

Introduction to Git in DevOps:

1. What is Git?

Git is a distributed version control system that helps track changes in source code during software development.

It allows multiple developers to work on a project simultaneously and merge their changes efficiently.

2. Key Concepts:

Repository: A collection of files and version history.

Commit: A snapshot of changes made to the code.

Branch: A separate line of development, often used for feature development or bug fixes.

Merge: Combining changes from one branch into another.

3. Why Git in DevOps?

Enables collaboration among developers and teams.

Facilitates continuous integration and continuous delivery (CI/CD) processes.

Supports rollbacks and version tracking for better release management.

Basic Git Commands:

1. Cloning a Repository:

```
bash
git clone <repository_url>
```

2. Creating a Branch:

```
bash
git branch <branch_name>
```

3. Switching Branches:

```
bash
git checkout <branch_name>
```

4. Making Changes:

```
bash
git add <file_name>
git commit -m "Commit message"
```

5. Merging Branches:

```
Bash
```

```
git checkout <target_branch>  
git merge <source_branch>
```

6. Pushing Changes:

```
bash  
git push origin <branch_name>
```

Advanced Git Topics in DevOps:

1. Git Hooks:

Execute custom scripts at various points in the Git workflow.

2. Git Flow:

A branching model that defines a strict branching structure for project development.

3. GitLab/GitHub Actions:

Automate workflows, build, test, and deploy directly from the Git repository.

4. Git for Infrastructure as Code (IaC):

Managing infrastructure configurations using Git for consistency and versioning.

5. Git Best Practices:

Guidelines for effective collaboration, commit messages, and branching strategies.
Integration with CI/CD:

1. Jenkins and Git Integration:

Automate build and deployment processes using Jenkins and Git.

2. GitHub/GitLab CI/CD Pipelines:

Define and manage CI/CD workflows within the version control system.

3. Automated Testing with Git:

Integration of testing frameworks with Git for continuous testing.

Troubleshooting and Maintenance:

1. Git Log and History:

Analyzing commit history for debugging and auditing.

2. Resolving Merge Conflicts:

Strategies to resolve conflicts when merging branches.

3. Git Maintenance Commands:

Cleaning up and optimizing the Git repository.

---> **Steps For Installing Git for Windows:**

Installing Git prompts you to select a text editor. If you don't have one, we strongly advise you to install prior to installing Git. Our roundup of the [best text editors for coding](#) may help you decide.

Note: If you are new to Git, refer to our post [How Does Git Work](#) to learn more about Git workflow and Git functions.

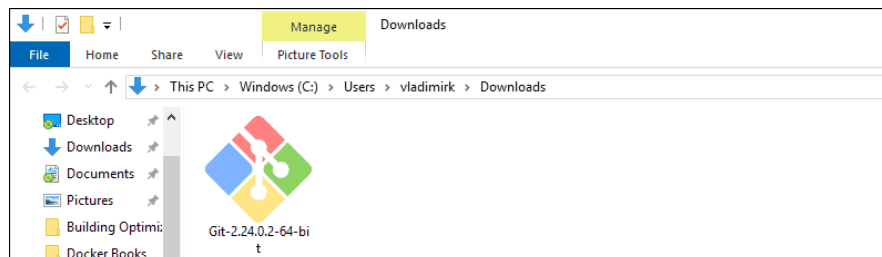
➤ Download Git for Windows

1. Browse to the official Git website: <https://git-scm.com/downloads>
2. Click the download link for Windows and allow the download to complete.

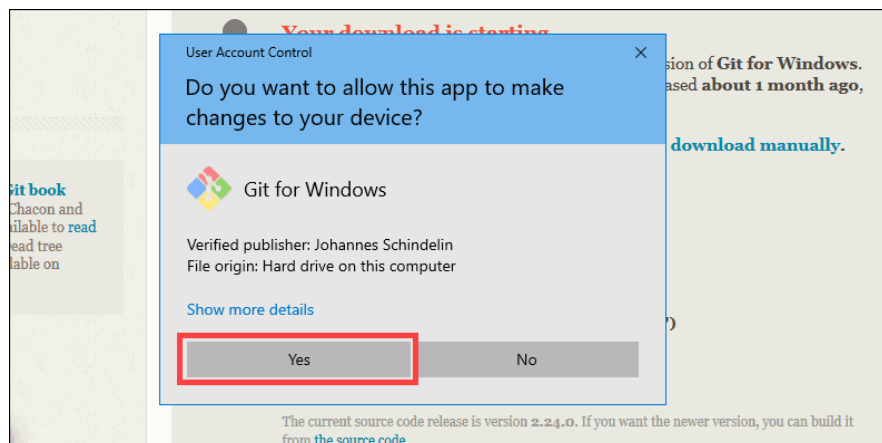


➤ Extract and Launch Git Installer

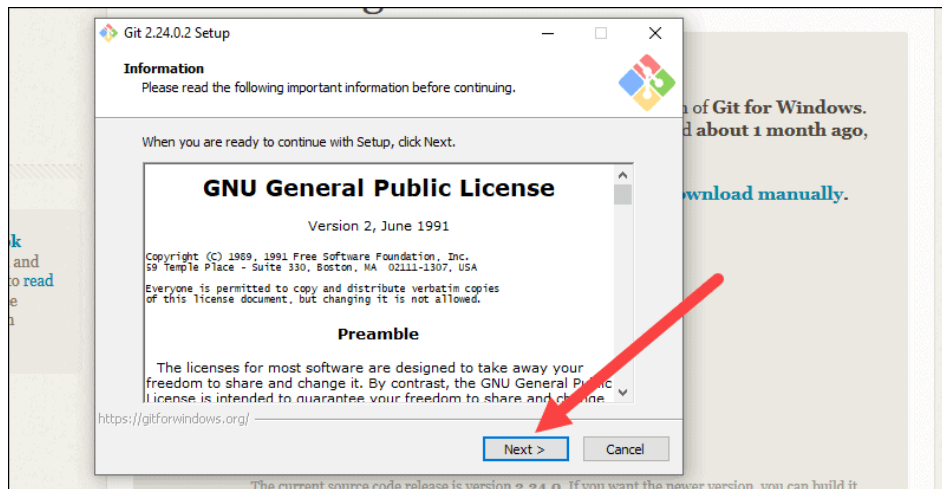
3. Browse to the download location (or use the download shortcut in your browser). Double-click the file to extract and launch the installer.



