Stillwater Creative Tech Docs

Welcome to the **Stillwater Creative Tech Docs**. Here you'll find a guide to the tools used for development at Stillwater Creative. This guide is intended to be a living document, and will be updated as new tools are added or existing tools are updated.

For the latest version of this guide visit stillwater-tech-guide.pages.dev

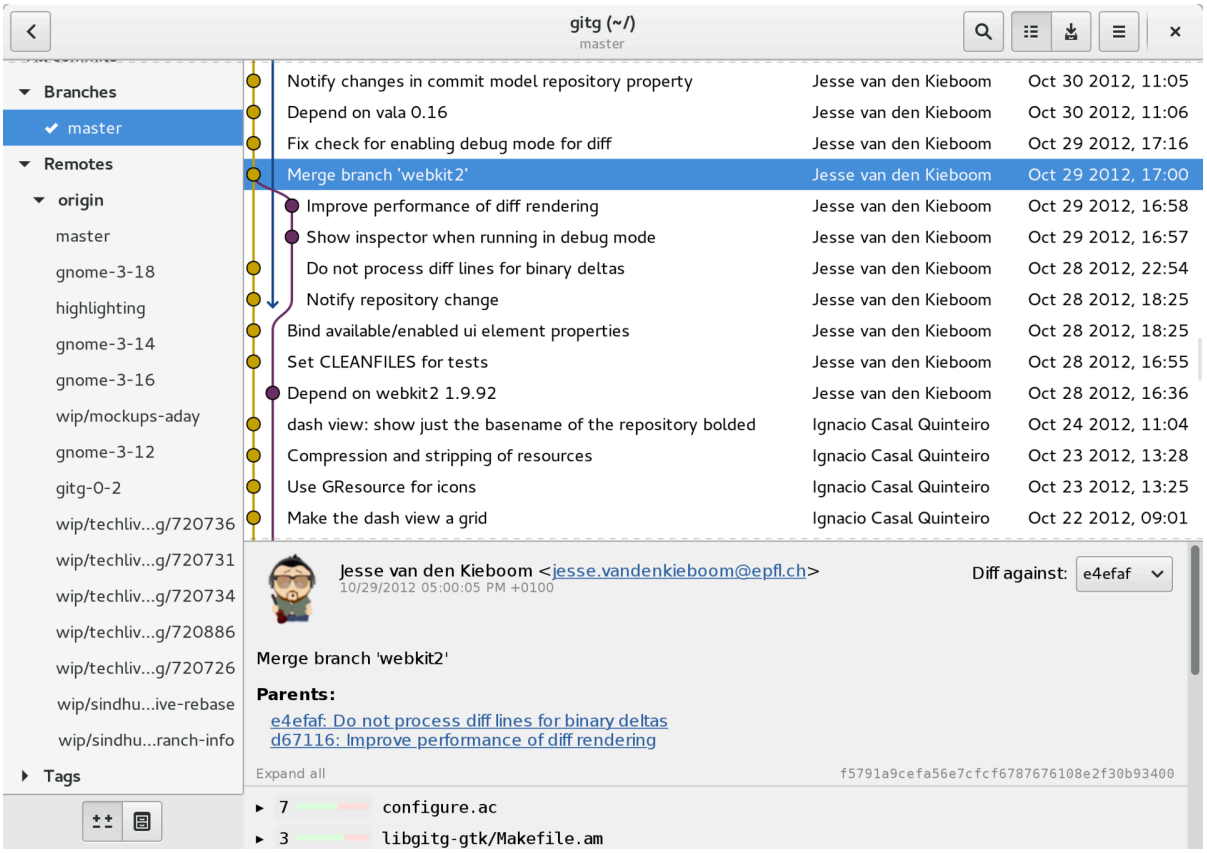
This documentation is confidential and should not be shared outside of Stillwater Creative.

Changelog

Version	Date (mm-dd-yyyy)	Changes
v1.0.0	07-28-2026	First version of the guide

Introduction: Git

Git is a version control system that tracks changes across your project. Git is a decentralized system, meaning it keeps a local copy of your entire repo, allowing you to rapidly compare changes. By adding a centralized server to this, you are able to collaborate with multiple people.

