**GitHub Repository Creation + Version Control Tutorial**

Benjamin Rittenhouse & Bradley Dinger - CIS 4282

**Step 1: Create an account at** [**github.com**](https://github.com/)

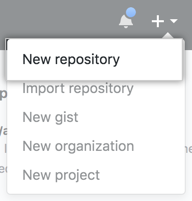
1. It is free to sign up. Use your @temple.edu account if you have one.
2. It is also free to create repositories. Repositories are basically just projects that store any code that you have created as well as the file structure (folders, etc).
   1. You can make private or public repositories, as well as add collaborators by username to be allowed to have admin access to it or view a private repo.

**Terminology / Definitions**

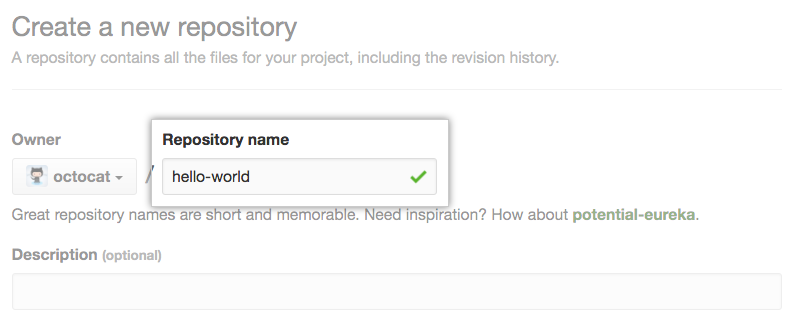
1. Once an account is created, you can now create your own *code repositories*. A code repository is basically a place to store and view code that you have changed as you work on a project.
   1. The term “commit” means making a change to your code, and pushing it to the repository to be reflected among all other changes and project history.
   2. Once you “push” a commit, it means this code is now the most recent version hosted on GitHub.
2. You can view any public repository on GitHub and even “clone” it to your desktop to edit and view the code yourself. Keep in mind, until you commit, nothing is changed on GitHub.
   1. If you are viewing a project that is not yours, or that you do not have modify access to, you can make a *pull request.* A pull request is basically saying “Hello open source project owner, I found some cool changes to make to your code, check them out!”
   2. The owner then looks at your changes and determines if they want to accept them. This is called *merging* a pull request.
3. Scroll further down for more in-depth information about commits, pushs, merges, etc.

**Step 2 - Create your own repository (via** [**github**](https://docs.github.com/en/get-started/quickstart/create-a-repo)**)**

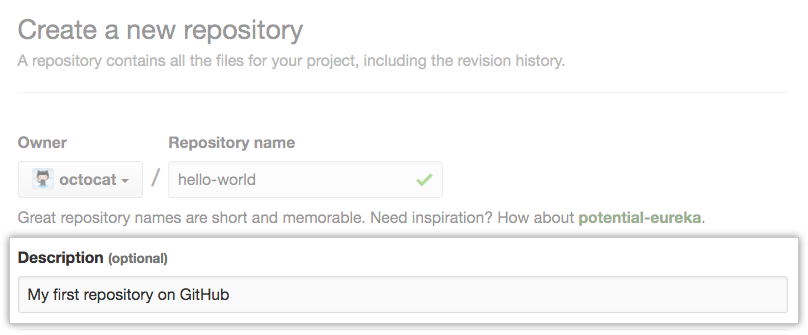
1. In the upper-right corner of any page, use the drop-down menu, and select New repository.



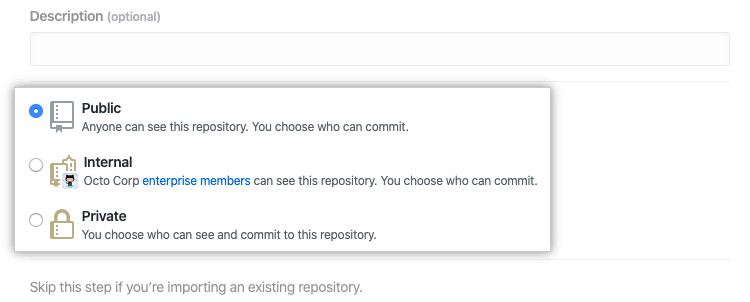
1. Type a short, memorable name for your repository. For example, "hello-world".



