Git and GitHub for Unity Game Development: A Beginner's Guide

Last updated: @December 26, 2022 by @abmarnie

Concise Version

Initial Setup

1. Sign up for GitHub.

2-7. Download, install, and setup Git and Git LFS.

8. Make a local Git repo inside your Unity project using `git init`.

9-12. Add and commit the .gitignore, .gitattributes, and .gitconfig files.

13. Create a new remote GitHub repo, and link it your local Git repo.
14. (Optional). Invite collaborators.
15. (Optional). Write a good README (project style guide, coding standards).

Everyday Workflow

1. Open Git Bash and navigate to your Unity project folder (your Git repo).
2. Keep your local repository up-to-date with any changes made by your collaborators using the `git pull` command.
3. Use the Unity Editor to work on your project and save your work frequently. Communicate with your team to avoid merge conflicts.

4. Check the status of your repository using `git status` when you reach a stopping point.
5. Commit and push your changes using the following commands (in order): `git pull`, `git add .`, `git commit -m "description of changes"`, `git push`.

Introduction and Prerequisites

This is a short guide for those who want to quickly setup a distributed version control system using Git and GitHub in their next Unity project. It is geared towards people who have never used Git or GitHub before. Some prerequisites are:

- A Unity project set up. See here.
- Enough bravery to learn how to use a command line.

What is a version control, and why should I care?

A version control system is a type of software that helps manage and track changes made to computer programs. It allows you to keep a record of a project's history, revert to previous versions, and collaborate with others in real-time. time. A *distributed* version control system stores the complete history of a codebase on each developer's computer, further protecting it from accidental deletion. Git is a popular distributed version control system, and the one which you will be learning today.

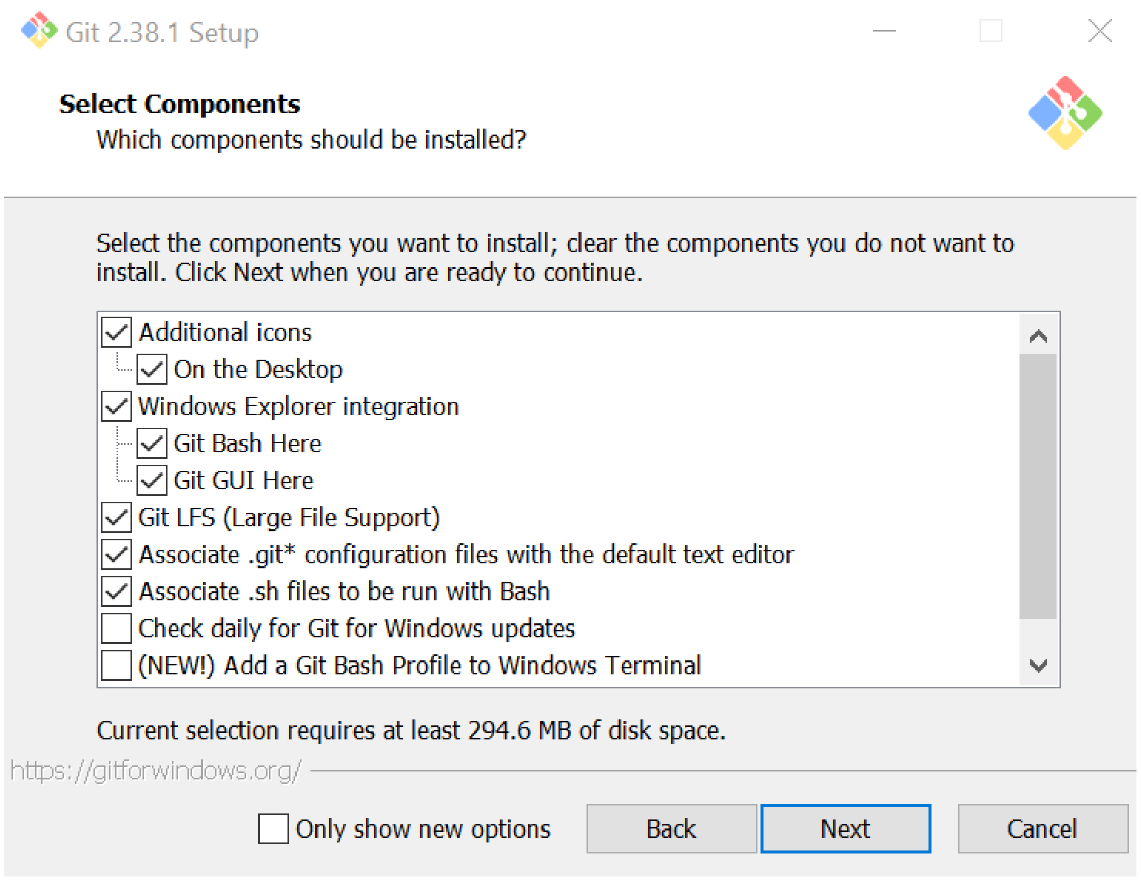
It can be hard to appreciate why all of this is useful if you’ve never used such a thing, but I promise that once you learn how to use Git, you will never want to go back. If you follow this guide, it’ll be super easy to keep track of your project's history, run old versions, and collaborate with others - all while protecting your project from accidental deletions or damage.

