filling out the information, press the 'Create repository' button to make your new repo.

### GitHub will ask if you want to create a new repo from scratch or if you want to add a repo you have created locally. In this case, since we've already created a new repo locally, we want to push that onto GitHub so follow the '....or push an existing repository from the command line' section:

### **mnelson:myproject mnelson$ git remote add origin https://github.com/cubeton/mynewrepository.git**

### **mnelson:myproject mnelson$ git push -u origin master**

### **Counting objects: 3, done.**

### **Writing objects: 100% (3/3), 263 bytes | 0 bytes/s, done.**

### **Total 3 (delta 0), reused 0 (delta 0)**

### **To https://github.com/cubeton/mynewrepository.git**

### **\* [new branch] master -> master**

### **Branch master set up to track remote branch master from origin.**

### [view raw](https://gist.github.com/cubeton/3a2616c44e35ca68a6b0/raw/41e5758cfdbd7db8a1659c1adaba9346680097f9/addgithub.md)

### [addgithub.md](https://gist.github.com/cubeton/3a2616c44e35ca68a6b0#file-addgithub-md) hosted with ❤ by [GitHub](https://github.com/)

### (You'll want to change the URL in the first command line to what GitHub lists in this section since your GitHub username and repo name are different.)

## **Step 7: Push a branch to GitHub**

### Now we'll push the commit in your branch to your new GitHub repo. This allows other people to see the changes you've made. If they're approved by the repository's owner, the changes can then be merged into the primary branch.

### To push changes onto a new branch on GitHub, you'll want to run [git push](http://git-scm.com/docs/git-push) origin yourbranchname. GitHub will automatically create the branch for you on the remote repository:

### mnelson:myproject mnelson$ git push origin my-new-branch

### Counting objects: 3, done.

### Delta compression using up to 8 threads.

### Compressing objects: 100% (2/2), done.

### Writing objects: 100% (3/3), 313 bytes | 0 bytes/s, done.

### Total 3 (delta 0), reused 0 (delta 0)

### To https://github.com/cubeton/mynewrepository.git

### \* [new branch] my-new-branch -> my-new-branch

### [view raw](https://gist.github.com/cubeton/bf8274609c344b6d0e70/raw/4764e740cac9a48eefad341d9e34ceb09f89b73f/addnewbranchgithub.md)

### [addnewbranchgithub.md](https://gist.github.com/cubeton/bf8274609c344b6d0e70#file-addnewbranchgithub-md) hosted with ❤ by [GitHub](https://github.com/)

### You might be wondering what that "origin" word means in the command above. What happens is that when you clone a remote repository to your local machine, git creates an alias for you. In nearly all cases this alias is called "[origin](https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes)." It's essentially shorthand for the remote repository's URL. So, to push your changes to the remote repository, you could've used either the command: git push git@github.com:git/git.git yourbranchname or git push origin yourbranchname

### (If this is your first time using GitHub locally, it might prompt you to log in with your GitHub username and password.)

### If you refresh the GitHub page, you'll see note saying a branch with your name has just been pushed into the repository. You can also click the 'branches' link to see your branch listed there.

### Git_3

### Now click the green button in the screenshot above. We're going to make a **pull request**!

## **Step 8: Create a pull request (PR)**

### A pull request (or PR) is a way to alert a repo's owners that you want to make some changes to their code. It allows them to review the code and make sure it looks good before putting your changes on the primary branch.

### This is what the PR page looks like before you've submitted it:

### Git_4

### And this is what it looks like once you've submitted the PR request:

### Git_5

### You might see a big green button at the bottom that says 'Merge pull request'. Clicking this means you'll merge your changes into the primary branch..

### Sometimes you'll be a co-owner or the sole owner of a repo, in which case you may not need to create a PR to merge your changes. However, it's still a good idea to make one so you can keep a more complete history of your updates and to make sure you always create a new branch when making changes.

## **Step 9: Merge a PR**

### Go ahead and click the green 'Merge pull request' button. This will merge your changes into the primary branch.

### Git_6

### When you're done, I recommend deleting your branch (too many branches can become messy), so hit that grey 'Delete branch' button as well.

### You can double check that your commits were merged by clicking on the 'Commits' link on the first page of your new repo.

### Git_7

### This will show you a list of all the commits in that branch. You can see the one I just merged right up top (Merge pull request #1).

### Git_8

### You can also see the [hash code](https://git-scm.com/docs/git-hash-object) of the commit on the right hand side. A hash code is a unique identifier for that specific commit. It's useful for referring to specific commits and when undoing changes (use the [git revert](http://git-scm.com/docs/git-revert) <hash code number> command to backtrack).

## **Step 10: Get changes on GitHub back to your computer**

### Right now, the repo on GitHub looks a little different than what you have on your local machine. For example, the commit you made in your branch and merged into the primary branch doesn't exist in the primary branch on your local machine.

### In order to get the most recent changes that you or others have merged on GitHub, use the git pull origin master command (when working on the primary branch). In most cases, this can be shortened to “git pull”.