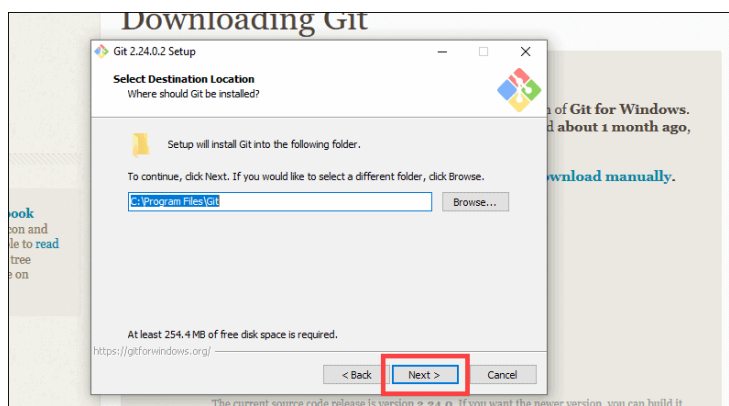
4. Allow the app to make changes to your device by clicking **Yes** on the User Account Control dialog that opens.



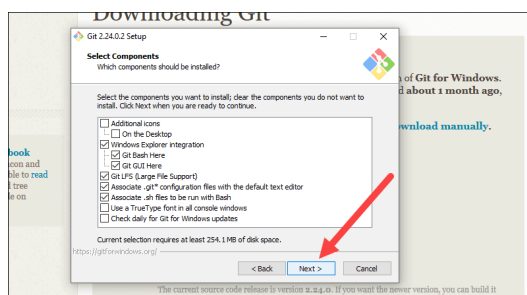
5. Review the [GNU General Public License](#), and when you're ready to install, click **Next**.



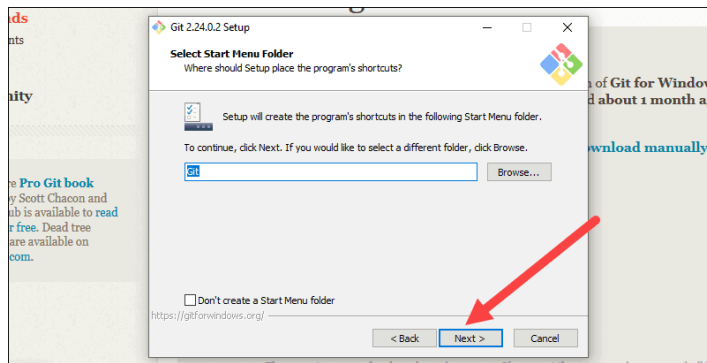
6. The installer will ask you for an installation location. Leave the default, unless you have reason to change it, and click **Next**.



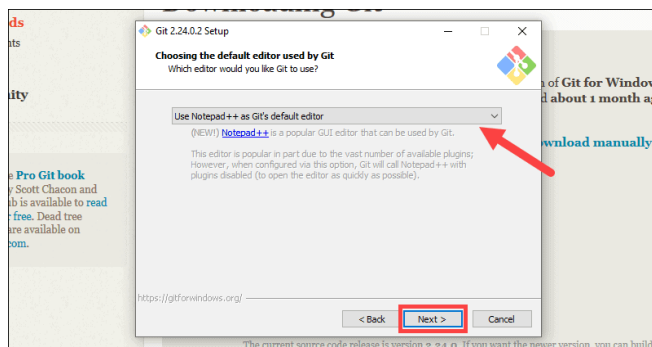
7. A component selection screen will appear. Leave the defaults unless you have a specific need to change them and click **Next**.



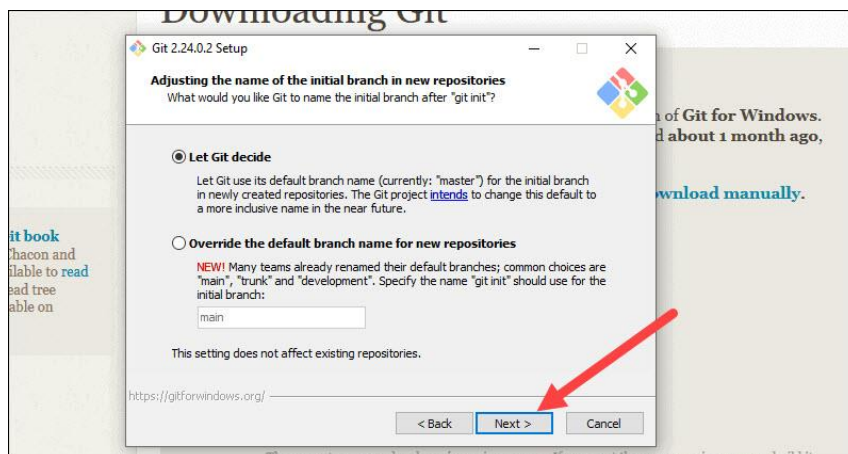
8. The installer will offer to create a start menu folder. Simply click **Next**.