How do I set up Git and GitHub?

1. Sign up for GitHub. GitHub is a platform that allows you to host your "Git repositories," which are essentially online versions of your projects. You can access these repositories and collaborate on them with others through the GitHub website.

2. Download and install Git. Git is entirely separate from GitHub; while GitHub is a website that you access through the internet, Git itself is the actual version control software itself. Git is accessed locally (that is, without needing to be connected to the internet) directly using a command line interface called Git Bash. While running the installer, you will have to select some options during setup. The important ones are:

    a. When at the "Select Components" window: make sure that "Git Bash Here", "Git LFS (Large File Support)", "Associate .git* configuration files with the default text editor", and "Associate .sh files to be run with Bash" are selected.



Git 2.38.1 Setup

Select Components
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [✓] Additional icons
  - [✓] On the Desktop
- [✓] Windows Explorer integration
  - [✓] Git Bash Here
  - [✓] Git GUI Here
- [✓] Git LFS (Large File Support)
- [✓] Associate .git* configuration files with the default text editor
- [✓] Associate .sh files to be run with Bash
- [ ] Check daily for Git for Windows updates
- [ ] (NEW!) Add a Git Bash Profile to Windows Terminal

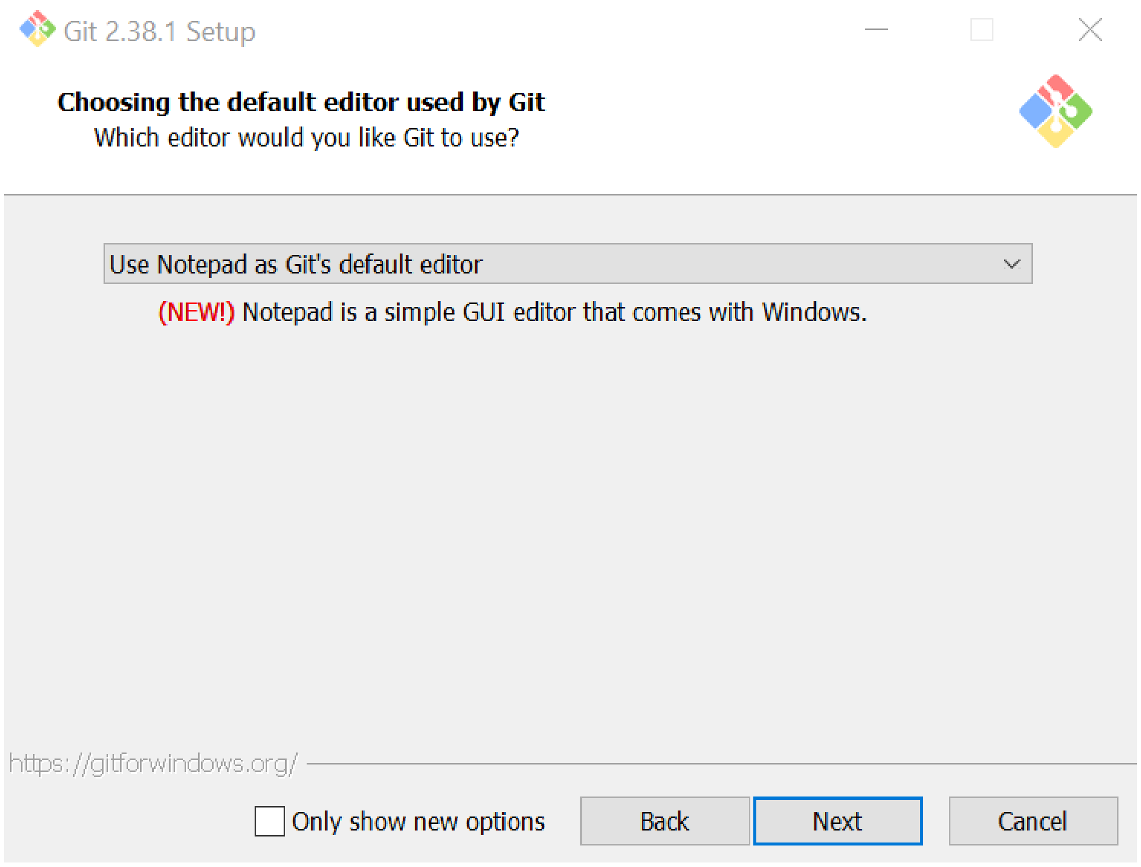
Current selection requires at least 294.6 MB of disk space.

https://gitforwindows.org/

[ ] Only show new options

It should look like this.

    b. When at the "Choosing the default editor used by Git" window: choose one you are comfortable with (e.g., Notepad).