### mnelson:myproject mnelson$ git pull origin master

### remote: Counting objects: 1, done.

### remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0

### Unpacking objects: 100% (1/1), done.

### From https://github.com/cubeton/mynewrepository

### \* branch master -> FETCH\_HEAD

### b345d9a..5381b7c master -> origin/master

### Merge made by the 'recursive' strategy.

### mnelson.txt | 1 +

### 1 file changed, 1 insertion(+)

### [view raw](https://gist.github.com/cubeton/48b5c726b496d50c3975/raw/fe2c68e0988c467fd218587e2397552076355b52/pulloriginmaster.md)

### [pulloriginmaster.md](https://gist.github.com/cubeton/48b5c726b496d50c3975#file-pulloriginmaster-md) hosted with ❤ by [GitHub](https://github.com/)

### This shows you all the files that have changed and how they've changed.

### Now we can use the [git log](http://git-scm.com/docs/git-log) command again to see all new commits.

### (You may need to switch branches back to the primary branch. You can do that using the git checkout master command.)

### mnelson:myproject mnelson$ git log

### commit 3e270876db0e5ffd3e9bfc5edede89b64b83812c

### Merge: 4f1cb17 5381b7c

### Author: Meghan Nelson <mnelson@hubspot.com>

### Date: Fri Sep 11 17:48:11 2015 -0400

### 

### Merge branch 'master' of https://github.com/cubeton/mynewrepository

### 

### commit 4f1cb1798b6e6890da797f98383e6337df577c2a

### Author: Meghan Nelson <mnelson@hubspot.com>

### Date: Fri Sep 11 17:48:00 2015 -0400

### 

### added a new file

### 

### commit 5381b7c53212ca92151c743b4ed7dde07d9be3ce

### Merge: b345d9a 1e8dc08

### Author: Meghan Nelson <meghan@meghan.net>

### Date: Fri Sep 11 17:43:22 2015 -0400

### 

### Merge pull request #2 from cubeton/my-newbranch

### 

### Added some more text to my file

### 

### commit 1e8dc0830b4db8c93efd80479ea886264768520c

### Author: Meghan Nelson <mnelson@hubspot.com>

### Date: Fri Sep 11 17:06:05 2015 -0400

### 

### Added some more text to my file

### 

### commit b345d9a25353037afdeaa9fcaf9f330effd157f1

### Author: Meghan Nelson <mnelson@hubspot.com>

### Date: Thu Sep 10 17:42:15 2015 -0400

### 

### This is my first commit!

### [view raw](https://gist.github.com/cubeton/48f55c5a237cd8e1a238/raw/3e31113a073b9bdec16800407d718b631dd0f587/gitlogaftermerge.md)

### [gitlogaftermerge.md](https://gist.github.com/cubeton/48f55c5a237cd8e1a238#file-gitlogaftermerge-md) hosted with ❤ by [GitHub](https://github.com/)

## **Step 11: Bask in your git glory**

### You've successfully made a PR and merged your code to the primary branch. Congratulations! If you'd like to dive deeper, check out these more advanced tutorials and resources:

### <https://training.github.com/>Github’s official git cheat sheets! Handy for remembering the everyday commands you’ll use.

### <https://learngitbranching.js.org/>Confused or intrigued by git’s branch system? That just means you’re human! It’s one of the deepest parts of git, but also arguably the most powerful. Understanding the branch model gives you git superpowers, and this tutorial gives you a way to learn git branches in a visual, intuitive way.

### [https://git-school.github.io/visualizing-git](https://git-school.github.io/visualizing-git/#rewritten-history) Another tool for exploring git visually. This one is more of an open-ended sandbox than learngitbranching.js.org

### [https://github.com/jlord/git-it-electron](https://github.com/jlord/git-it-electron#what-to-install) A desktop application that helps you learn git through challenges you have to solve. It has a series of levels, each requiring you to use git commands to arrive at a correct answer.

### <https://github.com/Gazler/githug>If you liked git-it, Githug is another puzzle-based tutorial designed to give you a practical way of learning git.

### I also recommend finding some time to work with your team on simulating a smaller group project like we did here. Have your team make a new folder with your team name, and add some files with text to it. Then, try pushing those changes to this remote repo. That way, your team can start making changes to files they didn't originally create and practice using the PR feature. And, use the git blame and git history tools on GitHub to get familiar with tracking which changes have been made in a file and who made those changes.

### 

### 

### 

### 

### **[Introduction](https://www.theodinproject.com/lessons/foundations-git-basics#introduction)**

In this lesson, we’ll cover common Git commands used to manage your projects and to upload your work onto GitHub. We refer to these commands as the basic Git workflow. When you’re using Git, these are the commands that you’ll use 70-80% of the time. So if you can get these down, you’ll be more than halfway done mastering Git!

### **[Lesson overview](https://www.theodinproject.com/lessons/foundations-git-basics#lesson-overview)**

This section contains a general overview of topics that you will learn in this lesson.