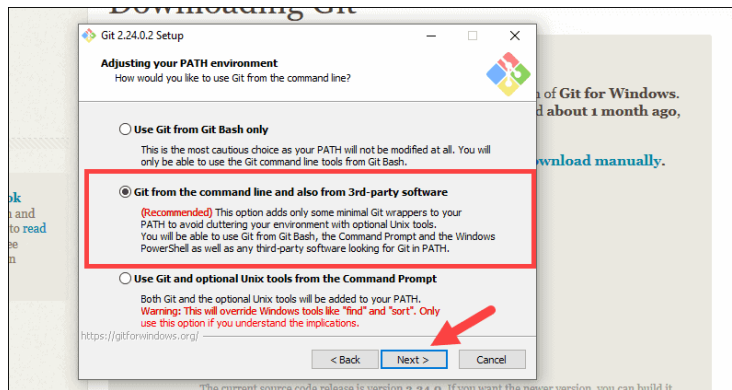
9. Select a text editor you'd like to use with Git. Use the drop-down menu to select Notepad++ (or whichever text editor you prefer) and click **Next**.



10. The next step allows you to choose a different name for your initial branch. The default is 'master.' Unless you're working in a team that requires a different name, leave the default option and click **Next**.

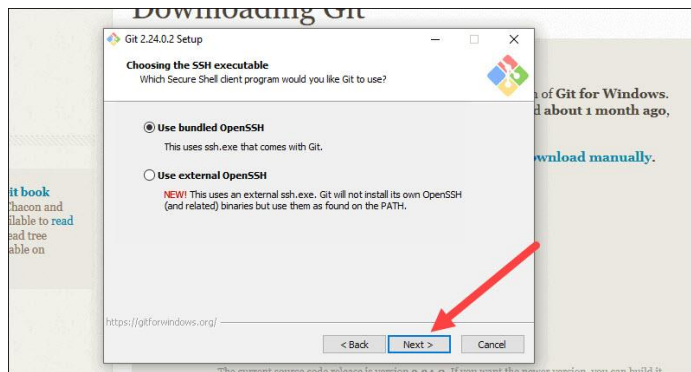


11. This installation step allows you to change the **PATH environment**. The **PATH** is the default set of directories included when you run a command from the command line. Leave this on the middle (recommended) selection and click **Next**.

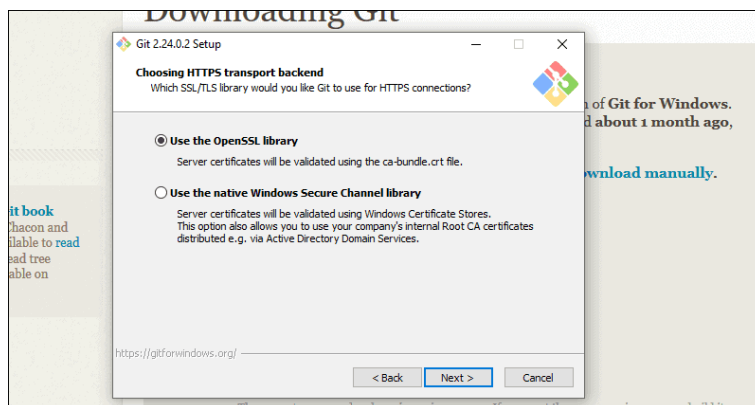


➤ Server Certificates, Line Endings and Terminal Emulators:

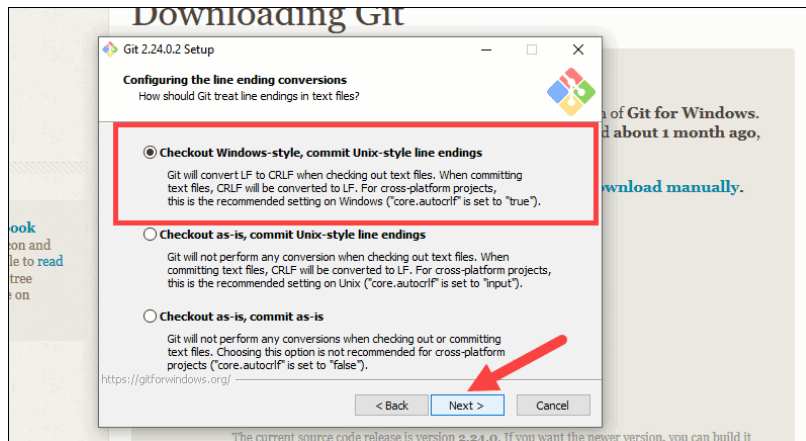
12. The installer now asks which SSH client you want Git to use. Git already comes with its own SSH client, so if you don't need a specific one, leave the default option and click **Next**.



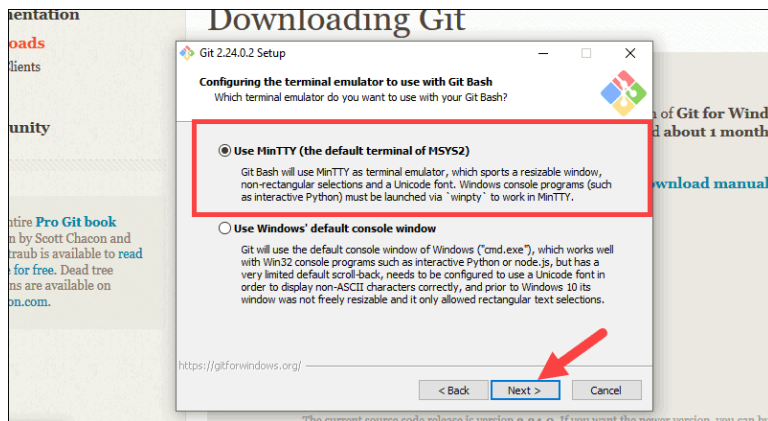
13. The next option relates to server certificates. Most users should use the default. If you're working in an Active Directory environment, you may need to switch to Windows Store certificates. Click **Next**.