Git 2.38.1 Setup

Choosing the default editor used by Git

Which editor would you like Git to use?

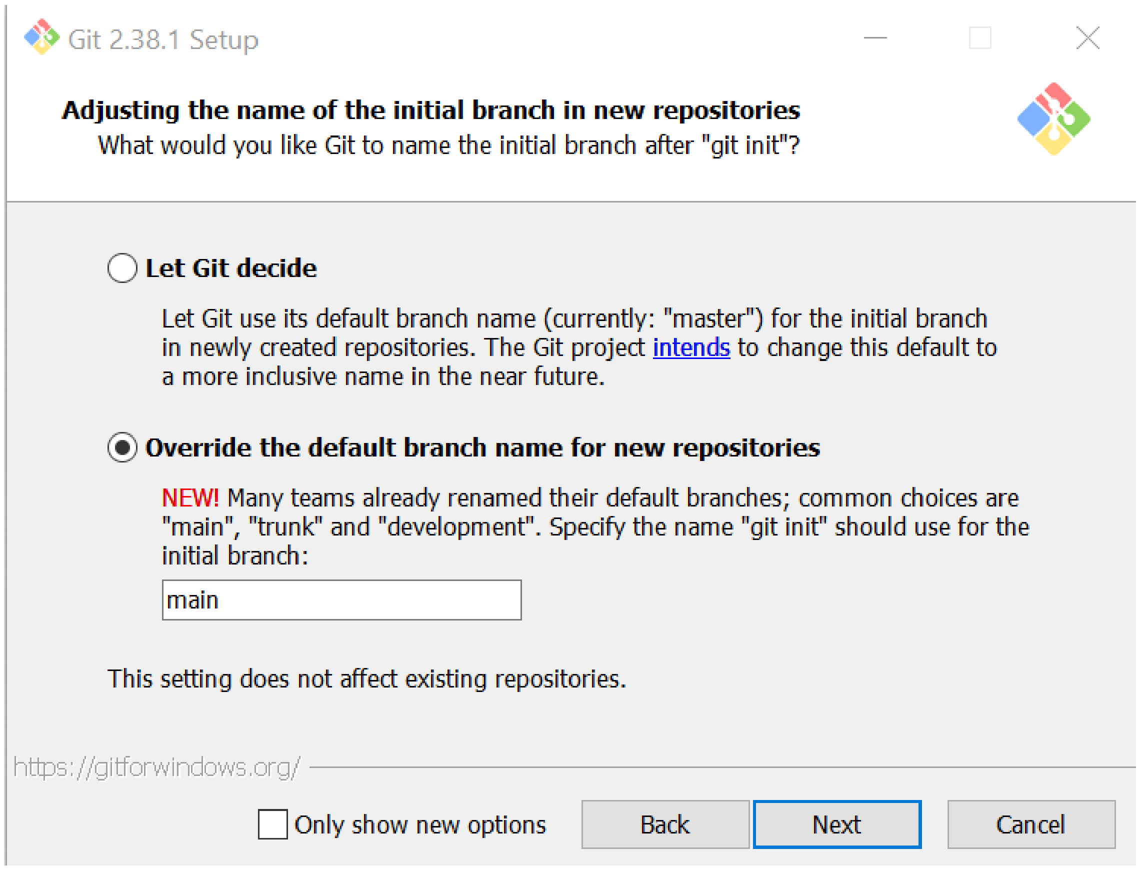
Use Notepad as Git's default editor

(NEW!) Notepad is a simple GUI editor that comes with Windows.

https://gitforwindows.org/

Don’t use VIM unless you’re tech-savvy.

c. When at the “Adjusting the name of the initial branch in new repositories” window: select the second option (”Override the default branch name for new repositories”), and use the name “main” (without quotation marks).


Git 2.38.1 Setup

Adjusting the name of the initial branch in new repositories

What would you like Git to name the initial branch after "git init"?

( ) **Let Git decide**

Let Git use its default branch name (currently: "master") for the initial branch in newly created repositories. The Git project intends to change this default to a more inclusive name in the near future.

(●) **Override the default branch name for new repositories**

NEW! Many teams already renamed their default branches; common choices are "main", "trunk" and "development". Specify the name "git init" should use for the initial branch:

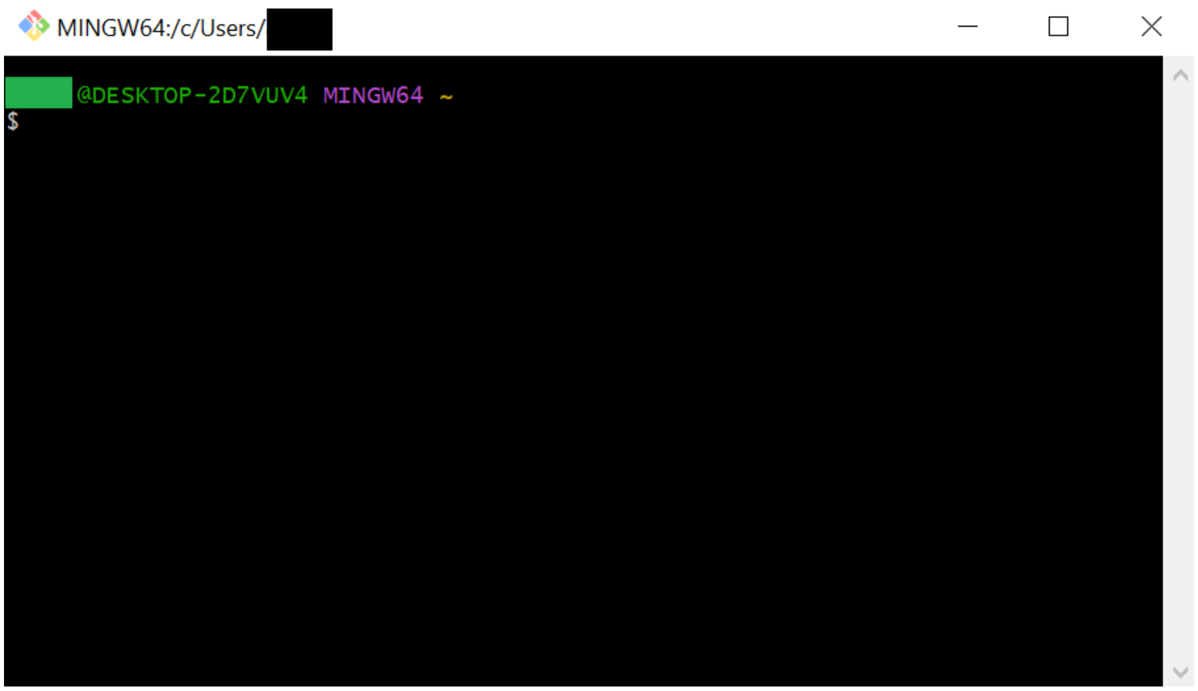
main

This setting does not affect existing repositories.

Select the second option, and use the name "main" (without quotes).

d. For all the other choices, you can just go with the default. Click "Next" until you can finally click "Install".

3. To use Git, launch Git Bash from your computer's start menu or by double-clicking the Git Bash icon on your desktop. You should see a command line interface window appear. In this command line interface, you should see the name of the current user logged into the computer in green, followed by MINGW64 in purple, followed by a tilde ~ (which is shorthand for "home directory"), and finally a dollar sign $ and blinking cursor on the next line.



```
[redacted]@DESKTOP-2D7VUV4 MINGW64 ~
$
```

When you open Git Bash, you should see this command line interface. Whenever you type here, text appears after the dollar sign $.

4. Familiarize yourself with the following Linux Commands: `pwd`, `ls`, `cd`; they allow you to navigate around in Git Bash. To use these commands, simply type them in the command line and press the Enter key. Some commands, like `cd`, require additional arguments to be specified in order to work properly. Others, like `pwd` and `ls` can be used immediately. A summary of these commands is given below.

```
pwd                #stands for "print working directory";
                   # displays the folder that you are currently in

ls                 #stands for "list";
                   # lists the files and folders in your current working directory

cd foldername      #stands for "change directory to foldername";
                   # jumps you into the folder that's named "foldername"

cd ..              #stands for "change directory to parent directory";
                   # jumps you back one folder
```

5. To navigate to your Unity project folder, use the `cd` command in Git Bash. For example, if your Unity project is located in your home directory and is named "ProjectName," you can type `cd ProjectName` and press the Enter key to jump into

that folder. If your project is located elsewhere, you may need to use the `cd` command multiple times to navigate to the correct directory. If you're having trouble finding your project folder, try using the `pwd` and `ls` commands to view the current directory and list its contents to help you navigate. If you're still unable to find your project, it may also be helpful to use Window's File Explorer to help you find it.

6. Setup Git to recognize you by typing in the following two commands, replacing "username" and "emailaddress@example.com" with your own information. Make sure to
   include quotation marks for "username" (but not for email address).

   ```
   $ git config --global user.name "username"
   $ git config --global user.email emailaddress@example.com
   ```

7. Download and install Git LFS. Once the installer has been run, back in Git Bash, type the following command:

   ```
   git lfs install
   ```

8. Now that you have Git and Git LFS installed and set up, it's time to create a new repository for your project. A repository is a tracked project that is managed and tracked in real-time using Git. It contains not only your project, but also all of its history. To create a new repository, first make sure you are inside your Unity project folder by using the `pwd` command in Git Bash. Then, use the `ls` command to list the contents of the folder and confirm that you see a folder named "Assets." Once you are certain you are in the correct directory, enter the following command in Git Bash:

