Git Feature Branch Workflow

[cite_start]The feature branch workflow is a standard practice that keeps your `master` branch clean and stable. [cite: 1727] [cite_start]It lets you work on new features in an isolated environment without affecting the main codebase. [cite: 1728] [cite_start]Once a feature is complete and tested, it's merged back into `master`. [cite: 1729] [cite_start]The Git commands are the same whether you're using Windows Shell (Command Prompt, PowerShell) or a Bash shell. [cite: 1730]

---

1. Start from `master`

[cite_start]Before you do anything, you need to make sure your local `master` branch is up-to-date with the remote repository (like GitHub or Azure DevOps). [cite: 1731] **CODE_BLOCK**

Switch to your master branch

git switch master

Pull the latest changes from the remote server

git pull **CODE_BLOCK**

---

2. Create and Switch to a Feature Branch

[cite_start]Now, you'll create a new branch for your feature. [cite: 1732] [cite_start]Branch names should be short and descriptive, like `login-form` or `user-profile-page`. [cite: 1733] [cite_start]This command creates a **new branch** and **immediately switches** to it. [cite: 1734] **CODE_BLOCK**

The -c flag stands for "create"

git switch -c new-feature-name **CODE_BLOCK**

[cite_start]*You can now work safely on this branch. [cite: 1735] Think of it as a separate copy of the project where your changes won't affect anyone else until you're ready.*

---

3. Develop the Feature: Add and Commit

[cite_start]This is where you'll do your work—writing code, adding files, and fixing bugs. [cite: 1736] As you complete small, logical chunks of work, you should **commit** them. [cite_start]A commit is like a permanent save point. [cite: 1737] The process for each commit is the same:

1. [cite_start]**Stage** your changes (`git add`). [cite: 1738]
2. [cite_start]**Commit** them with a clear message (`git commit`). [cite: 1739] **CODE_BLOCK**

Stage a specific file

git add path/to/your/file.js

Or stage all changed files in the project

git add .

Commit the staged files with a descriptive message

git commit -m "feat: Add user login form component" **CODE_BLOCK**

[cite_start]You can (and should) have many commits on your feature branch. [cite: 1742] [cite_start]Committing often creates a clear history of your work and makes it easier to undo changes if something goes wrong. [cite: 1742]

---

4. Keep Your Branch Synced (Optional but Recommended)

[cite_start]If you're working on a feature for a while, the `master` branch might get updated by your teammates. [cite: 1743] It's a good practice to pull those updates into your feature branch. [cite_start]This makes the final merge much easier. [cite: 1744] **CODE_BLOCK**

Fetch the latest changes from all remote branches

git fetch origin

Re-apply your commits on top of the latest master branch

git rebase origin/master **CODE_BLOCK**

[cite_start]The **`rebase`** command essentially "unplugs" your branch's changes, updates the base to the latest version of `master`, and then "re-plugs" your changes on top. [cite: 1745] [cite_start]This keeps your project history clean and linear. [cite: 1746]

---

5. Merge Your Feature into **`master`**

[cite_start]Once your feature is complete, tested, and ready to go, it's time to merge it back into the `master` branch. [cite: 1746] **CODE_BLOCK**

1. First, go back to the master branch

git switch master

2. Make sure it's up-to-date one last time

git pull

3. Merge your feature branch into master

git merge --no-ff new-feature-name **CODE_BLOCK**

[cite_start]Using **`--no-ff`** (no fast-forward) is a crucial best practice. [cite: 1747] [cite_start]It creates a "merge commit" that ties the history of your feature branch together. [cite: 1748] [cite_start]This makes it very easy to see when a specific feature was merged into `master` and which commits belonged to it. [cite: 1749]

---

6. Push and Clean Up

[cite_start]Your `master` branch now has the new feature, but only on your local machine. [cite: 1750] You need to push it to the remote server. [cite_start]After that, you can delete the feature branch, since its work is now part of `master`. [cite: 1751] **CODE_BLOCK**

1. Push the updated master branch to the remote

git push origin master

2. Delete the local feature branch

git branch -d new-feature-name

3. Delete the remote feature branch

git push origin --delete new-feature-name **CODE_BLOCK**

[cite_start]That's the complete lifecycle! [cite: 1752] [cite_start]□□ You've successfully created a feature, developed it in isolation, and safely merged it into the main project. [cite: 1753]