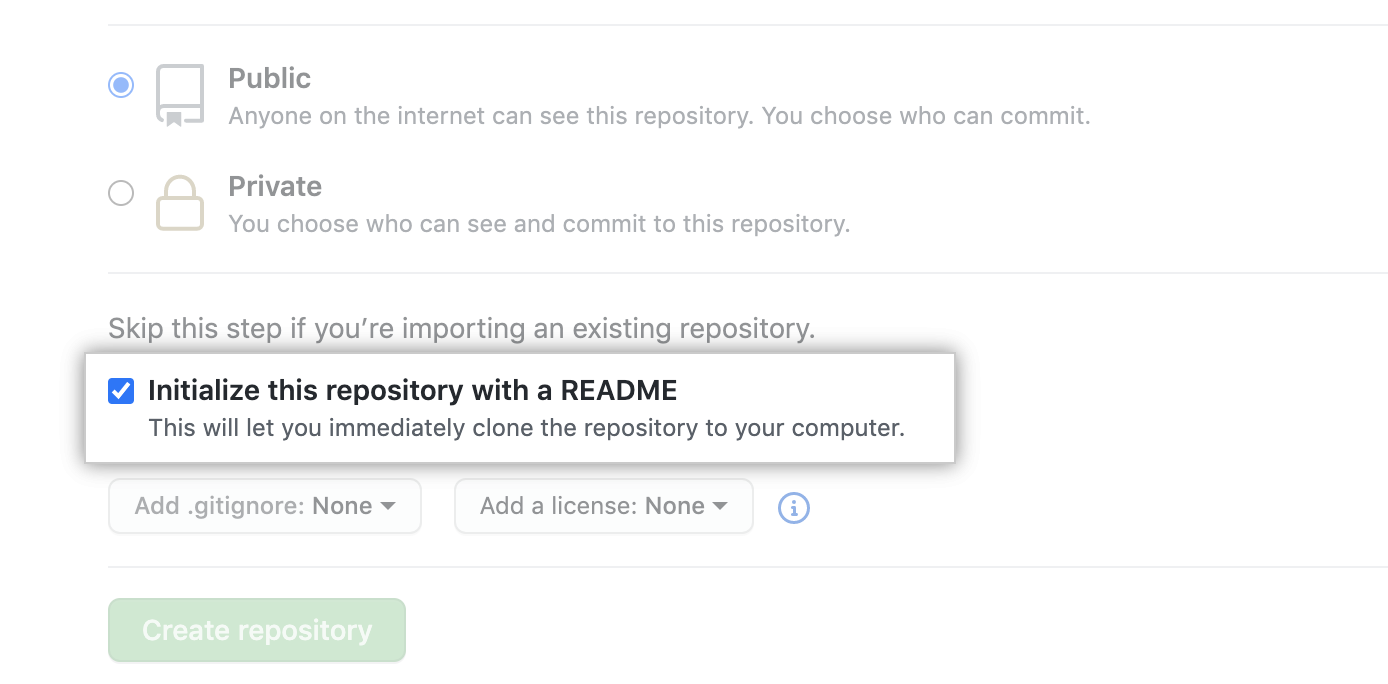
1. Optionally, add a description of your repository. For example, "My first repository on GitHub."