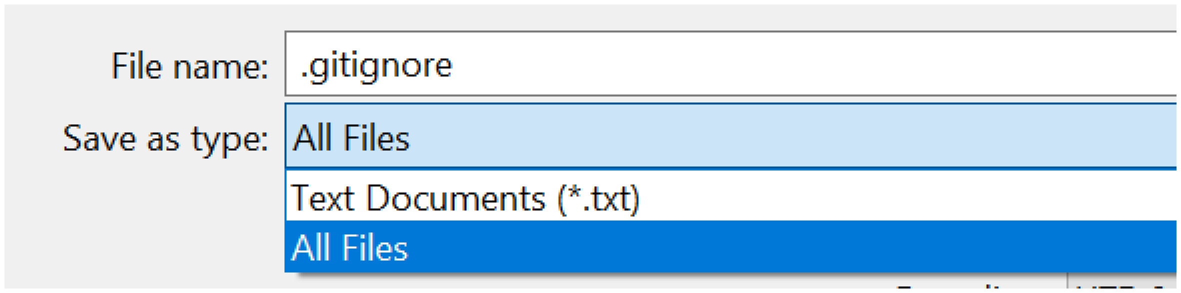
   ```
   git init
   ```

9. Now that your Unity project is being tracked by Git, all changes made within this folder will be recorded. However, there may be certain types of files that you don't want Git to track, such as temporary or backup files. The .gitignore file tells Git which types of files to ignore when tracking changes in your project; using it will

reduce clutter and improve performance. To set up .gitignore, copy the contents of the provided .gitignore file below.

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/c27ca5e0-792b-4eb5-9c7d-8837e94d1d8f/.gitignore

Then, open up a text editor (such as notepad), and copy and paste the contents of that file into the text editor. Finally, save the the file as “.gitignore” (make sure to not save it as a “.txt” file), and place the saved file inside your Unity project folder.


File name: .gitignore

Save as type: All Files

Make sure to select “All Files” in the “Save as type:” menu, otherwise, it will be saved as a .txt file.

10. Large files can cause problems when tracking changes using Git, as they can be slow to process and use up a lot of memory (a limited resource when using GitHub). That’s where Git LFS comes in. To set up Git LFS, you will need to use a .gitattributes file. Copy the contents of the provided .gitattributes file below.

    https://s3-us-west-2.amazonaws.com/secure.notion-static.com/f9416dfa-5ae8-448a-8753-ffddf604e6e7/.gitattributes

    Then, open up a text editor (such as notepad), and copy and paste the contents of that file into the text editor. Finally, save the the file as “.gitattributes” (make sure to not save it as a “.txt” file), and place the saved file inside your Unity project folder.

11. Git can sometimes have difficulty handling "merge conflicts" when working with Unity projects. To improve Git's handling of these conflicts, you can use YAML

Smart Merge. To set up YAML Smart Merge, you'll need to use a .gitconfig file. Copy the contents of the provided .gitconfig file below.

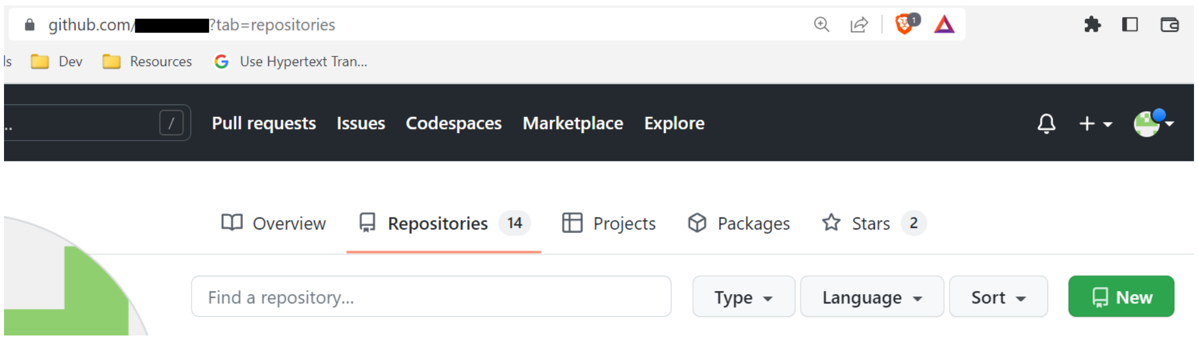
https://s3-us-west-2.amazonaws.com/secure.notion-static.com/32dd43b8-8f30-4bce-a0cc-6ead286a124c/.gitconfig

Then, open up a text editor (such as notepad), and copy and paste the contents of that file into the text editor. Finally, save the the file as ".gitconfig" (make sure to not save it as a ".txt" file), and place the saved file inside your Unity project folder.

