A Comprehensive Guide to GitHub

Updated 06/20/2025

# Table of Contents

[Table of Contents 1](#_Toc201275722)

[1. Introduction to GitHub 3](#_Toc201275723)

[1.1 Key Concepts 3](#_Toc201275724)

[1.2 Public vs. Private Repositories 3](#_Toc201275725)

[2. Creating a Repository 3](#_Toc201275726)

[2.1 GitHub Website 3](#_Toc201275727)

[2.2 GitHub Desktop 4](#_Toc201275728)

[2.3 Visual Studio Code (VS Code) 4](#_Toc201275729)

[3. Making Changes: Push & Pull 4](#_Toc201275730)

[3.1 Using GitHub Website 4](#_Toc201275731)

[3.2 Using GitHub Desktop 5](#_Toc201275732)

[3.3 Using VS Code 5](#_Toc201275733)

[4. Branching & Merging Changes 5](#_Toc201275734)

[4.1 Using GitHub Website 6](#_Toc201275735)

[4.1a Creating a Branch 6](#_Toc201275736)

[4.1b Merging Changes 6](#_Toc201275737)

[4.2 Using GitHub Desktop 6](#_Toc201275738)

[4.2a Creating a Branch 6](#_Toc201275739)

[4.2b Merging Changes 6](#_Toc201275740)

[4.3 Using Visual Studio Code 7](#_Toc201275741)

[4.3a Creating a Branch 7](#_Toc201275742)

[4.3b Merging Changes 7](#_Toc201275743)

[4.4 Best Practices 7](#_Toc201275744)

[5. Working with Public vs Private Repositories 7](#_Toc201275745)

[5.1 Using GitHub Website 7](#_Toc201275746)

[5.2 Using GitHub Desktop 8](#_Toc201275747)

[5.3 Using Visual Studio Code 8](#_Toc201275748)

[6. Understanding and Using .gitignore 9](#_Toc201275749)

[6.1 What is .gitignore? 9](#_Toc201275750)

[6.2 How Does .gitignore Work? 9](#_Toc201275751)

[6.3 Creating and Configuring .gitignore 9](#_Toc201275752)

[6.4 Common Use Cases 10](#_Toc201275753)

[6.5 Using Predefined Templates 10](#_Toc201275754)

[6.6 Best Practices for .gitignore 10](#_Toc201275755)

[6.7 Checking .gitignore 10](#_Toc201275756)

# 1. Introduction to GitHub

GitHub is a cloud-based platform for version control and collaboration, ideal for tracking and managing code changes. It allows multiple people to work on projects at once using Git, a distributed version control system.

## 1.1 Key Concepts

* **Repository** (repo): A project folder hosted on GitHub.
* **Clone**: Downloading a copy of a repository to your local machine.
* **Commit**: Saving changes to your local repository.
* **Push**: Uploading local commits to GitHub.
* **Pull**: Downloading changes from GitHub to your local repo.
* **Branch**: A parallel version of the repository.
* **Fork**: A copy of someone else's repository under your GitHub account.

## 1.2 Public vs. Private Repositories

| Type | Visibility | When to Use |
| --- | --- | --- |
| Public | Anyone can see | Open-source projects, portfolios |
| Private | Restricted access | Proprietary code, private teams |

*Tip: You can convert a repo between public/private under repo Settings.*

# 2. Creating a Repository

Creating a repository is a fundamental step in version control. Here are detailed instructions to create a repository using three tools: GitHub website, GitHub Desktop, and Visual Studio Code (VS Code).

## 2.1 GitHub Website

* Log in to your GitHub account at [github.com].
* Click the New button in the upper-right corner of the page or navigate to [github.com/new].
* Enter a name for your repository in the Repository name field. Make sure the name is unique within your account.
* Optionally, add a description for the repository in the Description field.
* Choose the repository’s visibility:
* Public: Anyone can see your repository.
* Private: Only you and collaborators can access the repository.

Optionally, initialize the repository with:

A README file: Provides an overview of your repository.

A .gitignore template: Specifies untracked files to ignore.

A license: Specifies the terms under which the code can be used.

Click Create repository to finalize.

## 2.2 GitHub Desktop

* Open GitHub Desktop and sign in to your GitHub account.
* Click File in the menu bar and select New repository....
* In the dialog box, fill in the following:
* Repository name: Enter a name for your repository.
* Description (optional): Add details about the repository.
* Local path: Choose where the repository will be saved on your computer.

Check Initialize this repository with a README if you want a README file created automatically.

Click Create Repository. Your repository will be created locally and can be published to GitHub using the Publish repository button.

## 2.3 Visual Studio Code (VS Code)

* Open Visual Studio Code and ensure Git is installed and properly configured.
* Click on the Source Control icon in the Activity Bar on the left-hand side.
* Click Initialize Repository in the Source Control view to create a new local Git repository.
* Open the Command Palette (Ctrl+Shift+P or Cmd+Shift+P on macOS) and type Git: Publish to GitHub. Select this option.
* If you're not signed in to GitHub, follow the prompts to authenticate.
* Enter a name for your repository when prompted and select the visibility (Public or Private).
* Click Publish to create the repository on GitHub and push any existing files.

