GitHub Desktop Guide

Installing GitHub Desktop

Go to GitHubDesktop and click "Download now" → "Download for Windows (64bit)". Run the downloaded executable by double-clicking it. Select "Sign in to GitHub.com". After signing in successfully, click "Authorize Desktop". Then select "Use my GitHub account name and email address" and click "Finish".

What is GitHub Desktop?

It's a visual tool that lets you work with Git repositories without typing commands in a terminal. It runs Git under the hood, so you don't need to install Git manually. However, it has the typical limitations of graphical interfaces compared to using Git directly.

What is the Remote Repository?

The remote repository is the version of your project that is stored on GitHub (the web). It acts as the central location where all team members can push their changes and pull updates from others.

What is the Local Repository?

The local repository is the version of your project that exists on your own computer. It is a full copy of the remote repository where you can make changes, create branches, and commit your work before pushing it back to GitHub.

Can I work locally without Git and push directly to GitHub?

Yes, you can. You can edit files on your computer and then upload them manually through the GitHub web interface. However, this method does not track changes, manage branches, or handle merges, so it is not recommended for team projects. Even for solo projects, using a local Git repository (with GitHub Desktop or Git) is safer and allows you to maintain a proper version history.

What does “Clone Repository” mean?

To “clone a repository” means to make a full copy of a remote GitHub repository on your local computer. This copy includes all the files, folders, and version history. Once cloned, you can edit files, create branches, commit changes, and later push your updates back to GitHub.

Does cloning a remote repository locally mean Git manages version control locally?

Yes. When you clone a repository, Git creates a local copy of the remote repository on your computer. This local repository tracks all changes, allows you to commit updates, create branches, and manage versions before pushing them back to GitHub. In other words, Git handles version control both locally and in coordination with the remote repository.

Cloning a GitHub Repository Locally

Before cloning a repository, you must accept the invitation to collaborate on GitHub. Then:

1) Open GitHub Desktop.

2) Go to File → Clone Repository → GitHub.com. You will see a list of remote repositories associated with your account. If you accepted the invitation, the project will appear in this list, select it.

3) Do not click Clone yet. First, select the folder where your local copy will be stored. Avoid cloud-synced folders (OneDrive, Google Drive) or restricted Windows folders (Documents, Desktop, Downloads, etc.), as permission issues may occur.
<u>Recommended path example:</u>
C:\Users\YourUserName\Deree_Projects

4) After selecting the folder, click Clone. GitHub Desktop will create a local repository containing all files, folders, and version history.

5) <u>The path to your project after cloning will be:</u>
C:\Users\YourUserName\Deree_Projects\TheNameOfTheRemoteRepository
So, you do not need to manually create a project folder, as GitHub Desktop automatically creates one using the repository name.

Git now manages version control locally, allowing you to safely commit changes, create branches, and push updates back to GitHub.

Selecting the Correct Branch

It is not recommended to work directly in the main branch. The next step is to switch to your personal branch. Click the “Current branch” dropdown at the top of the window in GitHub desktop and select the branch with your name. Once you switch, your local branch is linked to the corresponding branch on GitHub (the origin). Any changes you commit and push will go only to your branch, leaving main untouched. This ensures safe collaboration and version control.

Committing Changes

After editing files in your local repository, GitHub Desktop will display the changed files in the left panel. Write a clear commit message describing your changes. Click Commit to [your branch] (located at the bottom left of the screen) to save your changes locally.

It’s recommended to commit often, with small, meaningful changes rather than one large commit. You can be confident that you are committing to the correct branch because the commit button displays the branch name you are working on. Always double-check the branch name before committing to avoid accidentally committing to main.

Pushing Changes to Remote

After committing, click Push origin to upload your changes to GitHub. This makes your work visible to collaborators and backed up on the remote.

Pulling Changes from the Remote Repository Locally

If someone else makes changes in the remote repository (on GitHub), you can bring those changes into your local repository using GitHub Desktop. To do this, open GitHub Desktop and make sure you are on the branch you want to update. Look near the top right of the window, next to Current Branch. If there are new changes on the remote, you will see a “Pull origin” button with a downward arrow. Click Pull origin. GitHub Desktop will download the changes from the remote branch and merge them into your local branch. Always pull changes before starting new work to avoid conflicts and ensure your local repository is up to date with the remote.

Stashing Changes

Stash can be used if someone wants to temporarily save local changes without committing, for example when they need to switch to another branch to check something or apply a quick fix.
For instance, if you are working on a feature in your branch but need to switch to main or another branch to review or fix something. Instead of committing unfinished work, you can stash it, switch branches, and then come back to apply your changes.
However, stashed changes are not committed and are easy to lose if you accidentally overwrite the stash or close GitHub Desktop incorrectly. Conflicts can occur when applying a stash if the branch has changed since you stashed your work. In general, frequent commits are safer. Therefore, commit small changes often instead of relying on stash whenever possible. Use stash only when is absolutely necessary to temporarily save work.

Final Tips

- Avoid committing directly to main. Always check the branch you are on before starting work. If you accidentally start working on main, you can use stash to temporarily save your changes, switch to the correct branch, and then bring the changes to the new branch. I recommend to avoid it though.
- Pull changes from the remote before starting work. This ensures your local repository is up to date and reduces the risk of merge conflicts with other collaborators.
- Commit often, don’t rely on stashing. Frequent, small commits make it easier to track changes, roll back mistakes, and collaborate safely. Only use stash when you need to switch branches temporarily without committing.
- Write meaningful commit messages. Provide a clear summary and, for major changes, add a description. This helps collaborators understand your changes and makes the project history easier to follow.
- Be extremely careful with conflicts or stashed changes. Do not proceed until you understand and resolve the issue. Mistakes can make things worse, and if Git is not installed locally, recovering files can be very difficult. If you encounter something unexpected, take a moment to investigate, ask for help, or even use AI tools to understand what is happening before continuing. Safety comes first.

- You won’t need to perform any merges or create pull requests while working on your personal branch. The only actions you need are pulling changes from your remote branch to update your local copy and pushing your local commits to the remote branch on GitHub.

Commonly Confusing Vocabulary

- **Remote / Origin** = The GitHub repository online (central repository).
- **Local** = Your copy of the repository on your computer (Windows Explorer project directory).
- **Pull / Fetch from origin** = Bring changes from the remote repository to your local branch. This does not create a pull request; it updates your local branch with remote changes.
- **Push to origin** = Upload your local commits to the remote repository so others can see your changes.
- **Merge / Pull Request** = Combine changes from one branch into another (e.g., your branch into main). This is usually done on GitHub through a pull request.