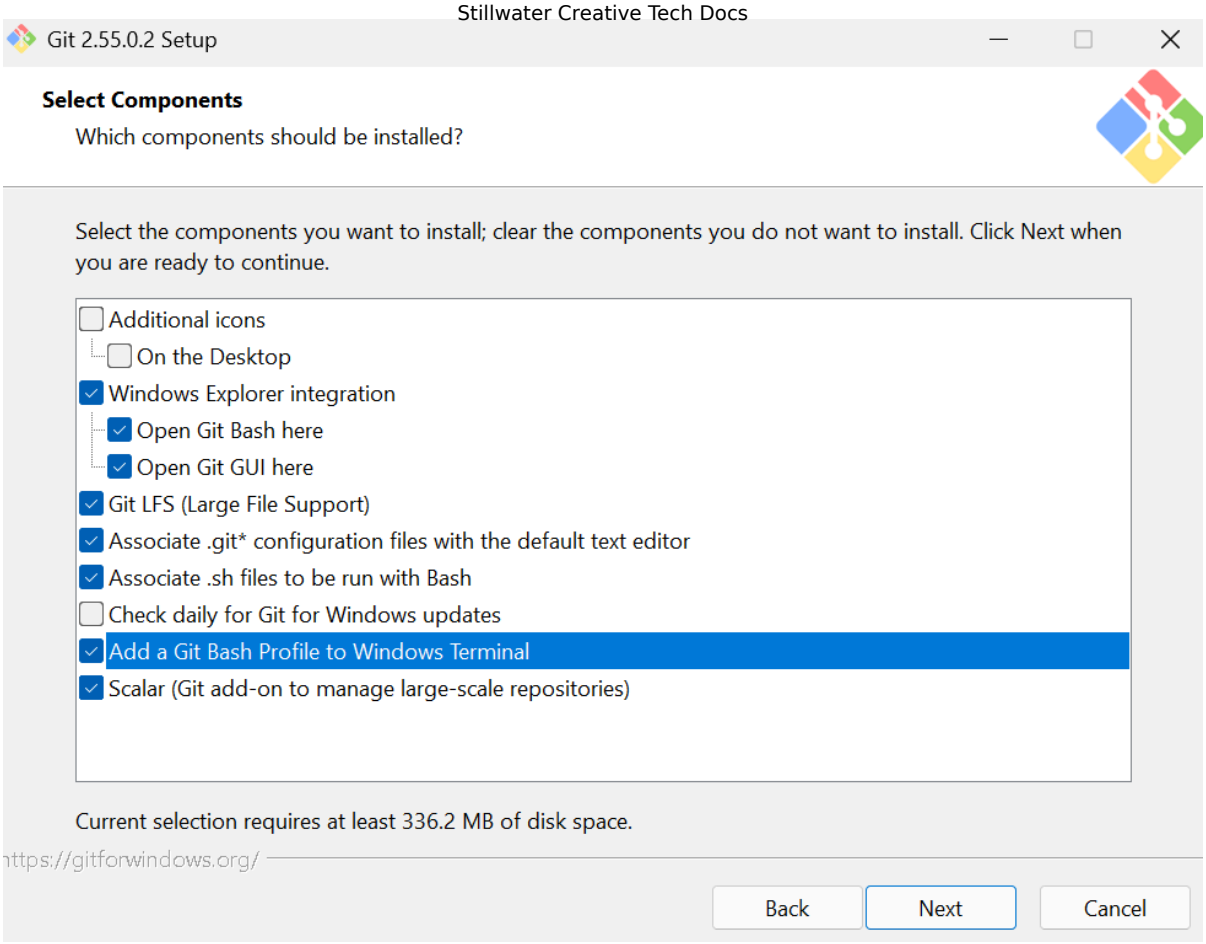


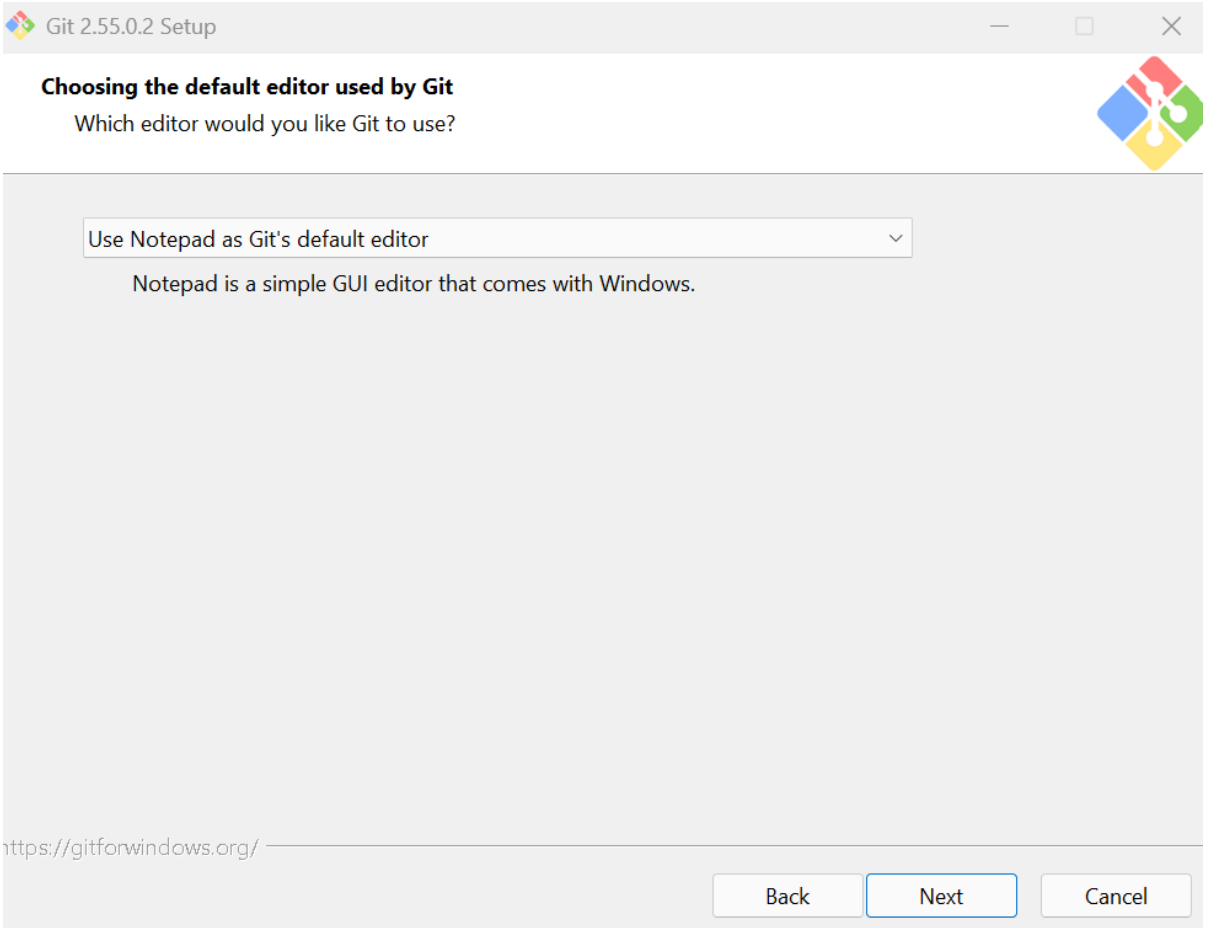
Note on Git files: Git keeps track of changes in a hidden folder called `.git`. Never delete or modify this folder, as it'll break your project history!