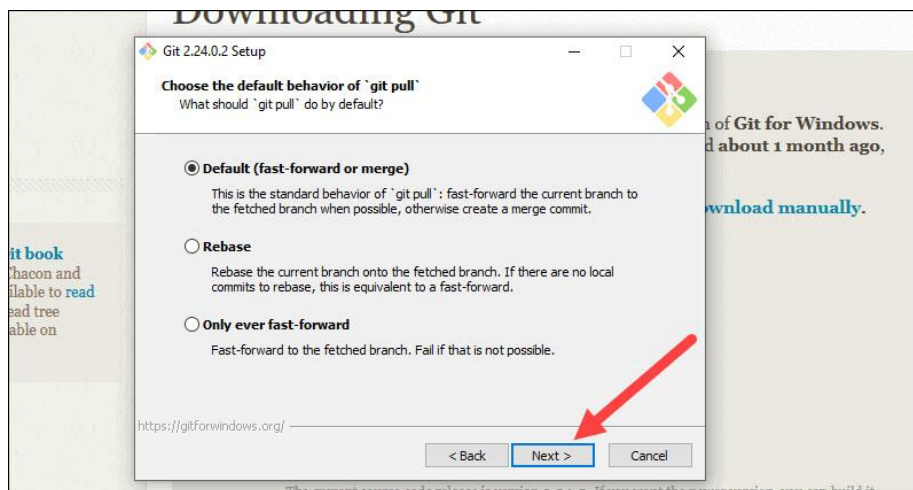
14. The next selection converts line endings. It is recommended that you leave the default selection. This relates to the way data is formatted and changing this option may cause problems. Click **Next**.



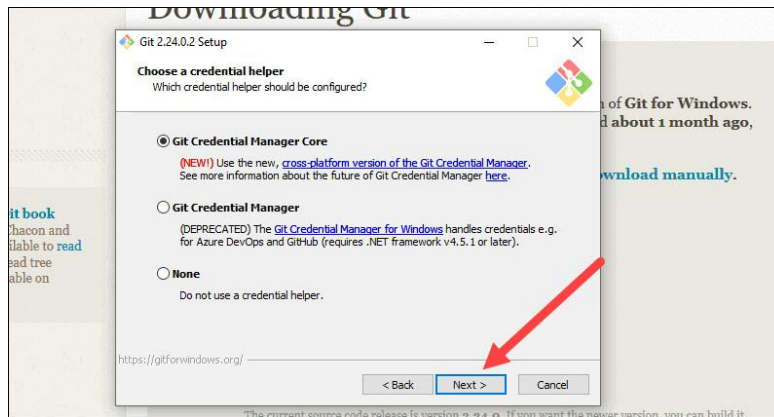
15. Choose the **terminal emulator** you want to use. The default MinTTY is recommended, for its features. Click **Next**.



16. The installer now asks what the **git pull** command should do. The default option is recommended unless you specifically need to change its behavior. Click **Next** to continue with the installation.

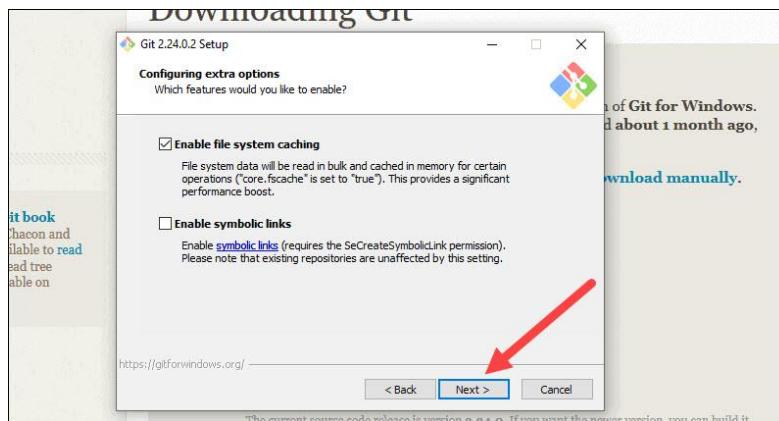


17. Next you should choose which credential helper to use. Git uses credential helpers to fetch or save credentials. Leave the default option as it is the most stable one, and click **Next**.

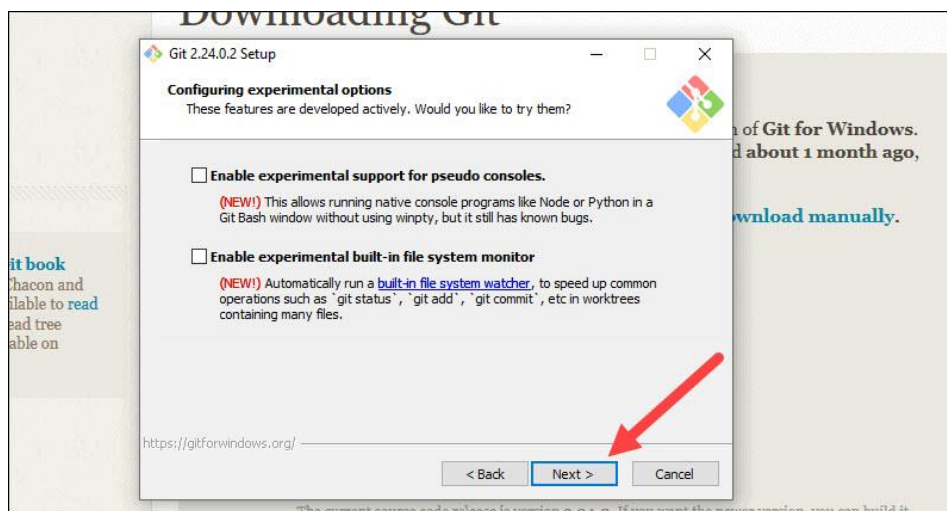


➤ **Additional Customization Options:**

18. The default options are recommended, however this step allows you to decide which extra option you would like to enable. If you use symbolic links, which are like shortcuts for the command line, tick the box. Click **Next**.