12. Now that you've created a .gitignore, .gitattributes, and .gitconfig file, you can add them to your repository by typing the following two commands in Git Bash:

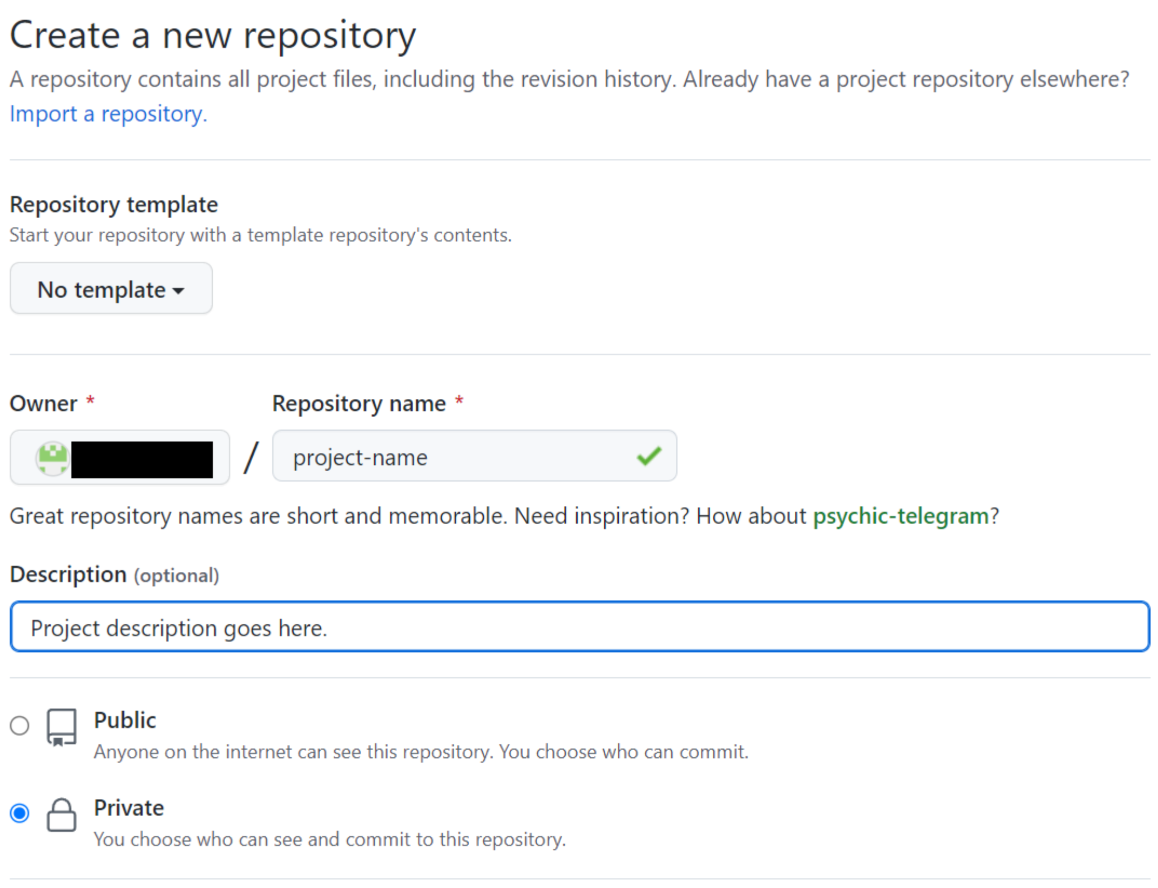
```
git add .
git commit -m "Added .gitignore, .gitattributes, and .gitconfig files."
```

12. With all of this done, you've setup a local version control. All that's left is to host it remotely on GitHub. Log into your GitHub account, navigate to your Repositories page, and click on the green "New" button on the right.



Click the green "New" button on the right.

After doing so, you should see a page titled "Create a new repository". Here, you can set your repository name (name it whatever your Unity project is named), and type a description of your project. *Make sure to set your repository to private if you do not want to open source your project.*



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Repository template

Start your repository with a template repository's contents.

No template

Owner * [redacted] / Repository name * project-name

Great repository names are short and memorable. Need inspiration? How about psychic-telegram?

Description (optional)