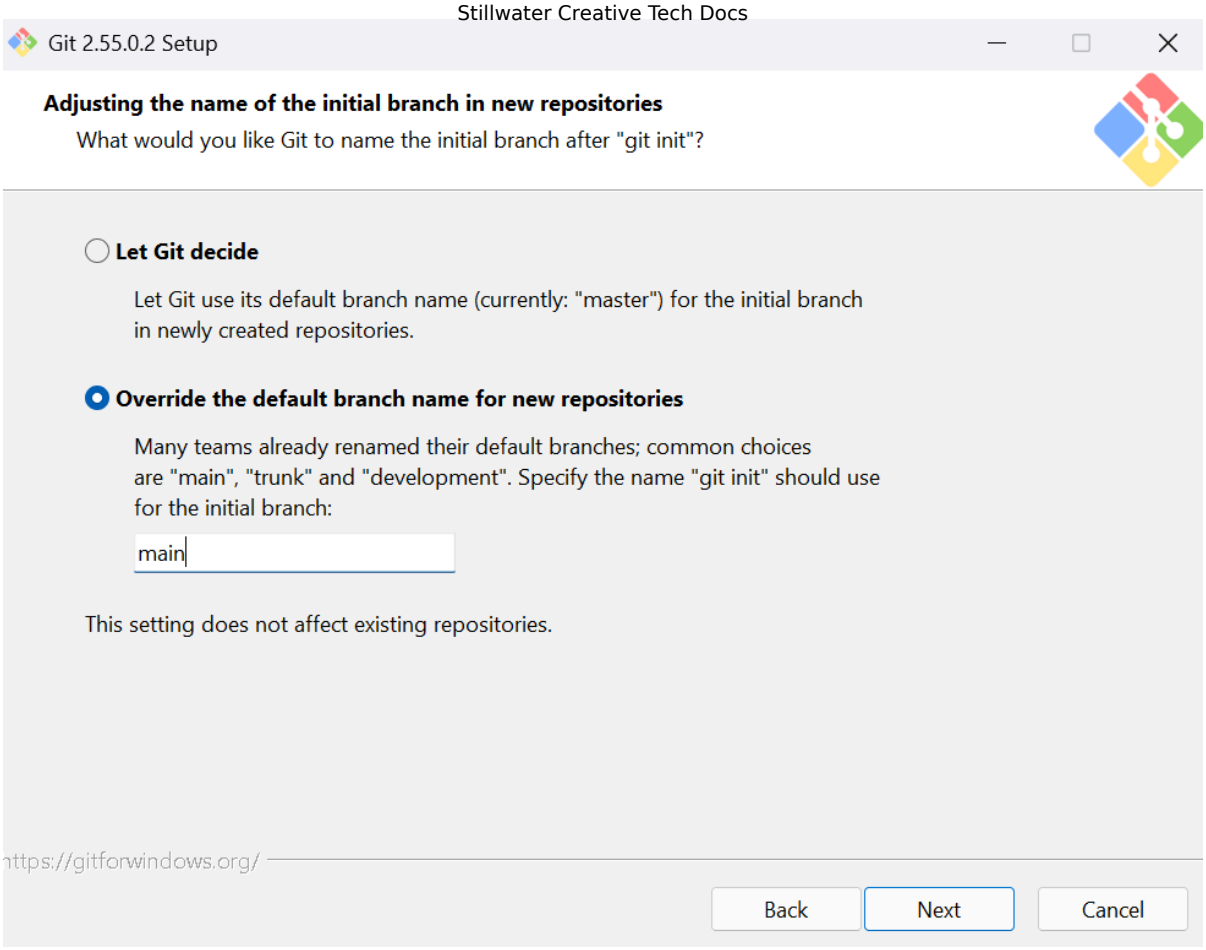
Installing Git

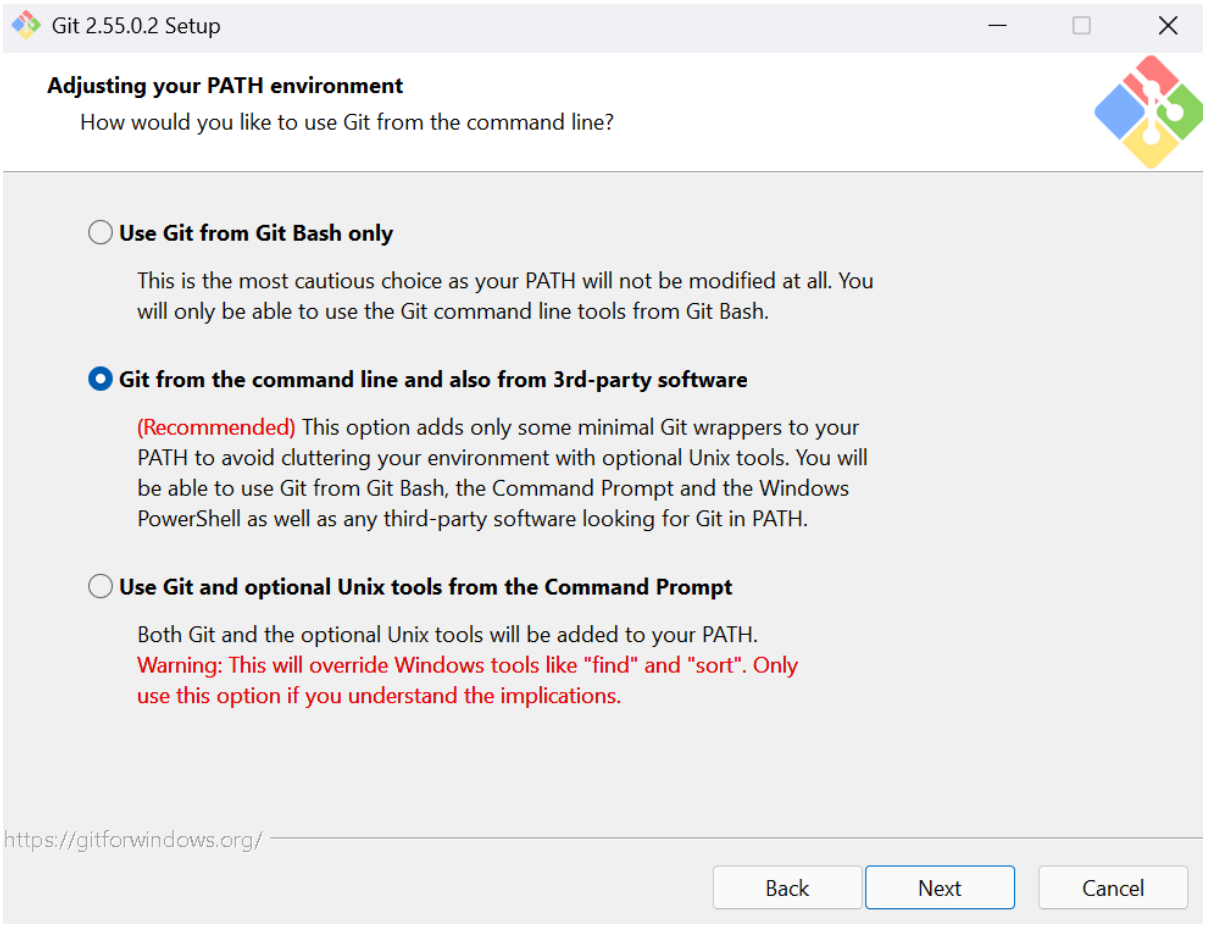
Your first step to setting up your development environment should be to install Git. Visit the download page and download the Git Setup.