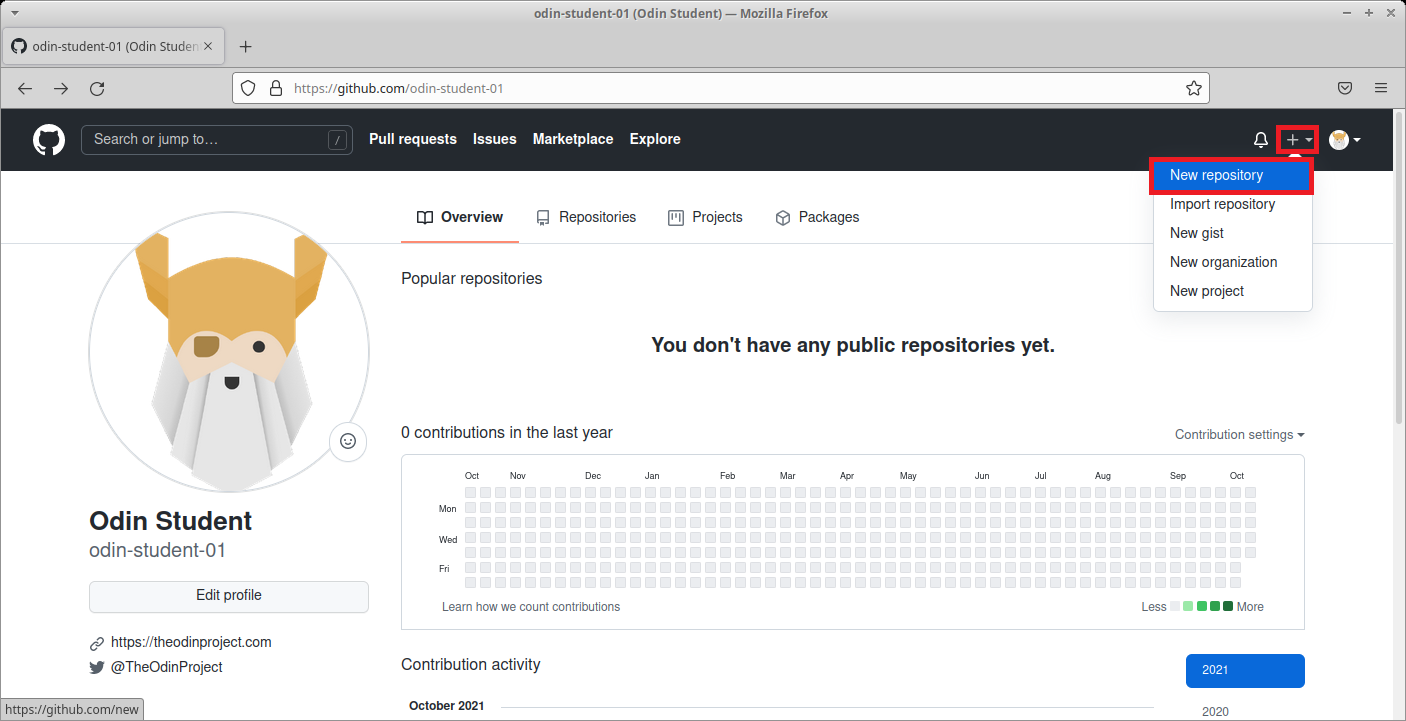
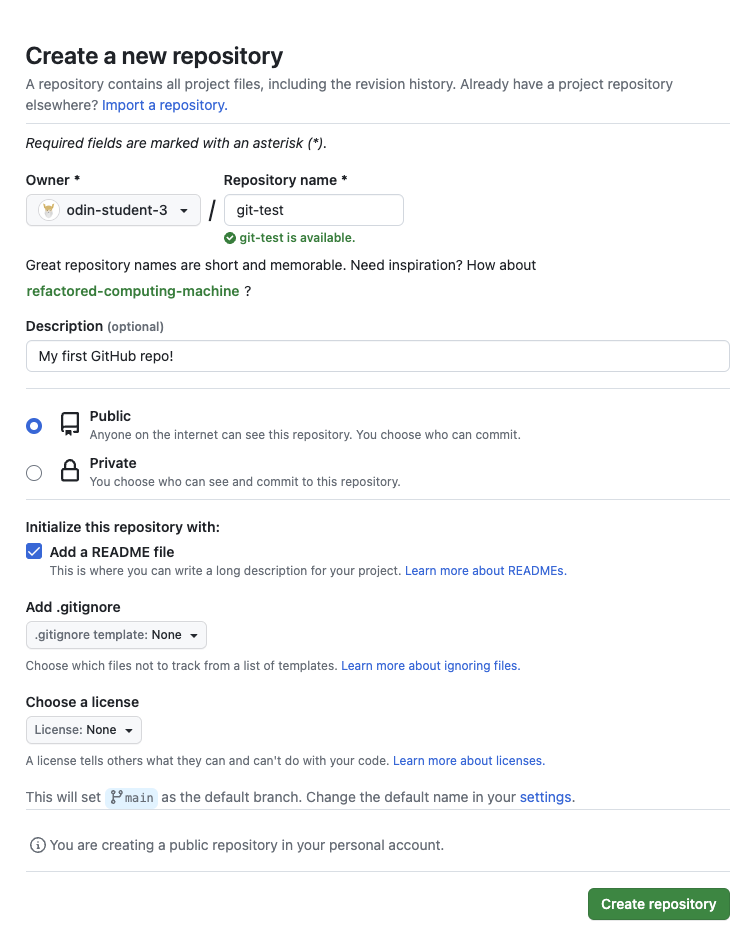
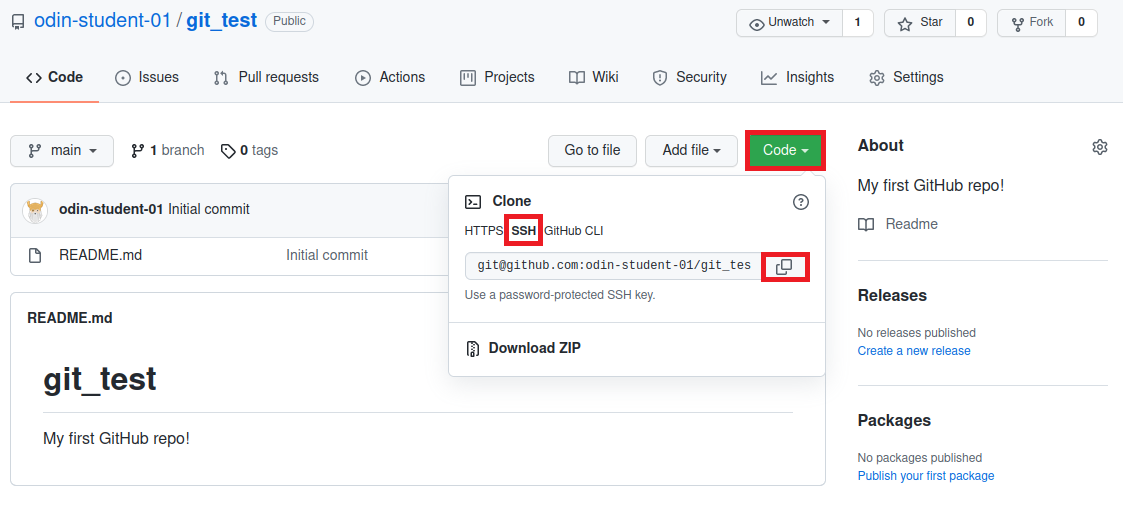
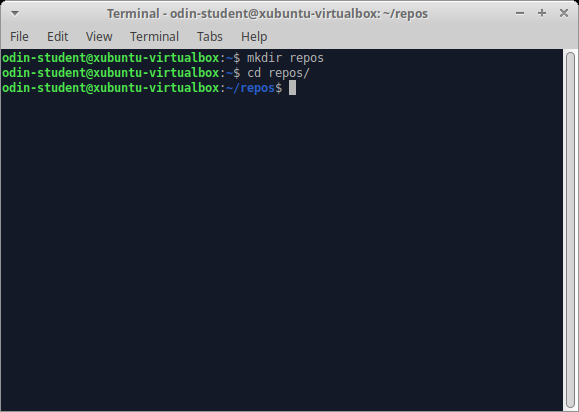
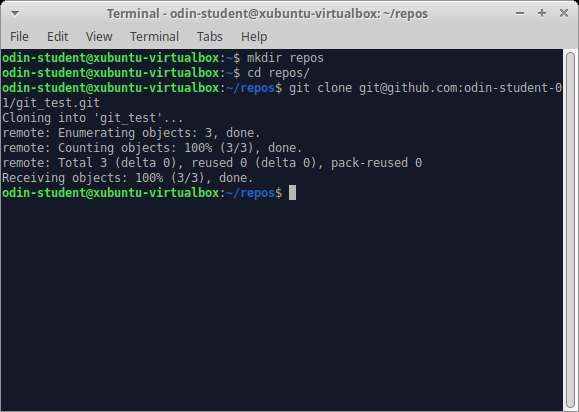
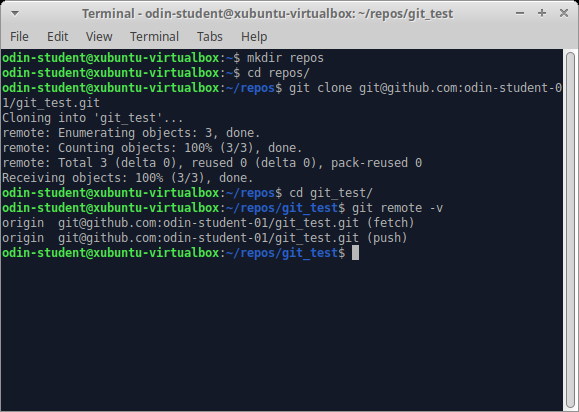
* How to create a repository on GitHub.
* How to get files to and from GitHub.
* How to take “snapshots” of your code.