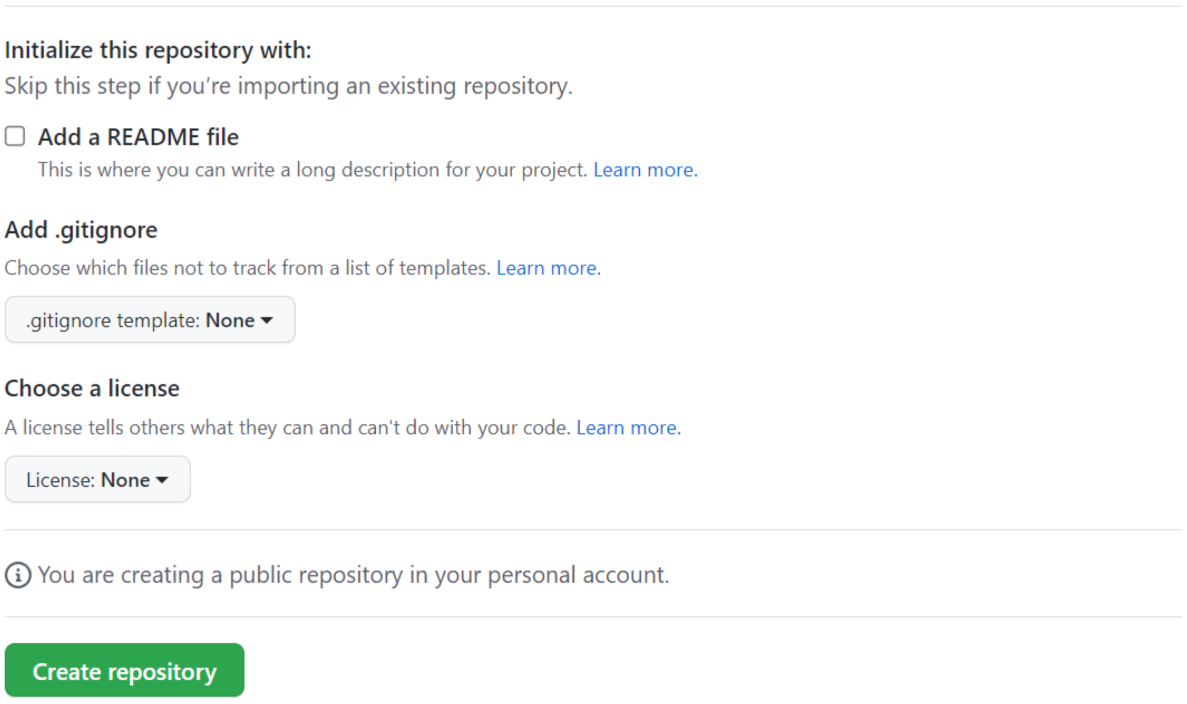
Project description goes here.

Public — Anyone on the internet can see this repository. You choose who can commit.

[✓ Private] — You choose who can see and commit to this repository.

Name your project, write a description, and set your project to private.

Below this, you will see some more options. Ignore them (if you are open sourcing your project, or working with other people, it may be worth researching which license to use and to select the appropriate license). Click the green “Create repository” button at the bottom of the page.


Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ Add a README file
This is where you can write a long description for your project. Learn more.

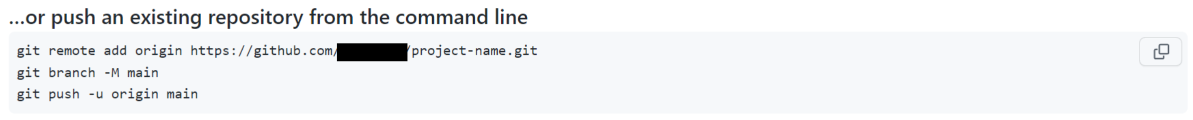
Add .gitignore
Choose which files not to track from a list of templates. Learn more.

Choose a license
A license tells others what they can and can't do with your code. Learn more.

You are creating a public repository in your personal account.

Ignore everything, just click "Create repository" at the bottom.

You should be greeted by a page with some instructions. The instructions you should follow are listed under the "…or push an existing repository from the command line" section. Inside Git Bash, type the commands listed below (replacing my URL with whatever URL it gives you).


…or push an existing repository from the command line

```
git remote add origin https://github.com/[redacted]/project-name.git
git branch -M main
git push -u origin main
```

```
git remote add origin https://github.com/username/project-name.git
git branch -M main
git push -u origin main
```

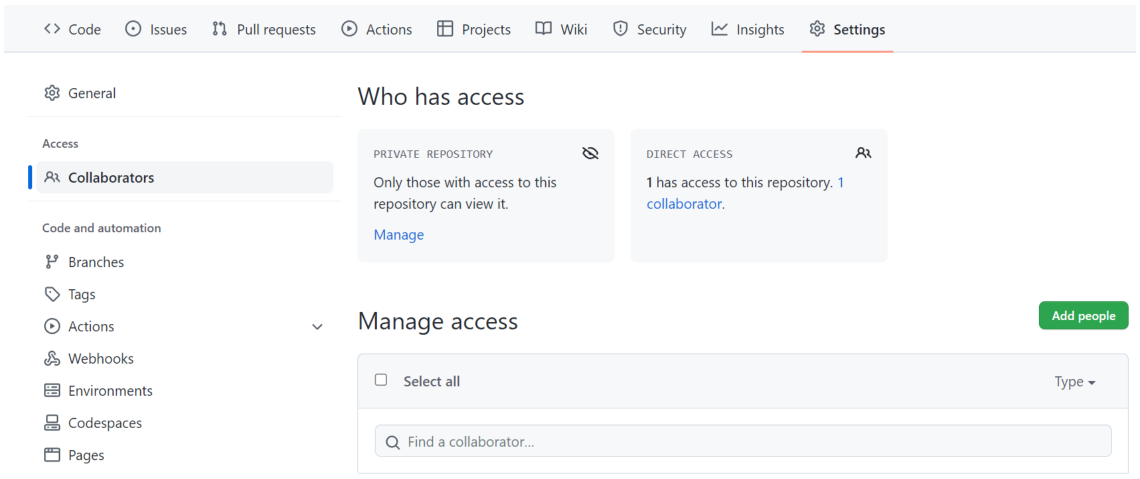
Gratz! You have now set up a local repository (a version control system stored on your local machine) and a remote repository (a copy hosted on GitHub for backup and collaboration purposes). With these repositories, you can track changes made to your Unity project, collaborate with others, and access your code from any device.