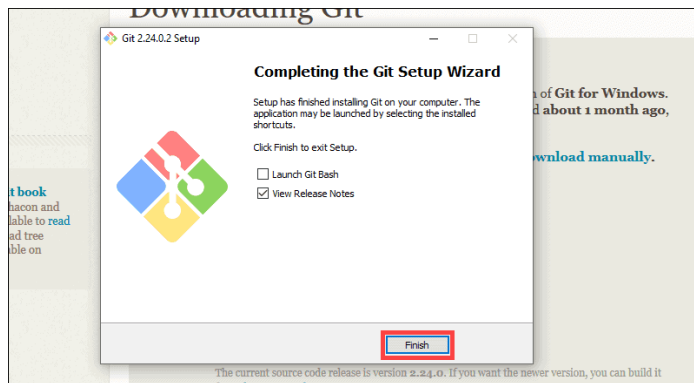


19. Depending on the version of Git you're installing, it may offer to install experimental features. At the time this article was written, the options to include support for pseudo controls and a built-in file system monitor were offered. Unless you are feeling adventurous, leave them unchecked and click **Install**.



➤ **Complete Git Installation Process :**

20. Once the installation is complete, tick the boxes to view the Release Notes or Launch Git Bash, then click **Finish**.

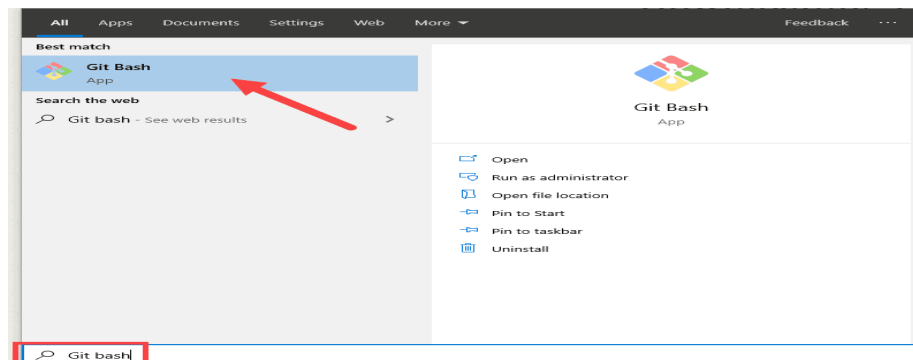


How to Launch Git in Windows:

Git has two modes of use – a **bash scripting shell** (or command line) and a **graphical user interface (GUI)**.

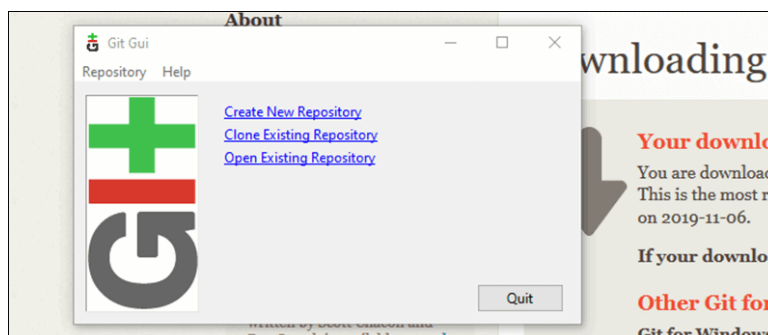
➤ Launch Git Bash Shell

To launch **Git Bash** open the **Windows Start** menu, type **git bash** and press **Enter** (or click the application icon).



Launch Git GUI

To launch **Git GUI** open the **Windows Start** menu, type **git gui** and press **Enter** (or click the application icon).



Connecting to a Remote Repository

You need a GitHub username and password for this next step.

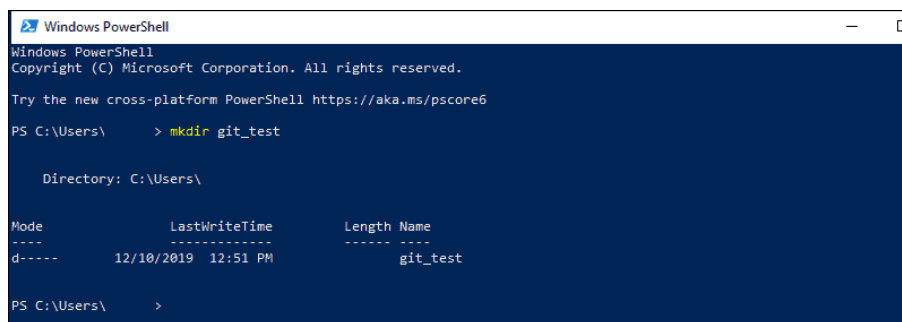
Create a Test Directory

Open a Windows PowerShell interface by pressing **Windows Key + x**, and then **i** once the menu appears.