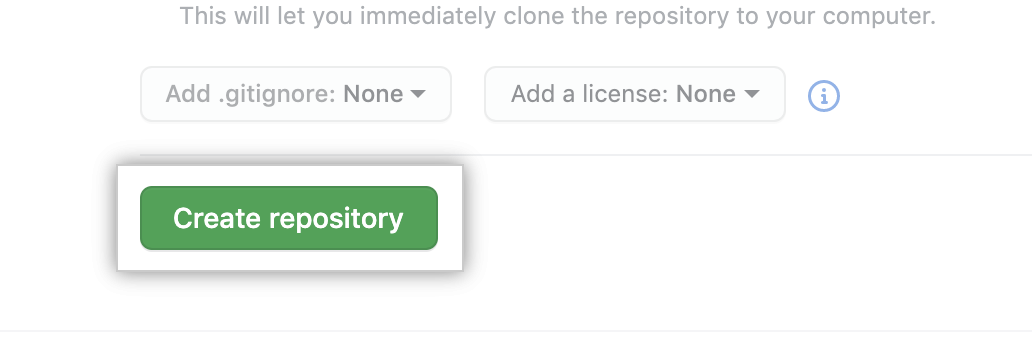
1. Choose a repository visibility. This is based on preference and the context of your project. For something like an independent study, you may want to go private first and change to public once you are finished.
   1. Keep in mind, if you have a private repository and make it public, all changes that were made once private will STILL be available when public!
   2. If you have a **private project** and would like select others to still be able to access it, simply [invite other collaborators](https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-access-to-your-personal-repositories/inviting-collaborators-to-a-personal-repository) (via github.com)



1. Select Initialize this repository with a README. A README doc is basically a description and overview of the repository code, what the project does, and the file structure or technology stack used within the project.



1. Click Create repository.



**Step 3 - Clone and Edit your repository**

1. To work on your project within an IDE or locally on your machine, use the clone command

Within a command line, wherever you want the code to exist, type

**git clone { LINK TO REPOSITORY }**

Where the { LINK TO REPOSITORY } is the new repository you just created or gained access to

If you are getting an error when trying to run “git” in a command line, [install git](https://github.com/git-guides/install-git).

1. Once the project has been cloned, “cd” into it, for example, you should now have a folder “new-repository” (or whatever it was called), you can now type **cd new-repository** to change directories into this repository.
2. Open the folder within your own IDE, for example VSCode or Sublime Text. Now you should be within the folder on both your command line and viewing the files in an IDE
3. This step is free for you to make any changes. A good first step is to **create a new file and commit it**
   1. Simply create a .txt file, .js file, .html file, or whatever you would like. Let’s say we create “index.html”
   2. Write whatever code you want in the file.
   3. Once you are ready to commit, do the following:

Within the command line, type:

**git add index.html**

* NOTE: here, you can add multiple files, or even an entire folder. We will link to all the commands later in the doc

Now that you have “added” the files that need to be committed, we

can go ahead and stage them for a commit with a comment

**git commit -m “comment goes here”**

This comment is important. It gives context to what you are committing, and will be listed in the GitHub commit history so other users know why you edited code, etc.

**Next,** let’s **push** the changes to the repository. This, upon successful completion, will reflect your changes on GitHub.

**git push**

Go check the repository and see your changes! The comment you added should be there as well.

**Collaboration Caution**

In GitHub, you may have multiple people working on one project at the same time. Keep in mind that if your code is “out of date” it may not let you push your changes until you have the most recent version.

For example, let’s say on our repository **hello-world,** my friend Joe pushes some changes while I am working on the code *locally.* I then like my changes, commit and go to push them, but since then, Joe has already made changes. Getting the most recent version uses a **pull** command

Within your project folder in the command line, type:

**git pull**

This will give you the latest code. It is similar to cloning, but just within a project you already have locally.

If you would like to avoid this, **branching** is a good idea. Basically, a branch is a pointer to a project with commits, and you are working on a side version of it. Once your branch is ready, you can request to merge it with the master branch.

This [link](https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging) talks about a **branching workflow overview** and how it works.

Creating a branch: git branch { BRANCH NAME }

Switching to the created branch: git checkout { BRANCH NAME }

Doing both in one command: git checkout -b {BRANCH NAME}

To **merge** this branch into the master branch, aka the code visible on GitHub, use:

git merge { BRANCH NAME }

^ This command must be run from “master”, so you must git checkout master, THEN run git merge { BRANCH NAME }. Within the branch, you still use git add and git commit, just not git push. Merge takes place of git push in this instance.

You can bounce between branches as you please, but for simplicity it may be easiest to use one at a time.