> **Note**
>
> Below are a set of recommended options for the Git installer. **All the menus that aren't represented here on this guide should be left with the default option unless you know what you're doing**


Git 2.55.0.2 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
Open Git Bash here
Open 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
Add a Git Bash Profile to Windows Terminal
Scalar (Git add-on to manage large-scale repositories)
Current selection requires at least 336.2 MB of disk space.
https://gitforwindows.org/
Back
Next
Cancel


Git 2.55.0.2 Setup
Choosing the default editor used by Git
Which editor would you like Git to use?
Use Notepad as Git's default editor
Notepad is a simple GUI editor that comes with Windows.
https://gitforwindows.org/
Back
Next
Cancel


Git 2.55.0.2 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.
Override the default branch name for new repositories
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.
https://gitforwindows.org/
Back
Next
Cancel


Git 2.55.0.2 Setup
Adjusting your PATH environment
How would you like to use Git from the command line?
Use Git from Git Bash only
This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.
Git from the command line and also from 3rd-party software
(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.
Use Git and optional Unix tools from the Command Prompt
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.
https://gitforwindows.org/
Back
Next
Cancel

Git vs. GitHub

The two are confused some times, but they are very different:

- **Git** is the tool: a program installed on your computer that tracks the changes in your files.
- **GitHub** is the destination: a remote hosting service. It acts as a central hub where we back up the Unreal project and sync everyone’s updates.

The Visual Workflow (GitHub Desktop)

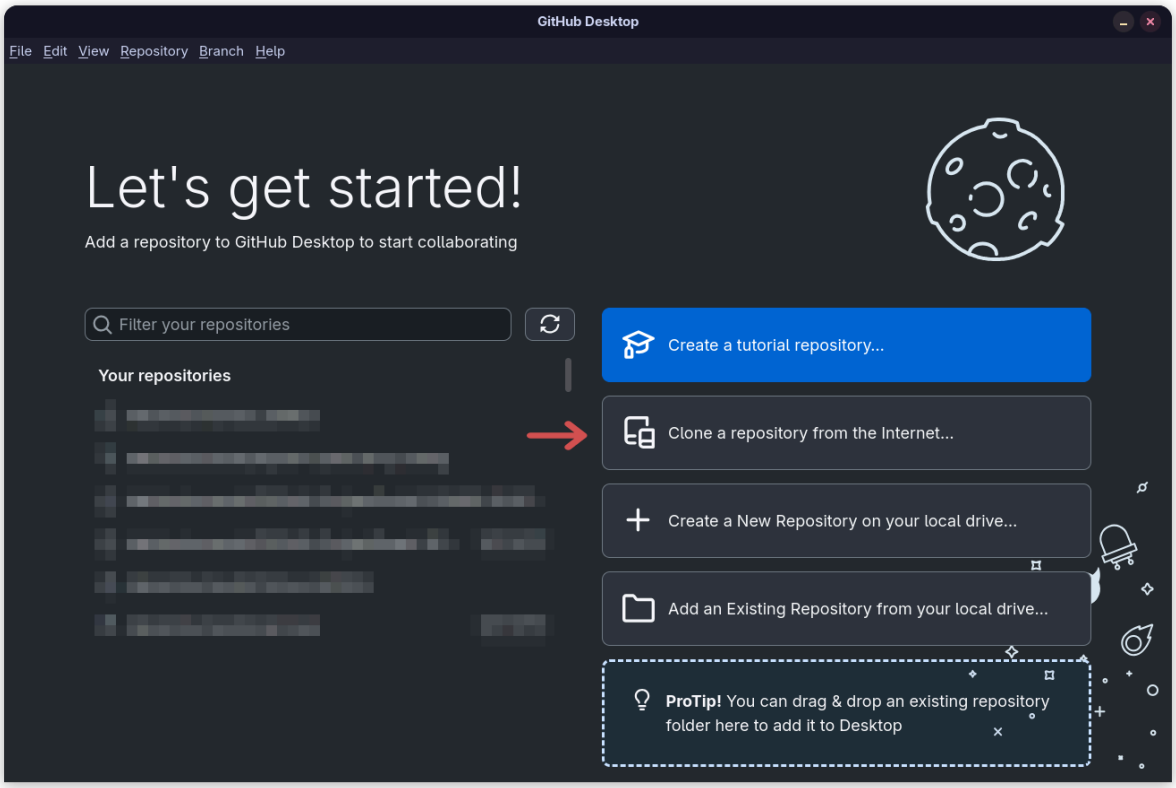
While Git is often associated with the command line, graphical interfaces like **GitHub Desktop** (or the Git interface in your code editor of choice) make the workflow more intuitive.