### **[Assignment](https://www.theodinproject.com/lessons/foundations-git-basics#assignment)**

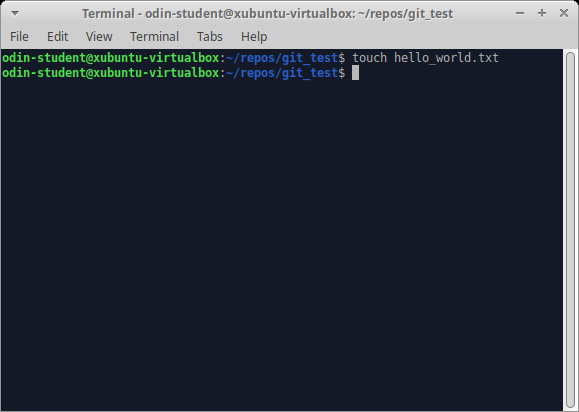
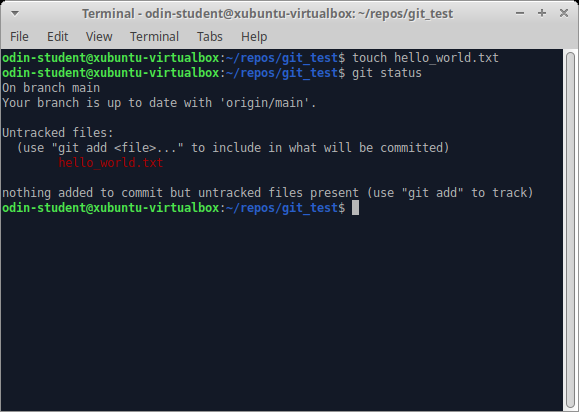
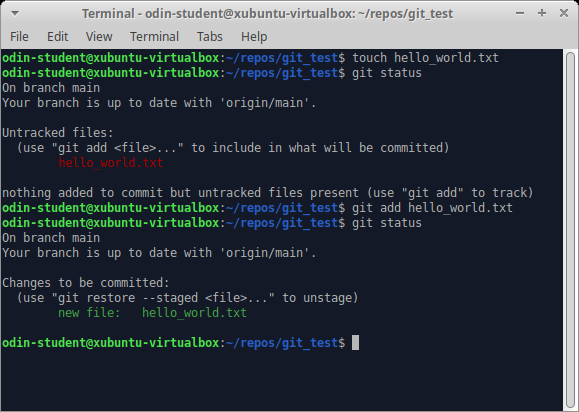
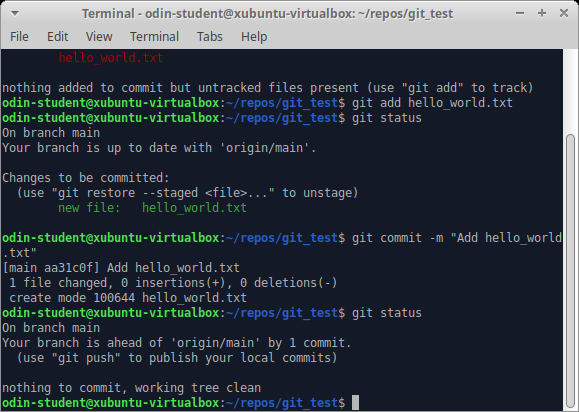
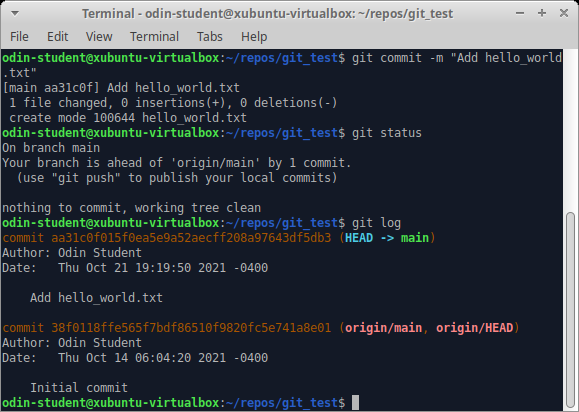
#### **Before you start!**