# 3. Making Changes: Push & Pull

## 3.1 Using GitHub Website

* Making Changes:
  + Navigate to your repository on GitHub.
  + Find the file you want to edit and click on it.
  + Select the pencil icon to edit the file directly.
  + After making changes, include a meaningful commit message explaining what was modified and click "Commit changes."
  + For creating new files, click "Add File" and select "Create new file."
* Pushing Changes: Changes made directly on the GitHub website are automatically committed to the repository.
* Pulling Changes: To update your local repository with the latest changes made on GitHub, use your terminal or Git GUI tools to run the `git pull origin main` command (replace "main" with your branch name if different).

## 3.2 Using GitHub Desktop

* Making Changes:
  + Open GitHub Desktop and select your repository.
  + Use external tools like a text editor or IDE to make changes to your files locally.
  + After editing, return to GitHub Desktop to view the changes under the "Changes" tab.
* Pushing Changes: In the "Changes" tab, add a descriptive commit message, review the changes, and click the "Commit to [branch name]" button. Then, click "Push origin" to upload your changes to the remote repository.
* Pulling Changes: To sync your local repository with remote updates, click "Fetch origin" and review any incoming changes. If updates exist, click "Pull origin" to integrate them into your local branch.

## 3.3 Using VS Code

* Making Changes:
  + Open the repository folder in VS Code.
  + Edit files as needed.
  + VS Code automatically detects changes, highlighting edited files in the "Source Control" panel.
* Pushing Changes: After finishing edits, navigate to the "Source Control" panel and click the "+" icon to stage your changes. Write a commit message in the text box and click the checkmark icon to commit. Then, click "Sync Changes" or use the terminal to run `git push origin [branch name]` for pushing changes.
* Pulling Changes: To pull remote updates, click "Sync Changes" in the "Source Control" panel or use the terminal to run `git pull origin [branch name]`. This keeps your local repository up to date with the remote repository.

# 4. Branching & Merging Changes

Branching and merging are essential for maintaining a streamlined and collaborative development process by enabling teams to work on different features or bug fixes simultaneously without causing conflicts.

Branching in Git is a feature that allows developers to diverge from the main codebase to independently work on new features, experiments, or bug fixes. By creating branches, contributors can isolate their changes, ensuring that the core project remains stable. Merging, on the other hand, is the process of integrating these changes back into the main branch or another branch. This workflow is crucial for maintaining an organized, collaborative, and conflict-free development process. Developers can experiment freely and only merge code once it has been tested and reviewed, making branching and merging indispensable tools for modern version control systems.

## 4.1 Using GitHub Website

### 4.1a Creating a Branch

* Log in to your GitHub account and navigate to your repository.
* Click on the "Branch" dropdown menu located above the file list.
* Type the name of your new branch in the field provided and click "Create branch".
* Your new branch is now created and selected.

### 4.1b Merging Changes

* Navigate to your repository and click on the "Pull requests" tab.
* Click "New pull request" to start the merge process.
* Ensure the correct branches are selected:
* Base branch: The branch into which you want to merge changes (usually 'main' or 'master').
* Compare branch: The branch containing the changes you want to merge.

Review the changes and add a descriptive title and comments for the pull request.

Click "Create pull request".

Once reviewed by collaborators, click "Merge pull request" and confirm the merge.

## 4.2 Using GitHub Desktop

### 4.2a Creating a Branch

* Open GitHub Desktop and select your repository.
* Click on the "Current Branch" dropdown at the top.
* Click "New Branch".
* Enter a branch name and ensure it is based on the correct branch (typically 'main').
* Click "Create Branch".

### 4.2b Merging Changes

* Switch to the branch you wish to merge into (e.g., 'main') by selecting it from the "Current Branch" dropdown.
* Click "Fetch origin" to ensure you have the latest changes from the remote repository.
* Click "Merge into [branch name]" from the dropdown menu next to "Current Branch".
* GitHub Desktop will automatically merge the changes if there are no conflicts.
* Push the changes to the remote repository by clicking "Push to origin".

## 4.3 Using Visual Studio Code

### 4.3a Creating a Branch

* Ensure Git is installed and VS Code is connected to your repository.
* Open the Source Control panel by clicking the Git icon in the sidebar.
* Click on the branch name in the status bar at the bottom of the window.
* Select "Create new branch" and enter a name for your branch.
* Your workspace will switch to the new branch.

### 4.3b Merging Changes

* Ensure you are on the branch you want to merge into (e.g., 'main').
* Open the terminal in VS Code and fetch the latest changes by running git fetch origin.
* Switch to the branch containing the changes by running git checkout [branch name].
* Merge the changes into your current branch by running git merge [branch name].
* If there are conflicts, resolve them manually in the editor. Once resolved, stage the changes using git add ., commit using git commit -m "Resolved merge conflicts", and push them using git push origin [branch name].

