A Beginner's Guide to Git Branching

Note: This document assumes you have a basic understanding of Git and are familiar with using the terminal/command line. New to Git? Refer to the official tutorial.

Branching is an essential feature of Git that allows users to create isolated versions of the code. Developers can work on new features, code fixes, or even experiment independently without affecting the main codebase. Once the changes are tested and validated, the branch can be merged back into the main branch.

Why is branching useful?

Git branches provide multiple benefits:

- Experiment safely without risking the main codebase

- Isolate tasks by feature, bug, or update for better clarity

- Simplify rollback by keeping changes confined to a single main branch

- Allow multiple developers to work in parallel

- Help keep main branch stable and production ready

- Support testing and feedback on specific changes before merging

- Separate workspaces reduce code rewrites and rework

Without branches, developers would work on the original version of code, making collaboration chaotic and full of errors.

How to use Git branches?

1. Go to the project folder

Before creating a branch, make sure you are inside the main project folder in the terminal.

```
cd your-main-folder
```

This moves you into the folder where your Git project is located.

If the folder hasn't been initialized yet.

```
git init
```

This sets up Git in the current folder. Git initialization is only required once per project.

2. Create a new branch

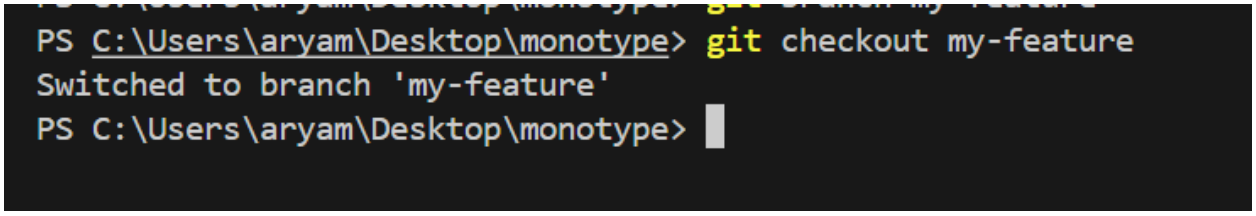
```
git branch my-feature
```

The new branch `my-feature` is created which is a copy of the main code.

3. Switch to the new branch

```
git checkout my-feature
```

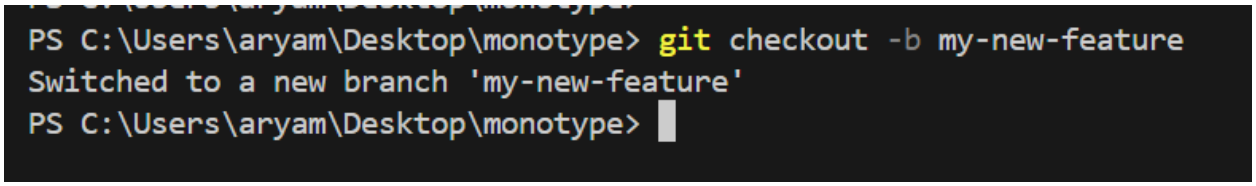
Switches to the `my-feature` branch.



4. Create and switch in one step

```
git checkout -b my-new-feature
```

Creates and switches to the branch `my-new-feature` immediately.



5. After making changes, commit and save

```
git add .
git commit -m "Added login feature"
```

Stages and saves the changes made in the new branch. Commit in simple terms means snapshot of the project's current stage.

```
PS C:\Users\aryam\Desktop\monotype> git add .
>> git commit -m "Added login feature"
[my-feature 7a2cbf9] Added login feature
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\aryam\Desktop\monotype>
```

6. Merge the branch into master

```
git checkout master
git merge my-feature
```

Combines the changes back into the main branch. In a real time collaborative environment, only after proper review is the branch merged. This is done via Pull Request.

```
PS C:\Users\aryam\Desktop\monotype> git checkout master
>> git merge my-feature
Switched to branch 'master'
Updating ac2c95d..7a2cbf9
Fast-forward
 main.text | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
PS C:\Users\aryam\Desktop\monotype>
```

7. Delete the branch

```
git branch -d my-feature
```

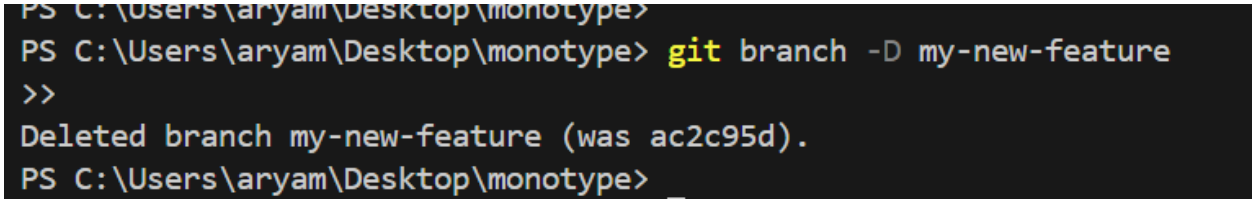
Deletes the branch safely after it is merged.

```
PS C:\Users\aryam\Desktop\monotype> git branch -d my-feature
>>
Deleted branch my-feature (was 7a2cbf9).
PS C:\Users\aryam\Desktop\monotype>
```

8. Force delete an unmerged branch

```
git branch -D my-new-feature
```

Force deletes the branch even if not merged, so use with caution.



Best Practices

- Create separate branches for each feature or bug fix

- Use clear, descriptive names, for example, feature/user-logout

- Delete branches after the merge is complete to keep the repository clean

- Pull updates from the main/master branch frequently to avoid conflicts

- Use pull requests to review code before merging

Common Issues

1. Creating a branch before first commit

If you run `git branch my-feature` immediately after `git init`, Git will throw an error. The reason is that there is no main branch, that is, no snapshot of the main code. So you must first commit in the main folder which by default creates the `master` branch.

```
PS C:\Users\aryam\Desktop\monotype> git branch my-feature
fatal: not a valid object name: 'master'
PS C:\Users\aryam\Desktop\monotype> git add .
PS C:\Users\aryam\Desktop\monotype> git commit -m "first commit"
[master (root-commit) ac2c95d] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 main.text
PS C:\Users\aryam\Desktop\monotype> git branch my-feature
PS C:\Users\aryam\Desktop\monotype>
```

2. Forgetting to switch to the branch

Many beginners make the mistake of not switching to the new branch after creating it, so always use the checkout command, or better, create and switch in one step.

3. Merge conflicts

This happens when two branches change the same part of a file differently. Git won't know which one to keep and will flag a conflict. Solution for this is that you will have to manually resolve the conflicts in the code editor and then commit.

```
git add conflicted-file
git commit -m "Resolved conflict"
```

4. Making changes on the wrong branch

It's common to forget to switch branches before starting work. So you might think that you are on `my-feature`, but you are still on `master`. So run:

```
git branch
```

The command `git branch` will show the list of branches. The current branch will have an asterisk (*) next to it. It is recommended to always check the current branch you are working on.

```
PS C:\Users\aryam\Desktop\monotype> git checkout -b my-feature
Switched to a new branch 'my-feature'
PS C:\Users\aryam\Desktop\monotype> git branch
  master
* my-feature
PS C:\Users\aryam\Desktop\monotype>
```

To learn more, visit git-scm.com/docs/git-branch