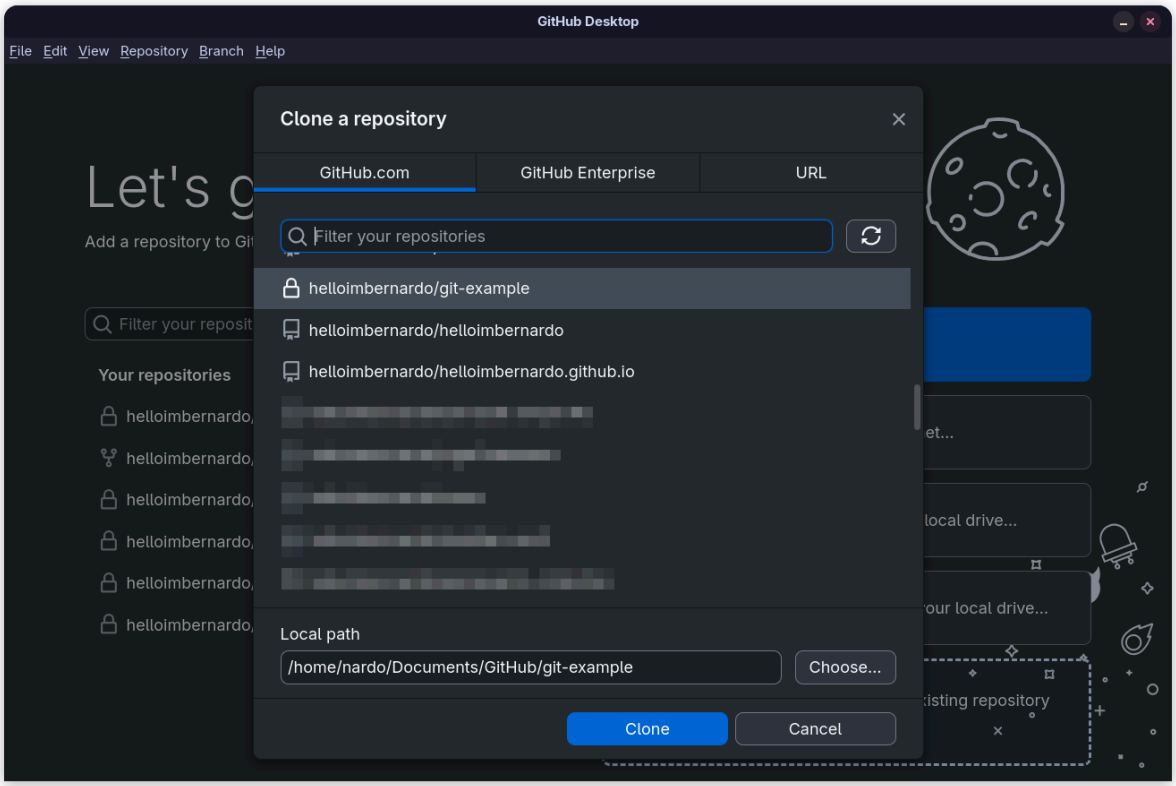
Installing GitHub Desktop

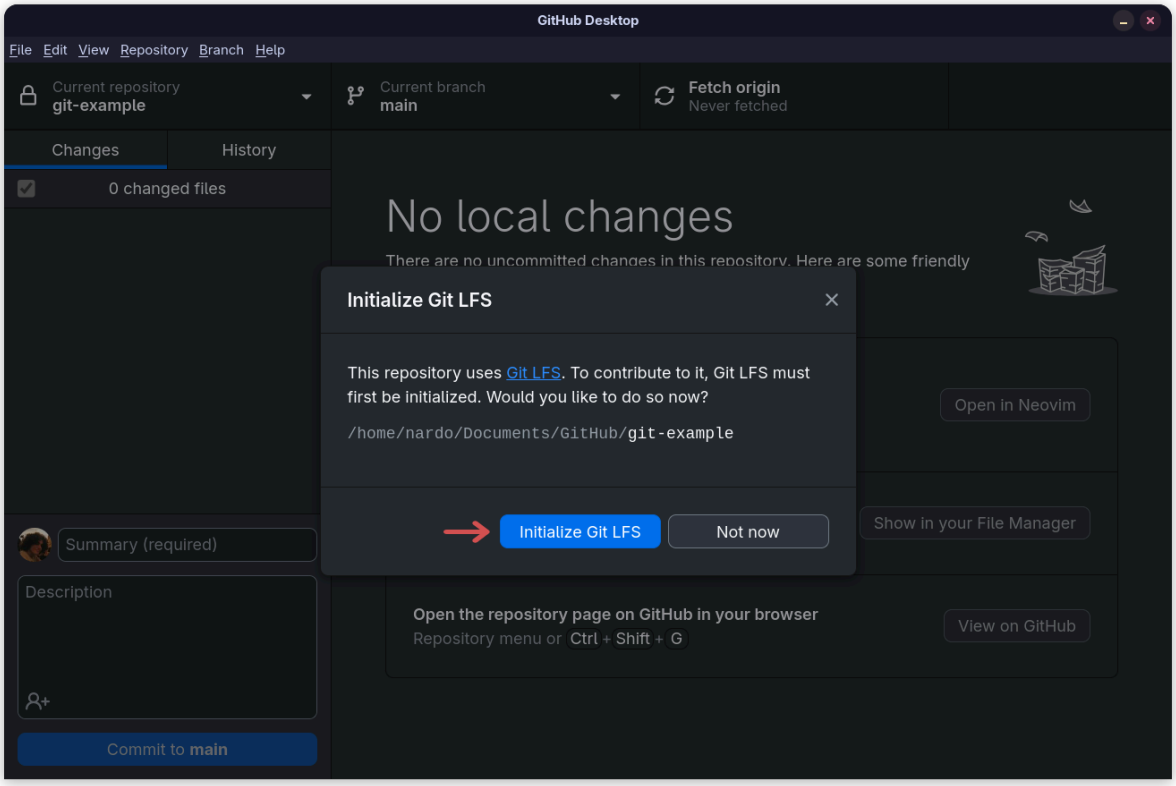
To download GitHub Desktop you can visit the the official website and download and execute the installer.