13. (Optional) If you have collaborators, they will need access to your Git repository too. For them to get access:

    i. They need to make a GitHub account (step 1).

    ii. They need to download, install, and setup Git and Git LFS (steps 2-4 and steps 6-7, respectively).

    iii. If your GitHub repo is set to private, you will need to add them as a collaborator (if it is public, skip to step 'v'). To do so, go to your GitHub repo webpage (the URL should be something like https://github.com/username/project-name.git). Then click the "Settings" tab. Then, on the left side under "Access", click "Collaborators". To the right of "Manage Access", there is a green "Add people" button; click it, and then type in the email addresses or GitHub usernames of all your project collaborators. This sends each of them an email invite link.



Who has access

PRIVATE REPOSITORY
Only those with access to this repository can view it.

DIRECT ACCESS
1 has access to this repository. 1 collaborator.

Manage access

Click Settings > Collaborators > Add people.

    iv. They need to go to their email and accept the email invite link. Once they've done so, they should be able access the repo hosted on GitHub.

    v. They need to open up Git Bash, navigate to where they want to download the project using the `cd` command, and then type in:

```
git clone https://github.com/username/project-name.git
```

    Now your collaborator should have a complete copy of the project and it's history! Not only that, but they can follow the workflow in the next section to update the project and synch their changes with yours.

15. (Optional). Write a good project README.md. You can do this by clicking the Edit button on the GitHub page for your project.

What is the everyday workflow?

Here is a basic/minimalist workflow for using Git:

1. Open up Git Bash, and use the `cd` command to navigate to your Unity project folder (your Git repo). You can double check that you're in the right folder by typing `git status`.
2. Type `git pull`. This "pulls" any changes that any of your collaborators might have "pushed" to your remote (GitHub hosted) Repo. It basically keeps you up-to-date.
3. Open up your Unity project using the Unity Editor, work on your project (making changes like you normally would), and save your work frequently (like you normally would).
    a. If you are collaborating with a team, make sure to closely communicate to avoid "merge conflicts". The most painful merge conflicts are YAML merge conflicts that occur when multiple people edit the same Unity Scene, or edit the same Prefab file. To avoid this, I recommend having each team member have their own dedicated Scene to work in, and for them to make copies of any prefab files before editing them. If communication is good, a dedicated team member can manually reimplement any changes in the "final" scene. If you encounter normal merge conflicts (from editing the same code; not a YAML merge conflict), they can be easily fixed by manually deleting lines of code in most cases. Also, running `git pull` frequently is a good habit while working, especially if you know a collaborator is working at the same time as you.
4. Once you've gotten to a good stopping point (you've saved your work, right?) check the status of your repo through `git status`. This is a good habit to do; it displays a bunch of useful information to you, such as any new files you've added, which files have changed, and more.
5. Once you're happy with what you see, and are ready to commit your changes, type the following commands:

```
git pull
git add .
git commit -m "description of changes you made"
git push
```

1. The first command “pulls” any updates your collaborators might have made while you were working. You should always pull before you make a commit. Sometimes, pulling leads to merge conflicts.
2. The first command “adds” all the changes you made to a “staging area”, basically getting the changes ready to commit.
3. The second command “commits” all the changes you made, officially logging it into the project history (make sure to replace “description of changes you made” with an actual description, such as “fixed the floating enemy bug”, also make sure to *include the quotation marks*).
4. The final command “pushes” all the local commits you’ve last made to the remote repo; it updates the remote repo so that your collaborators can have access to the updates you just made, and also helps keep your project safe and accessible on other devices.

So, the basic workflow just consists of doing these five steps over and over again. After you’ve made sufficiently many commits, your project will eventually be finished! It's a good idea to push and pull frequently while working in a team to keep everyone up-to-date and minimize merge conflicts.

Further considerations

For the sake of brevity, I’ve only listed the most bare-bones workflow above, that is, the commands that you will be using every day, 99% of the time. But sometimes, you may need to do something specific, in which case, you should search up what you want to do on Stack Overflow. For example:

How do I undo the most recent local commits in Git?

How do I undo 'git add' before commit?

How do I revert a Git repository to a previous commit?

If you are working with a bigger team (with multiple full-time Unity developers), you should look up more advanced Git tutorials which cover topics like branching workflows. You should also consider issue tracking (via GitHub, especially or larger projects), as well as automated push notifications to whichever platform your team most frequently communicates. For example, if you use Discord, you can search up “GitHub Discord Integration”. You should also eventually create a “README.md” file.