* Github recently updated the way it names the default branch. This means you need to make sure you are using a recent version of git (2.28 or later). You can check your version by running: git --version
* If you haven’t already, set your local default git branch to main. You can do so by running: git config --global init.defaultBranch main
* For more information on the change from master to main see [GitHub’s Renaming Repository](https://github.com/github/renaming).

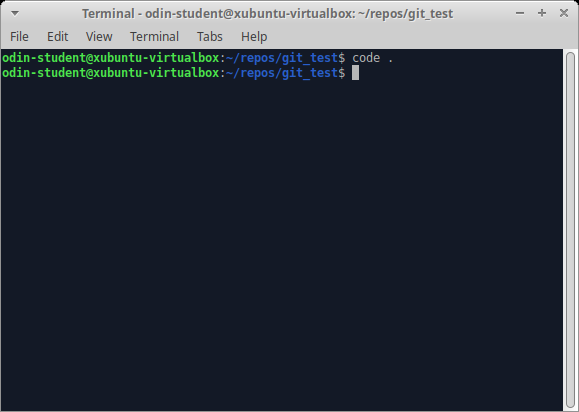
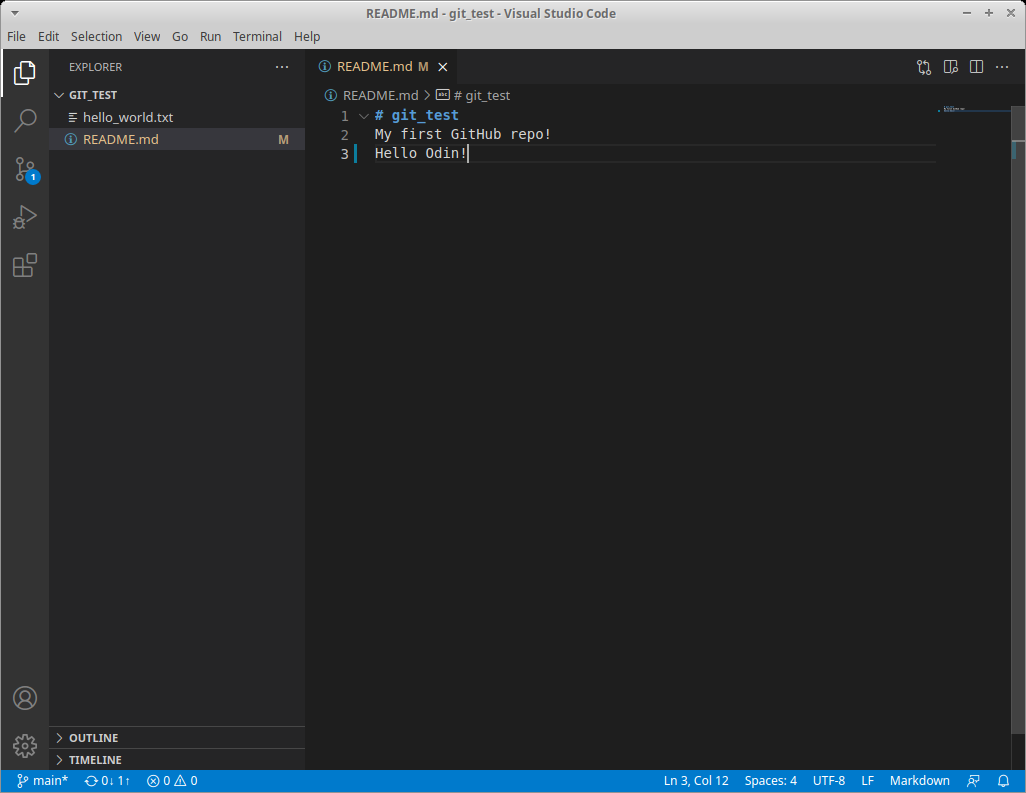
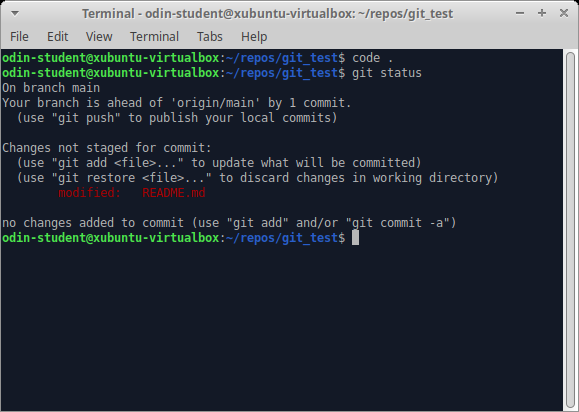
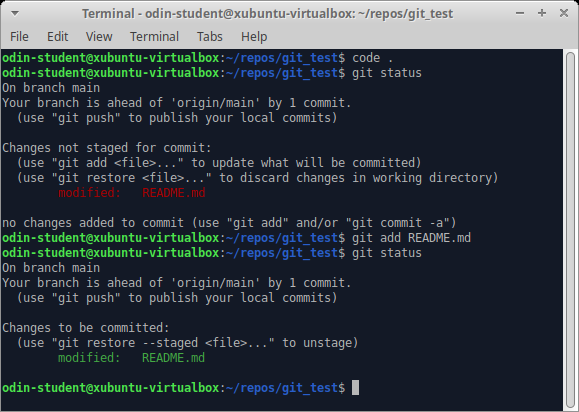
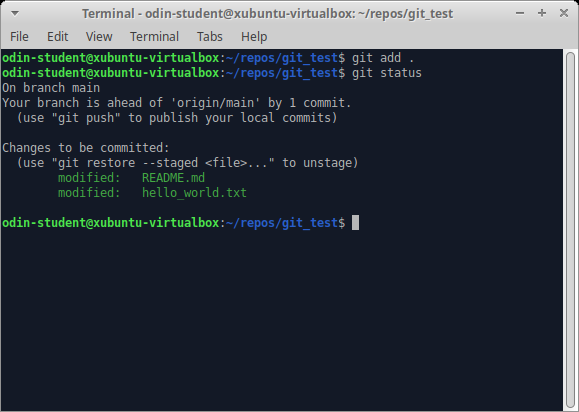
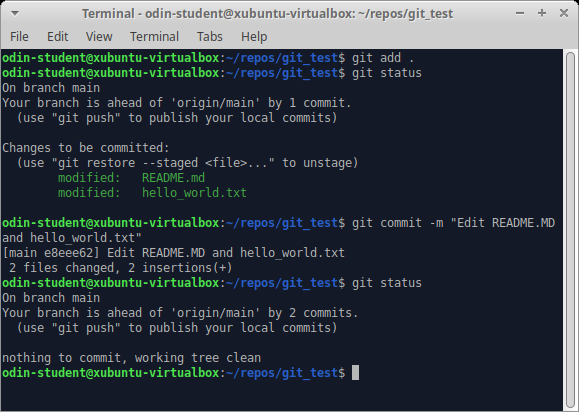
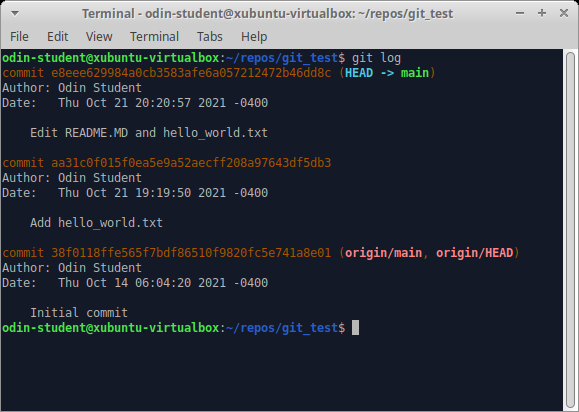
#### **Create the repository**