## 4.4 Best Practices

* Always name branches descriptively to reflect the purpose (e.g., 'feature/add-login', 'bugfix/fix-typo').
* Regularly synchronize your branches with the remote repository to prevent conflicts.
* Before merging, ensure all tests pass and the code review process is complete.
* Delete merged branches from the remote repository to declutter your workspace.

# 5. Working with Public vs Private Repositories

Public and private repositories serve different purposes in development workflows. Public repositories are visible to everyone, making them ideal for open-source projects. Private repositories, on the other hand, are restricted to selected collaborators, making them suitable for confidential or collaborative work.

## 5.1 Using GitHub Website

* Creating a Repository:
* - Navigate to your GitHub profile and click on the "+" icon in the top-right corner, then select "New Repository."
* - Name your repository and choose its visibility: Public or Private.
* - Add a README file, .gitignore template, or select a license if needed, then click "Create repository."
* Managing Access:
* - For private repositories, go to "Settings" > "Collaborators and teams" to invite collaborators by entering their GitHub username or email.
* - Public repositories do not require explicit access management; anyone can view and fork them.
* Cloning or Forking:
* - To work locally, clone the repository by clicking on the green "Code" button and copying the HTTPS/SSH link. Use `git clone` followed by the URL in your terminal.
* - Public repositories can also be forked to your account for independent contributions.

## 5.2 Using GitHub Desktop

* Cloning Repositories:
* - Open GitHub Desktop and click "File" > "Clone repository."
* - Select the repository from your list or paste its URL for public repositories. Private repositories require you to sign in to your GitHub account for access.
* Publishing Local Repositories:
* - If you’ve created a local repository, click "Publish repository" in GitHub Desktop.
* - Choose visibility (Public or Private) during the publishing process.
* Collaborating:
* - For private repositories, changes made in GitHub Desktop can be pushed and pulled only by authorized collaborators.
* - In public repositories, your contributions may be visible to all users.

## 5.3 Using Visual Studio Code

* Setting Up:
* - Install the GitHub extension for Visual Studio Code for seamless integration.
* - Sign in to your GitHub account through VS Code.
* Cloning a Repository:
* - Open the Command Palette (`Ctrl+Shift+P` or `Cmd+Shift+P` on Mac), then select "Git: Clone."
* - Enter the repository URL or select from your GitHub account. Private repositories require authentication.
* Creating a Repository:
* - Initialize a local repository in VS Code, then link it to GitHub using the GitHub extension.
* - Choose visibility (Public or Private) when pushing the repository for the first time.
* Collaborating:
* - For private repositories, ensure collaborators have access before pulling or pushing changes.
* - For public repositories, fork the original repository to maintain independent contributions.

Public repositories facilitate transparency and open-source collaboration, while private repositories safeguard sensitive projects and enable controlled teamwork across platforms.

# 6. Understanding and Using .gitignore

When working with Git repositories, the `.gitignore` file is a powerful tool that helps developers manage which files and directories should be excluded from version control. This ensures that sensitive or unnecessary files don’t clutter the repository or compromise its functionality.

## 6.1 What is .gitignore?

A `.gitignore` file is a plain-text document used by Git to determine which files and directories should be ignored during the version control process. Files listed in `.gitignore` will not be tracked by Git, meaning any changes to them won’t be staged, committed, or pushed. This is especially useful for excluding temporary files, build artifacts, secret credentials, or system-specific configurations.

## 6.2 How Does .gitignore Work?

Git reads the rules defined in the `.gitignore` file and applies them to the working directory. Each rule specifies a pattern of files or folders to ignore. For example:

* Temporary files: `\*.tmp`
* Log files: `\*.log`
* Node modules: `node\_modules/`
* Environment files: `.env`

If a file is already being tracked by Git, adding its pattern to `.gitignore` won’t stop Git from tracking it. You will need to remove the file from the repository using `git rm --cached ` for Git to respect the `.gitignore` rule.

## 6.3 Creating and Configuring .gitignore

Creating a `.gitignore` file is simple:

* In the root directory of your repository, create a new file named `.gitignore`.
* Add patterns of files and directories you want Git to ignore.

For example:

```

# Ignore log files

\*.log

# Ignore the .env file

.env

# Ignore the node\_modules folder

node\_modules/

```

## 6.4 Common Use Cases

Developers often use `.gitignore` to:

* Exclude system files like `.DS\_Store` (macOS) or `Thumbs.db` (Windows).
* Ignore build artifacts such as `dist/` or `bin/` folders.
* Protect sensitive information such as API keys stored in `.env` files.

## 6.5 Using Predefined Templates

GitHub offers predefined `.gitignore` templates for various programming languages, frameworks, and tools. These can be a great starting point for your project. For example, a Node.js project template might include:

```

# Logs

\*.log

npm-debug.log\*

# Dependency directories

node\_modules/

```

## 6.6 Best Practices for .gitignore

To maximize the usefulness of `.gitignore`:

* Define patterns early in the project to maintain consistency.
* Use comments to explain why specific files are ignored.
* Regularly review and update the `.gitignore` file as your project evolves.

## 6.7 Checking .gitignore

To verify whether a file is being ignored, you can use:

```

git check-ignore -v

```

This command shows the rule that causes Git to ignore the file.