Create a new test directory (folder) by entering the following:

```
mkdir git_test
```

An example of the PowerShell output.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\ > mkdir git_test

Directory: C:\Users\

Mode                LastWriteTime         Length Name
----                -
d-----         12/10/2019 12:51 PM             git_test

PS C:\Users\ >
```

Change your location to the newly created directory:

```
cd git_test
```

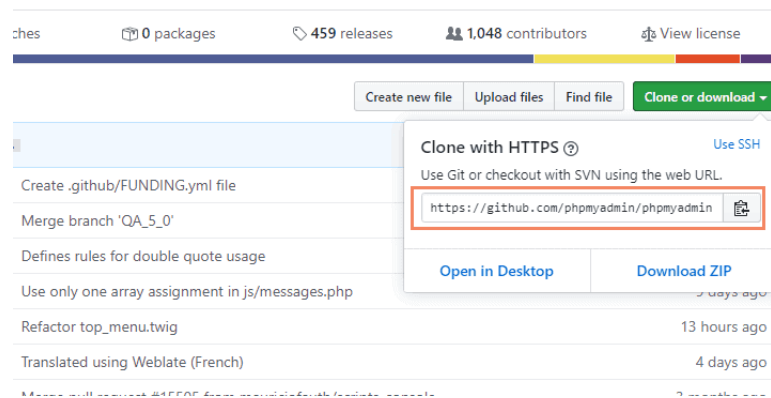
Configure GitHub Credentials

Configure your local Git installation to use your GitHub credentials by entering the following:

```
git config --global user.name "github_username"
git config --global user.email "email_address"
```

Clone a GitHub Repository

Go to your repository on GitHub. In the top right above the list of files, open the **Clone or download** drop-down menu. Copy the **URL for cloning over HTTPS**.



Switch to your PowerShell window, and enter the following:

```
git clone repository_url
```

Pushing Local Files to the Remote Repository

Once you've done some work on the project, you may want to submit those changes to the remote project on GitHub.

1. For example, create a new text file by entering the following into your PowerShell window:

```
new-item text.txt
```

2. Confirmation that the new file is created.

```
PS C:\Users\CCBill\git_test> new-item text.txt

Directory: C:\Users\CCBill\git_test

Mode                LastWriteTime         Length Name
----                -
-a----           12/10/2019 12:58 PM              0 text.txt

PS C:\Users\CCBill\git_test>
```

3. Now check the status of your [new Git branch](#) and untracked files:

```
git status
```

4. Add your new file to the local project:

```
git add text.txt
```

5. Run **git status** again to make sure the text.txt file has been added. Next, commit the changes to the local project:

```
git commit -m "Sample 1"
```

6. Finally, push the changes to the remote GitHub repository:

```
git push
```

You may need to enter your username and password for GitHub.

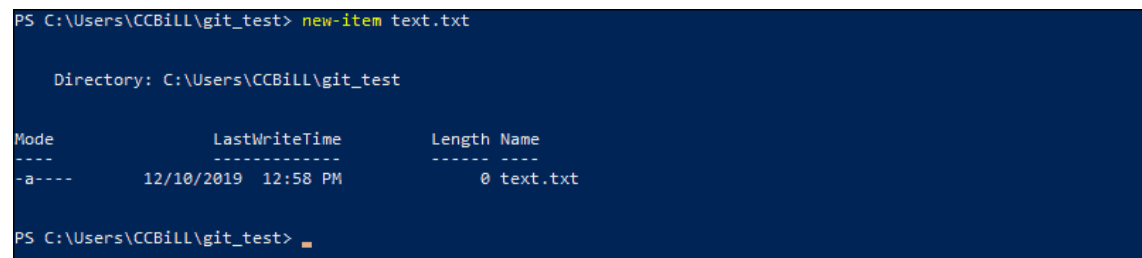
Pushing Local Files to the Remote Repository

Once you've done some work on the project, you may want to submit those changes to the remote project on GitHub.

1. For example, create a new text file by entering the following into your PowerShell window:

```
new-item text.txt
```

2. Confirmation that the new file is created.



```
PS C:\Users\CCBill\git_test> new-item text.txt

Directory: C:\Users\CCBill\git_test

Mode                LastWriteTime         Length Name
----                -
-a-----         12/10/2019 12:58 PM              0 text.txt

PS C:\Users\CCBill\git_test>
```

3. Now check the status of your [new Git branch](#) and untracked files:

```
git status
```

4. Add your new file to the local project:

```
git add text.txt
```

5. Run **git status** again to make sure the text.txt file has been added. Next, commit the changes to the local project:

```
git commit -m "Sample 1"
```

6. Finally, push the changes to the remote GitHub repository:

```
git push
```

You may need to enter your username and password for GitHub.

Working with repositories:

Working with repositories in Git involves various commands and actions to manage your version-controlled projects. Here's a basic guide to get you started:

1. Initializing a Repository:

To start tracking a project, navigate to your project folder and run:

```
csharp  
git init
```

2. Cloning a Repository:

To copy an existing repository from a URL:

```
bash  
git clone <repository_url>
```

3. Checking the Status:

To see the status of changes:

```
lua  
git status
```

4. Adding Changes:

To stage changes for commit:

```
scss  
git add <file(s)>
```

5. Committing Changes:

To commit staged changes:

```
sql  
git commit -m "Your commit message here"
```

6. Viewing Commit History:

To see the commit history:

```
bash  
git log
```

7. Branching:

To create a new branch:

```
php  
git branch <branch_name>
```

To switch to a branch:

```
php  
git checkout <branch_name>  
To create and switch to a new branch:
```

```
css  
git checkout -b <new_branch_name>
```

8. Merging Branches:

To merge changes from one branch to another:

```
php  
git merge <branch_name>
```

9. Remote Repositories:

To add a remote repository:

```
php  
  
git remote add <remote_name> <repository_url>  
To fetch changes from a remote repository:
```