1. You should have already created a GitHub account in the [Setting Up Git](https://www.theodinproject.com/lessons/foundations-setting-up-git) lesson. If you haven’t done that yet, you can sign up [here](https://github.com/).
2. Create a new repository by clicking the button shown in the screenshot below.  
   
3. Give your repository the name “git\_test” in the repository name input field. Check “Add a README file”. And then create the repository by clicking the “Create repository” button at the bottom of the page.  
   
4. This will redirect you to your new repository on GitHub. To get ready to copy (clone) this repository onto your local machine, click the green “Code” button. Then select the SSH option, and copy the line below it. NOTE: You MUST click the SSH option to get the correct URL.  
   
5. Let’s use the command line on your local machine to create a new directory for all of your Odin projects. Create a directory called repos with the mkdir command in your home folder. Your home folder is represented by ~. [Navigating Files and Directories](https://swcarpentry.github.io/shell-novice/02-filedir.html#callout1) covered variations of home folders - sometimes ~ stands for /Users/your\_username and sometimes it stands for /home/your\_username. If you’re not sure if you’re in your home folder, just type cd ~. Once it’s made, move into it with the cd command.  
   
6. Now it’s time to clone your repository from GitHub onto your computer with git clone followed by the URL you copied in the last step. The full command should look similar to git clone git@github.com:USER-NAME/REPOSITORY-NAME.git. If your URL looks like https://github.com/USER-NAME/REPOSITORY-NAME.git, you have selected the HTTPS option, not the required SSH option.  
   
7. That’s it! You have successfully connected the repository you created on GitHub to your local machine. To test this, you can cd into the new git\_test folder that was downloaded and then enter git remote -v on your command line. This will display the URL of the repository you created on GitHub, which is the remote for your local copy. You may have also noticed the word origin at the start of the git remote -v output, which is the name of your remote connection. The name “origin” is both the default and the convention for the remote repository. But it could have just as easily been named “party-parrot” or “dancing-banana”. (Don’t worry about the details of origin for now; it will come up again near the end of this tutorial.)  
   