Cloning: Getting the repository

The first step is getting the repository from the internet: we call this **Cloning**. After logging into GitHub on the program, select "*Clone a repository from the internet...*", select the correct repository and finally "*Initialize Git LFS*".



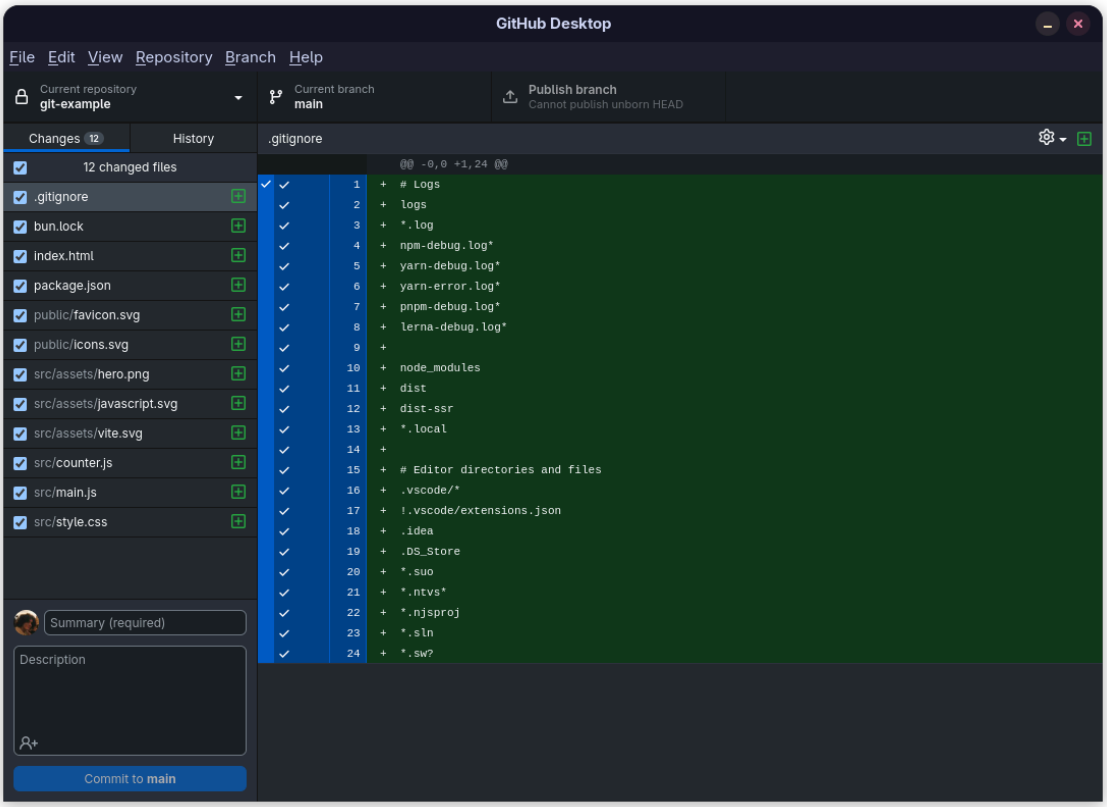




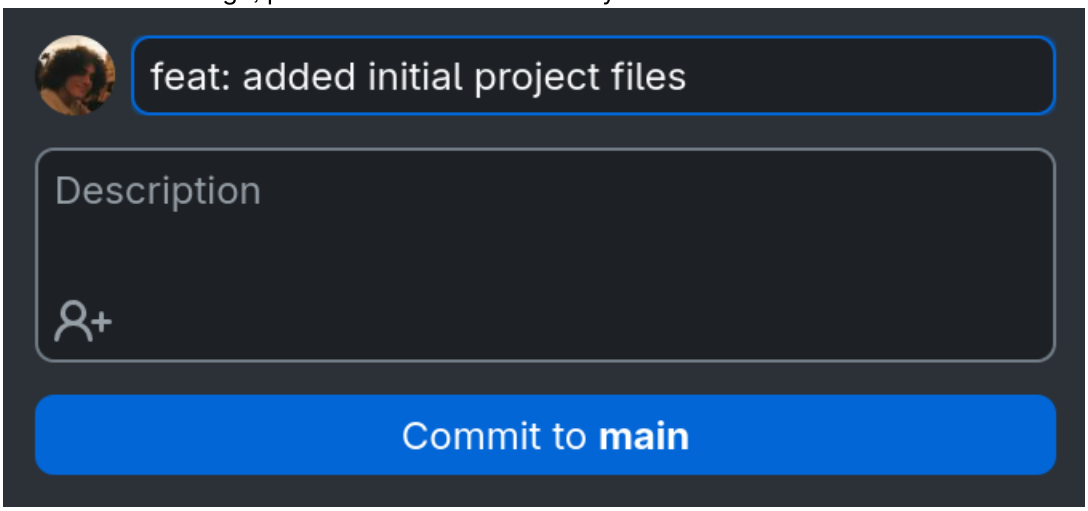
Daily Workflow

To start working, there are three main steps you need to remember:

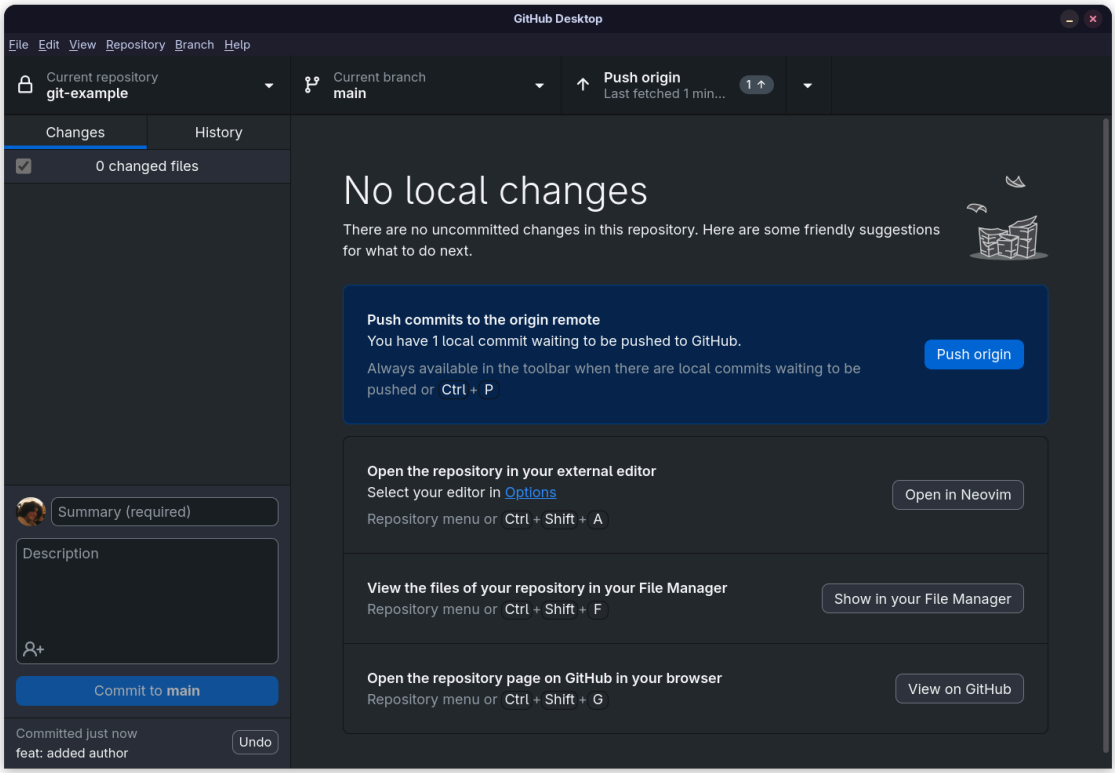
1. **Staging:** As you edit files, the GUI will show which files have changed. You simply check the boxes next to the files you want to group together for your next commit.



2. **Committing:** Once your files are selected, you write a short summary of what you did and click **Commit to main**. This saves the snapshot permanently to your local history. When writing the commit message, please remember to abide by the Commit Standards.



3. **Pushing:** Finally, clicking **Push origin** sends your local commits up to GitHub, syncing your work with the rest of the team.


GitHub Desktop
File Edit View Repository Branch Help
Current repository
git-example
Current branch
main
Push origin
Last fetched 1 min...
1
Changes
History
0 changed files
Summary (required)
Description
Commit to main
Committed just now
feat: added author
Undo
No local changes
There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.
Push commits to the origin remote
You have 1 local commit waiting to be pushed to GitHub.
Always available in the toolbar when there are local commits waiting to be pushed or Ctrl + P
Push origin
Open the repository in your external editor
Select your editor in Options
Repository menu or Ctrl + Shift + A
Open in Neovim
View the files of your repository in your File Manager
Repository menu or Ctrl + Shift + F
Show in your File Manager
Open the repository page on GitHub in your browser
Repository menu or Ctrl + Shift + G
View on GitHub

A Note on The Workflow

> **Caution**
>
> Important note on committing files: In Git there's no way of knowing if someone is working on the same file as you so for any non-text files **make sure you coordinate with others** so that you don't overwrite eachother's work. **Only one person should be working on any given (non-text) file at any given time!**

What Not to Track (.gitignore)

Every time you open Unreal or compile code, it generates temporary files. If we sync these, our repository will become bloated and broken. Our project includes a hidden file called `.gitignore`, which automatically hides these folders from GitHub Desktop.

Never force-commit files from these folders:

- `DerivedDataCache/`
- `Intermediate/`
- `Saved/`
- `Binaries/`

Handling Large Files: Git LFS

Git is great for text (like code or markdown), but it struggles with large binary files. Every time you update a high-res image, PDF document or audio file, Git saves a completely new copy, which can quickly bloat the repository.

To solve this, we use **Git LFS** (Large File Storage). It replaces the heavy files in your repository with a reference to a remote location, while storing the actual file contents on GitHub's servers.

Setting Up LFS for a New File Type

GitHub Desktop handles all the heavy lifting behind the scenes, but if you introduce a brand-new file format to the project (for example, Photoshop documents `.psd`), you need to tell Git to track it.

Because GitHub Desktop runs on top of standard Git, the cleanest way to register a new format is a quick one-time terminal command inside your project folder:

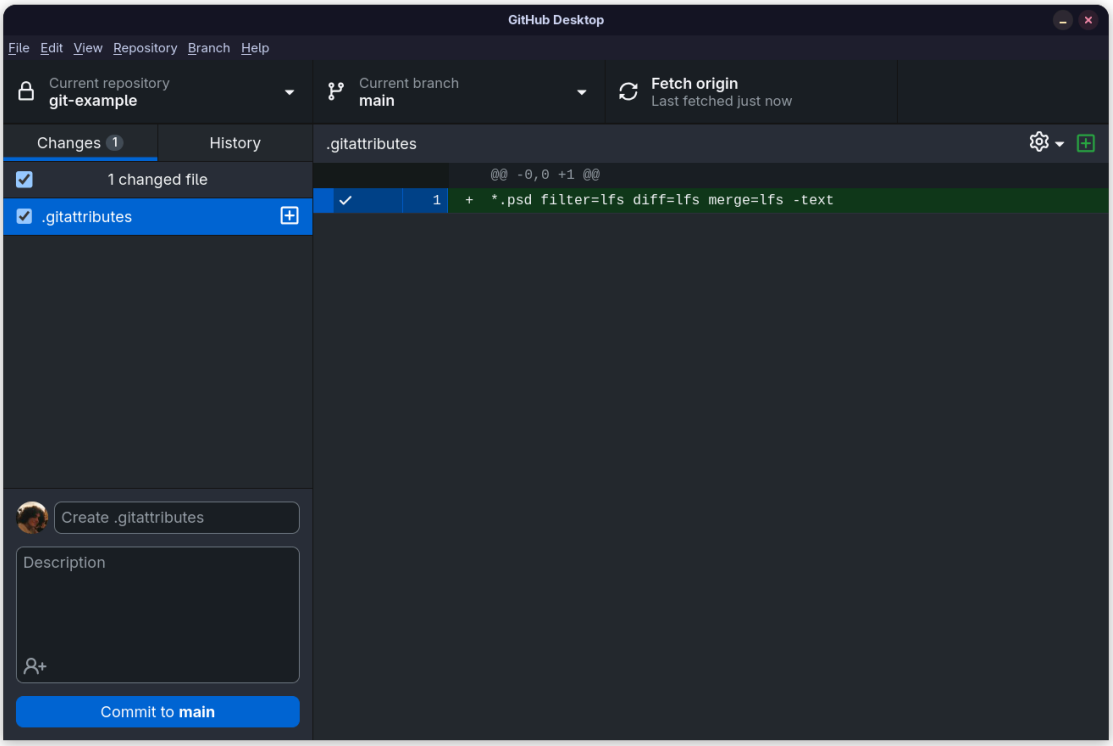
```
git lfs track "*.psd"
```

This command creates or updates a hidden configuration file in your main directory called `.gitattributes`.

Guaranteeing it Works in GitHub Desktop

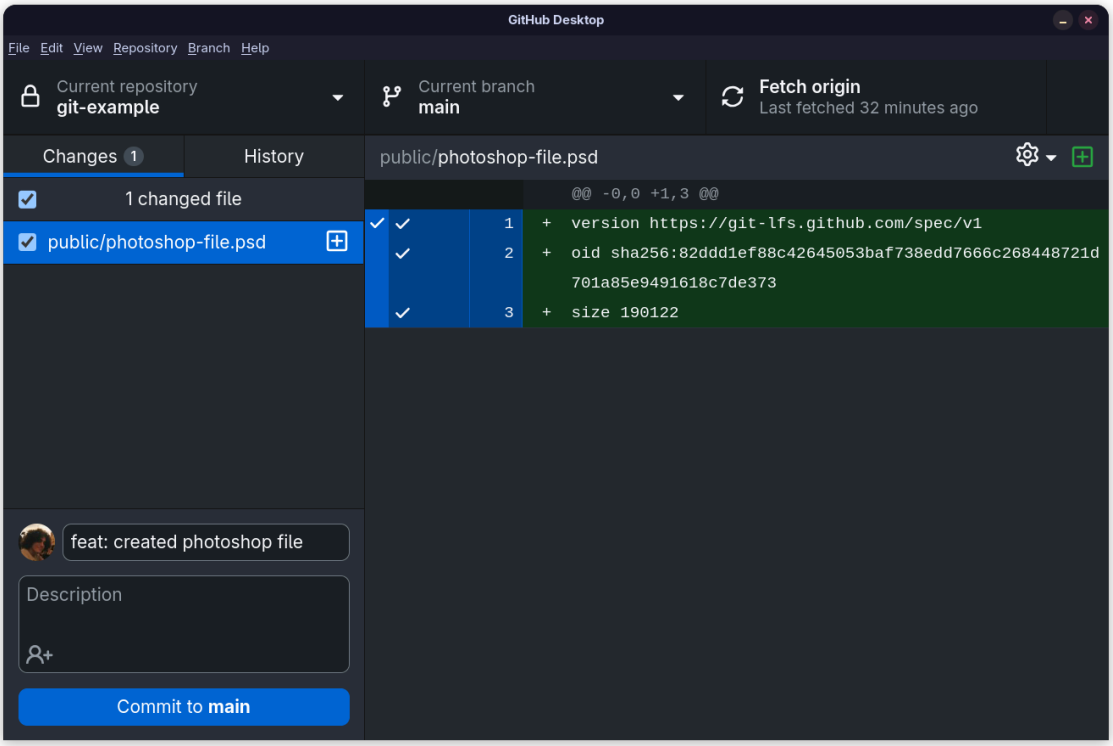
For GitHub Desktop to manage your large files, it requires that `.gitattributes` file. To ensure everything works smoothly, follow these rules:

1. **Commit the Attributes File First:** Whenever you track a new file type, GitHub Desktop will show that `.gitattributes` has been modified. **Always commit and push this change to GitHub before adding your large files.**



2. **Add Your Files Normally:** Once the file extension is registered in `.gitattributes`, you don't have to do anything special. Just drop your large images or assets into your project folders.

3. **Verify in GitHub Desktop:** When you open GitHub Desktop, look at the changed files list. You can proceed with your standard workflow (Staging → Committing → Pushing). The app will automatically read the attributes file, recognize the extension, convert the file to an LFS pointer, and upload the heavy asset in the background.



> **Caution**

> Note on Unreal Assets: Because `.uasset` and `.umap` files are binary, Git cannot merge them. If two people edit the same Blueprint or Level at the same time, one person's work will overwrite the other's. Always communicate with the team before modifying core assets!

Commit Standards for This Project

To keep our project history clean, searchable, and easy to read, we follow the **Conventional Commits** specification.

This is a lightweight standard that adds a predictable prefix to your commit messages. Anyone scanning the history can instantly understand what kind of work was done.

> **Note**
>
> It is recommended you follow the source specification, as this section is merely a short introduction/cheatsheet.

A standard commit message looks like this:

```
type(optional-scope): short description
```

Here are the primary types you will use for this project:

- **feat:** A new feature, mechanic, or major asset addition (e.g., `feat: add double jump to player character`).
- **fix:** A bug fix, crash resolution, or visual correction (e.g., `fix: resolve collision issue on bridge mesh`).
- **docs:** Changes purely to documentation (e.g., `docs: update the Git LFS instructions`).
- **chore:** Routine tasks, project setting tweaks, or engine upgrades (e.g., `chore: update project to Unreal 5.3`).

> **Tip**
>
> Always write your description in the imperative (like giving an order to the engine). Use *"fix lighting"* instead of *"fixed lighting"* or *"fixing lighting"*.