```
php  
  
git fetch <remote_name>  
To pull changes from a remote repository into your current branch:
```

```
php  
  
git pull <remote_name> <branch_name>  
To push changes to a remote repository:
```

```
php  
  
git push <remote_name> <branch_name>
```

10. Handling Conflicts:

If there are conflicts during a merge, resolve them manually and then:

```
scss  
  
git add <conflicted_file(s)>  
git commit
```

11. Ignoring Files:

Create a .gitignore file to specify files or patterns to ignore.

12. Undoing Changes:

To discard changes in your working directory:

```
scss
```

```
git checkout -- <file(s)>
```

To unstage changes:

```
scss
```

```
git reset HEAD <file(s)>
```

To undo the last commit:

```
perl
```

```
git reset HEAD^
```

13. Tagging:

To tag specific commits for easy reference:

```
php
```

```
git tag -a <tag_name> -m "Tag message" <commit_hash>
```

✧ These are some fundamental Git commands to help you manage your repositories. Git is a powerful tool with many features, so feel free to explore more as you become more comfortable with its basics.

GIT Remote Repositories:

In Git, remote repositories are versions of your project that are hosted on the internet or another network. They allow collaboration with others and provide a central place to share changes to the codebase. Here are some common commands for working with remote repositories:

1. Cloning a Remote Repository:

To create a local copy of a remote repository:

```
bash
```

```
git clone <repository_url>
```

3. Adding a Remote Repository:

To add a remote repository:

```
bash
```

```
git remote add <remote_name> <repository_url>
```


4. Listing Remote Repositories:

To see a list of remote repositories:

```
bash  
git remote -v
```

5. Fetching Changes from a Remote Repository:

To get the latest changes from a remote repository:

```
bash  
git fetch <remote_name>
```

6. Pulling Changes from a Remote Repository:

To fetch changes and merge them into your local branch:

```
bash  
git pull <remote_name> <branch_name>
```

6. Pushing Changes to a Remote Repository:

To push your changes to a remote repository:

```
bash  
git push <remote_name> <branch_name>
```

7. Renaming a Remote:

To rename a remote repository:

```
bash  
git remote rename <old_remote_name> <new_remote_name>
```

8. Removing a Remote:

To remove a remote repository:

```
bash  
git remote remove <remote_name>
```

9. Inspecting Remote Information:

To see information about a remote repository:

```
bash  
git remote show <remote_name>
```

10. Changing the URL of a Remote:

To update the URL of a remote repository:

```
bash
```

```
git remote set-url <remote_name> <new_repository_url>
```

11. Pushing Tags to a Remote:

To push tags to a remote repository:

```
bash  
git push <remote_name> --tags
```

12. Pulling Tags from a Remote:

To fetch tags from a remote repository:

```
bash  
git fetch <remote_name> --tags
```

- ✧ These commands allow you to interact with remote repositories, facilitating collaboration and version control in a distributed development environment. Understanding how to work with remote repositories is crucial for effective collaboration and code sharing.

Branching in GIT :

Branching is a fundamental concept in Git that allows you to work on different features, bug fixes, or experiments concurrently without affecting the main codebase. Here's a guide on branching in Git:

1. Create a New Branch:

To create a new branch:

```
bash  
git branch <branch_name>
```

2. Switch to a Branch:

To switch to an existing branch:

```
bash  
git checkout <branch_name>
```

To create and switch to a new branch in one command:

```
bash  
git checkout -b <new_branch_name>
```

In recent Git versions (2.23 and later), you can use:

```
bash  
git switch -c <new_branch_name>
```

3. List Branches:

To see a list of all branches in your repository:

```
bash  
git branch
```

4. Merge Branches:

To merge changes from one branch into another:

```
bash
git checkout <destination_branch>
git merge <source_branch>
```

5. Delete a Branch:

To delete a local branch:

```
bash
git branch -d <branch_name>
```

To force delete a branch (use with caution):

```
bash
git branch -D <branch_name>
```

6. Pushing a New Branch to Remote:

To push a new local branch to a remote repository:

```
bash
git push -u <remote_name> <branch_name>
```

7. Pulling Changes from a Remote Branch:

To fetch changes from a remote branch:

```
bash
git fetch <remote_name>
git checkout <remote_branch_name>
```

To fetch and merge changes in one command:

```
bash
git pull <remote_name> <remote_branch_name>
```

8. Tracking Remote Branches:

To create a local branch that tracks a remote branch:

```
bash
git checkout -b <local_branch_name> <remote_name>/<remote_branch_name>
```

9. Renaming a Branch:

To rename a local branch:

```
bash
git branch -m <old_branch_name> <new_branch_name>
```

10. Viewing Merged and Unmerged Branches:

To see branches that are fully merged into the current branch:

```
bash
git branch --merged
```

To see branches that contain work you haven't merged:

```
bash
git branch --no-merged
```

11. Stashing Changes Before Switching Branches:

To stash changes before switching branches:

```
bash
git stash
```

To apply stashed changes:

```
bash
git stash apply
```

- ✧ These commands cover the basic operations related to branching in Git. Branches provide a powerful way to organize and isolate different aspects of your development work.

Merging in Git:

Merging in Git involves combining changes from different branches. The primary goal is to bring the changes made in one branch into another. Here are the basic steps for merging branches in Git:

1. Switch to the Destination Branch:

Ensure you are on the branch where you want to merge changes into.

```
bash
git checkout <destination_branch>
```

2. Merge Changes:

Merge changes from another branch into the current branch.