#### **Use the Git workflow**

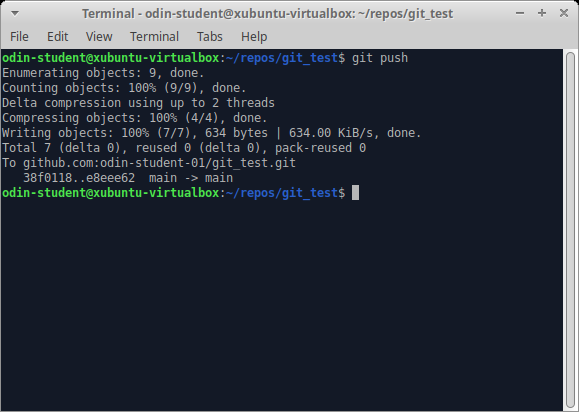
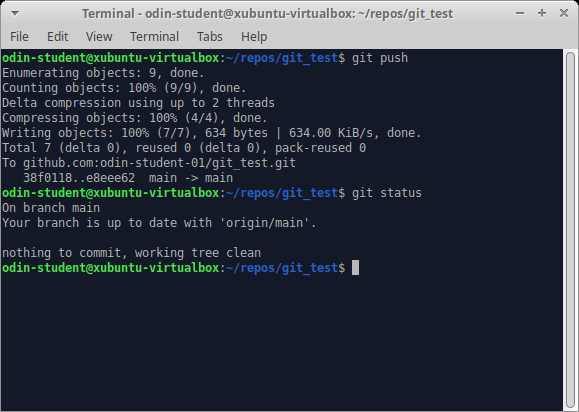
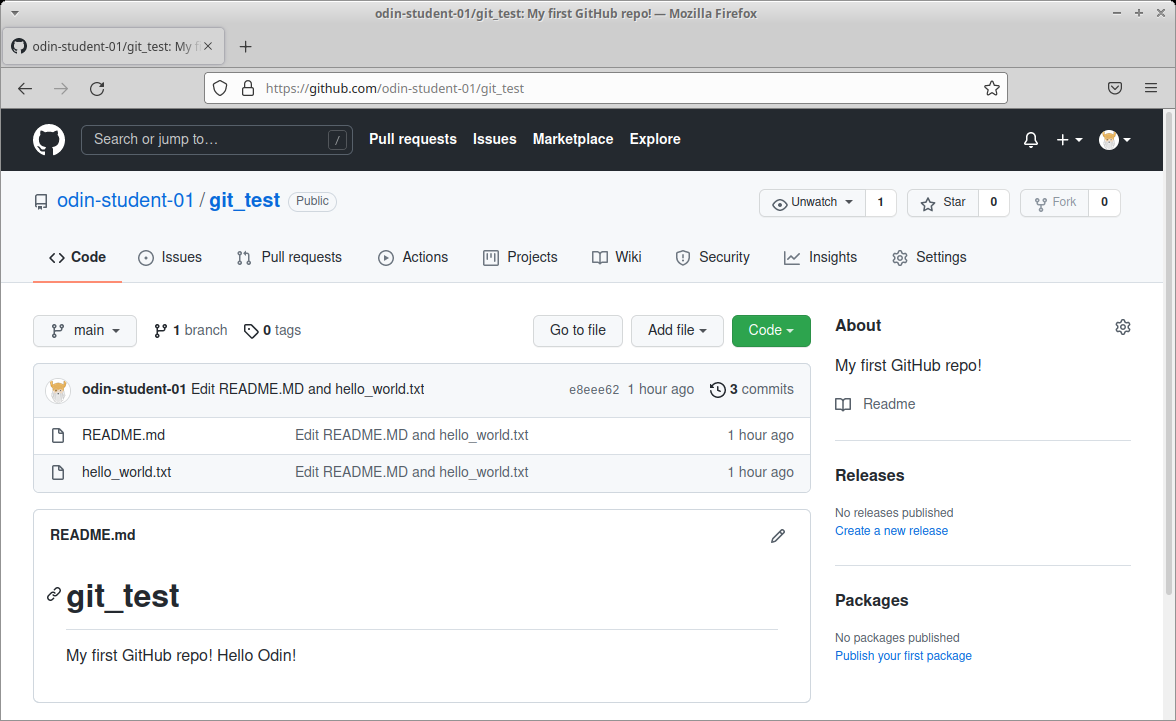
1. Create a new file in the git\_test folder called “hello\_world.txt” with the command touch hello\_world.txt.  
   
2. Type git status in your terminal. In the output, notice that your hello\_world.txt file is shown in red, which means that this file is not staged.  
   
3. Type git add hello\_world.txt. This command adds your hello\_world.txt file to the staging area in Git. The staging area is part of the two-step process for making a commit in Git. Think of the staging area as a “waiting room” for your changes until you commit them. Now, type git status again. In the output, notice that your file is now shown in green, which means that this file is now in the staging area.  
   
4. Type git commit -m "Add hello\_world.txt" and then type git status once more. The output should now say: “*nothing to commit, working tree clean*”, indicating your changes have been committed. Don’t worry if you get a message that says “*upstream is gone*”. This is normal and only shows when your cloned repository currently has no branches. It will be resolved once you have followed the rest of the steps in this project.  
   The message, “*Your branch is ahead of ‘origin/main’ by 1 commit*” just means that you now have newer snapshots than what is on your remote repository. You will be uploading your snapshots further down in this lesson.  
   
5. Type git log and look at the output. You should see an entry for your “*Add hello\_world.txt*” commit. You will also see details on the author who made the commit and the date and time of when the commit was made. If your terminal is stuck in a screen with (END) at the bottom, just press “q” to escape. You can configure settings for this later, but don’t worry about it too much for now.  
   