```
bash
git merge <source_branch>
```

If you're using the newer Git versions (2.23 and later), you can also use the following syntax:

```
bash
git switch <destination_branch>
git merge <source_branch>
```

3. Resolve Merge Conflicts:

If Git encounters conflicts during the merge, it will pause the process. You need to resolve the conflicts manually.

Git marks the conflicted areas in the affected files. Open these files, resolve conflicts, and save the changes.

After resolving conflicts, add the modified files to mark them as resolved.

```
bash
git add <conflicted_file(s)>
Complete the merge.
```

```
bash
git merge --continue
```

Alternatively, you can abort the merge if needed.

```
bash
git merge --abort
```

4. Fast-forward Merge:

If there are no changes in the destination branch since you created the source branch, Git performs a fast-forward merge.

```
bash
git merge <source_branch>
```

5. Delete Source Branch (Optional):

If you've merged changes and no longer need the source branch, you can delete it.

```
bash
git branch -d <source_branch>
```

Use -D if the branch contains unmerged changes (use with caution).

```
bash
git branch -D <source_branch>
```

6. View Merge History:

To view the branches that have been merged into the current branch:

```
bash
git branch --merged
```

To view branches that have not been merged:

```
bash
git branch --no-merged
```

7. Undo a Merge:

If you want to undo a merge that hasn't been pushed yet, you can use:

```
bash
git reset --hard HEAD^
```

Be cautious when using this command, as it discards the entire merge.

- ✧ Merging is a crucial aspect of collaborative development in Git. Understanding how to merge branches efficiently and handle conflicts is essential for managing code changes effectively.

Workflows in GIT:

In Git, workflows define how different developers collaborate on a project, manage branches, and integrate changes. There are various Git workflows, each with its own advantages and use cases. Here are some common Git workflows:

Centralized Workflow:

- **Description:** This is a simple workflow where a central repository acts as the single point of collaboration.
- **Usage:** Small teams or projects where a centralized control is sufficient.
- ❖ **Workflow:** Clone the central repository. Make changes and commit locally.
- ❖ Push changes to the central repository.

Feature Branch Workflow:

- **Description:** Each new feature or bug fix is developed in its own branch, which is later merged into the main branch.
- **Usage:** Medium to large teams where parallel development is necessary.
- ❖ **Workflow:** Create a new branch for a feature or bug fix. Commit changes to the feature branch.
- ❖ Merge the feature branch into the main branch when the feature is complete.

Gitflow Workflow:

- **Description:** Extends the feature branch workflow with support for releases and hotfixes.
- **Usage:** Projects with a regular release cycle and a need for versioning.

Workflow:

- ❖ Maintain develop and master branches.
- ❖ Create feature branches from develop.
- ❖ Merge features into develop when complete.

Forking Workflow:

- **Description:** Contributors fork the main repository, make changes in their fork, and create pull requests to propose changes.
- **Usage:** Open source projects with many contributors.

Workflow:

- ❖ Fork the main repository.

- ❖ Clone the forked repository.
- ❖ Create a feature branch for changes.
- ❖ Commit changes and push to the fork.
- ❖ Create a pull request to propose changes to the main repository.

Pull Request Workflow:

- **Description:** Similar to the forking workflow, but contributors push branches directly to the main repository.
- **Usage:** Projects with a smaller team and where contributors have write access to the main repository.

Workflow:

- ❖ Clone the main repository.
 - ❖ Create a feature branch.
 - ❖ Commit changes and push the branch to the main repository.
 - ❖ Create a pull request to propose changes.
- ✧ Choose a workflow based on your project's needs, team size, and development/release cycles. It's also common for teams to customize or combine elements from different workflows to suit their specific requirements.

Working with GitHub:

Working with GitHub involves using its platform for version control, collaboration, and code hosting. GitHub is widely used in the software development community, but it can also be utilized for managing any type of project or document. Here's a basic guide to get you started:

1. Creating a GitHub Account:

Go to GitHub and sign up for an account if you don't have one.

2. Setting Up Git:

Install Git on your local machine if it's not already installed.

Set your username and email using the following commands in your terminal or command prompt:

arduino

git config --global user.name "Your Name"

git config --global user.email "your.email@example.com".

3. Creating a Repository:

On GitHub, click the "+" icon in the top right corner and select "New repository."

Give your repository a name, description, and initialize it with a README if you want.

Click "Create repository."

4. Cloning a Repository:

To work on a project locally, you need to clone the repository to your machine.

bash

git clone <https://github.com/username/repository.git>

5. Branching:

Create a new branch for your work to avoid making changes directly to the main branch.

```
git branch feature-branch  
git checkout feature-branch
```

6. Making Changes:

Edit files in your local repository.

Use git add to stage your changes and git commit to commit them.

```
sql  
git add .  
git commit -m "Your commit message".
```

7. Pushing Changes:

Push your changes to the remote repository.

```
perl  
git push origin feature-branch
```

8. Pull Requests:

On GitHub, create a pull request (PR) to propose changes from your branch to the main branch.

Discuss and review changes with collaborators.

Once approved, merge the PR.

9. Pulling Changes:

Keep your local repository updated with changes from the remote repository.

```
css  
git pull origin main
```

10. Issues and Projects:

Use GitHub Issues to track tasks, bugs, and enhancements.

Utilize GitHub Projects to organize and prioritize tasks.

11. Collaborating:

Add collaborators to your repository to work on the same project.

Use the Issues tab for communication and planning.

12. GitHub Pages:

Host a website directly from your GitHub repository using GitHub Pages.

13. GitHub Desktop:

Consider using GitHub Desktop, a graphical user interface for Git and GitHub.

14. Git Ignore:

Create a .gitignore file to specify files and directories that should be ignored by Git.

This is a basic overview, and GitHub offers many more features. Explore documentation and guides as needed, and don't forget to check for updates or changes to GitHub features.