#### **Modify a file or two**

1. Open README.md in your text editor of choice. In this example, we will open the directory in Visual Studio Code by using the command code . inside your repository.  
     
   MacOS users: If your terminal reads *“command not found: code”*, you must head back to [Command Line Basics](https://www.theodinproject.com/lessons/foundations-command-line-basics#opening-files-in-vscode-from-the-command-line) and follow the instructions provided to allow this command to work.
2. Add “Hello Odin!” to line 3 of README.md and save the file with Ctrl + S (Mac: Cmd + S).  
   
3. Go back to your terminal or if you’re using Visual Studio Code you can open the built-in terminal by pressing Ctrl + ` (backtick). Then type git status. You’ll notice that README.md is now shown as not staged or committed.  
   
4. Add README.md to the staging area with git add README.md.
5. Can you guess what git status will output now? README.md will be displayed in green text. That means README.md has been added to the staging area. The file hello\_world.txt will not show up because it has not been modified since it was committed.  
   
6. Open hello\_world.txt, add some text to it, save it and stage it. You can use git add . to add all files in the current directory and all subsequent directories to the staging area. Then, type git status once more, and everything should now be in the staging area.  
   
7. Finally, let’s commit all of the files that are in the staging area and add a descriptive commit message. git commit -m "Edit README.md and hello\_world.txt". Then, type git status once again, which will output “*nothing to commit*”.  
   
8. Take one last look at your commit history by typing git log. You should now see three entries.  
   

#### **Push your work to GitHub**

Finally, let’s upload your work to the GitHub repository you created at the start of this tutorial.

1. Type git push. To be more specific, type git push origin main. Since you are not dealing with another branch (other than *main*) or a different remote (as mentioned above), you can leave it as git push to save a few keystrokes. NOTE: If at this point you receive a message that says “Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.”, you have followed the steps incorrectly and cloned with HTTPS, not SSH. Please follow [these steps](https://docs.github.com/en/get-started/getting-started-with-git/managing-remote-repositories#switching-remote-urls-from-https-to-ssh) to change your remote to SSH, then attempt to push to Github.  
   
2. Type git status one final time. It should output “*Your branch is up to date with ‘origin/main’. nothing to commit, working tree clean*”.  
   
3. When you reload the repository on GitHub, you should see the README.md and hello\_world.txt files that you just pushed there from your local machine.  
   

### **[Note/Warning](https://www.theodinproject.com/lessons/foundations-git-basics#notewarning)**

When trying to make simple changes to the files in your repo, such as attempting to fix a typo in your README.md you might be tempted to make this change directly via Github. However, it is best to avoid this as it will cause issues that require more advanced Git knowledge than we want to go over at this stage (it is covered in a future lesson), for now it is advised to make any changes via your local files then commit and push them using Git commands in your terminal once ready.

### **[Cheatsheet](https://www.theodinproject.com/lessons/foundations-git-basics#cheatsheet)**

This is a reference list of the most commonly used Git commands. (You might consider bookmarking this handy page.) Try to familiarize yourself with the commands so that you can eventually remember them all:

* Commands related to a remote repository:
  + git clone git@github.com:USER-NAME/REPOSITORY-NAME.git
  + git push or git push origin main (Both accomplish the same goal in this context)
* Commands related to the workflow:
  + git add .
  + git commit -m "A message describing what you have done to make this snapshot different"
* Commands related to checking status or log history
  + git status
  + git log

The basic Git syntax is program | action | destination.

For example,

* git add . is read as git | add | ., where the period represents everything in the current directory;
* git commit -m "message" is read as git | commit -m | "message"; and
* git status is read as git | status | (no destination).

### **[Git best practices](https://www.theodinproject.com/lessons/foundations-git-basics#git-best-practices)**

There’s a lot to learn about using Git. But it is worth taking the time to highlight some best practices so that you can be a better collaborator. Git is not only helpful when collaborating with others. It’s also useful when working independently. You will be relying more and more on your own commit history in the future when revisiting old code.

Two helpful best practices to consider are atomic commits and leveraging those atomic commits to make your commit messages more useful to future collaborators.

An atomic commit is a commit that includes changes related to only one feature or task of your program. There are two main reasons for doing this: first, if something you change turns out to cause some problems, it is easy to revert the specific change without losing other changes; and second, it enables you to write better commit messages. You’ll learn more about what a good commit message looks like in a future lesson!

### **[Changing the Git commit message editor](https://www.theodinproject.com/lessons/foundations-git-basics#changing-the-git-commit-message-editor)**

If you are using *Visual Studio Code* (and you should be if you’re following this curriculum), there’s a way to ensure that if you use git commit without the message flag (-m), you won’t get stuck writing your commit message in [Vim](https://en.wikipedia.org/wiki/Vim_(text_editor)).

Changing the default message editor is a good idea in case you accidentally omit the flag, unless you prefer using Vim. There is no downside to changing it, because you will have the option to write your commit messages in the terminal or in the comfort of VS Code.

The following command will set this configuration. Type (or copy & paste) this command into your terminal and hit Enter.

git config --global core.editor "code --wait"

There will be no confirmation or any output on the terminal after entering this command.

With that done, you can now choose to use either git commit -m "your message here" or git commit to type your message with Visual Studio Code!

To make a commit with Visual Studio Code as the text editor, just type git commit. After you hit Enter a new tab in VS Code will open for you to write your commit message. You may provide more details on multiple lines as part of your commit message. After typing your commit message, save it Ctrl + S (Mac: Cmd + S) and close the tab. If you return to the command line, you will see your commit message and a summary of your changes.