Using Branches in Git

Separation of Concerns: good in your code, good in your coding

Wolf 9 March 2021

Why am I talking about branches?

- Because they help you get your work done faster and more easily
- Because they help you collaborate with others

What do I want you to come away with?

- The desire to use branches, and enough knowledge to benefit from them

What are the most important things I can teach you?

- Branching is useful, quick, and easy (maybe merging not quite as easy)
- Branches let you handle multiple problems at once
- Shared branches facilitate collaboration
- Commit often: commit before merging, rebasing, or switching (unless you are explicitly trying to take your changes with you to the new branch)
- Merge puts your commits *onto* the target branch
- Rebase moves your branch from one place to another, but it's still just a branch; you need to merge to get your commits onto the target branch
- Branching, merging, and rebasing all work together

How can branches make you happy?

- You have some commits you want to push, and others that are not ready
- You had to stash all your changes to work on an emergency fix
- You have changes you can't put onto the main branch, but you want to save them … maybe for a long time
- You have lots of small commits (that's good!), but you just want one big fat commit to land on main
- You want to try an experiment
- You want to see if *your* changes introduced the bug

What are the major pitfalls?

- Merge conflicts
- Your loose (uncommitted) changes try to come with you when you switch branches. Sometimes this is a benefit. Sometimes it's a hazard.
- Rebasing commits upon which other people have based their work

Even if you don’t grasp the details yet…

I want you to know what’s possible, so you know where to look for more answers

The topics we're going to cover

- Branching
- Rebasing
- Merging
- Collaboration & Workflows

The starting line

- Terminology (more later)
  - The ‘tip’ of the branch, and the branch name itself, refer to the newest commit on the branch. Inside Git, this is simply a pointer to that commit
  - `HEAD` is the commit you have checked-out right now. It usually points to the top of a branch, i.e., it is a pointer to a pointer
  - Usually refer to the same commit, but when you checkout a specific commit or tag, you are no longer at the tip of a branch. `HEAD` is “detached” from any branch

The starting line

- I'm using Git version 2.30.1. You should be using at least 2.23.0 so that you have git switch
- You should already know, at least roughly, how to use `git log`
- `git log –online –abbrev-commit –decorate`. The `–decorate` is the key for showing `HEAD` and the branch tips
- `git config init.defaultbranch main`

A lot of people and organizations are abandoning `master` as a branch name. The general consensus among those switching is to use `main` instead.

Branching

What is a branch?

- Inside Git, a branch is just a pointer to a commit that moves forward when you add new commits
- With respect to your work, a branch is a separate line of development that doesn't interfere with other lines of development
- Every separate line of development is a branch even if you only have one: your primary branch is, in fact, a branch
- Once a branch has commits on it — it *looks* like the branch of a tree coming off the trunk (or off of some other branch). This happens whenever a commit has more than one child

How branches work inside Git

Live demo

Create a new repo

Add a commit

Look at `git log`

Look at `.git/refs/heads/main`

Look at `.git/HEAD`

Make another commit

Examine all the same things

Creating a new repo automatically creates your first branch. In my case, it's named `main`, for you it might be `master` or anything else you desire.

Make a new branch

- `git switch -c <new-branchname> [<start-point>]`
- `git checkout -b <new-branchname> [<start-point>]`
- `git branch <new-branchname> [<start-point>]`

The `switch` and `checkout` versions both create the new branch and switch to it in one step. The `branch` method only creates the branch.

Delete a (local) branch

- `git branch ( -d | -D ) <branchname>`

You can’t delete the branch you currently have checked-out

Work on a different branch

- `git switch <other-branchname>`

There are lots of options, but this is the core command. You've already seen how git switch can create a new branch before switching to it.

Multiple branches, inside Git

Live demo, using the repo from the last live demo

Switch to a new branch named `my-branch`

Look at `git log –all`

Look at `.git/refs/heads/my-branch`

Look at `.git/HEAD`

Make another commit

Examine all the same things

Switch back to `main` (note that all the changes from `my-branch` disappeared)

Make a new commit on `main`

Examine all the same things

Look at a graphical view of the log

So the two things we've learned here are reinforcing how Git manages branches internally; and seeing how to do work on two different branches.

Rebasing

What is rebasing?

- “Moving” a set of commits (like a branch) from one “base” to another by replaying the changes that made those commits starting from the new base
- This creates a new set of commits that look very much like the originals, but have different hashes
- The branch head is moved to the tip of the new set of commits so the old set is “forgotten” and the branch looks and acts exactly like it moved

Rebasing, inside Git

Live demo, using ready-to-rebase repo

Look at `git log –oneline –all –graph –decorate`; examine in Gitk

Tag `my-branch`

Rebase `my-branch` onto the tip of `main`

Examine all the same things again

Note how the old commits are still reachable from the tag, but the `my-branch` has “moved” to the tip of `main`

Rebasing, inside Git

Live demo, using ready-to-rebase repo

Create and switch to a `another-branch` from somewhere back in time

Examine `HEAD` and `refs/heads/another-branch`

Rebase `another-branch` onto the tip of `main`

Examine all the same things again

Note how in this case, since `another-branch` was already reachable from `main`, all this had to do was more the branch head. This was a fast forward

Rebasing, inside Git

Now what happens if we rebase main itself?

Live demo, using ready-to-rebase repo

Get a graphical view of where main is right now

Switch to main, and rebase it onto the tip of my-branch

Refresh the graphical view

Note how just like the last case, all this had to do was more the branch head. This was another fast forward. This is one way to merge

Merging

What is merging?

- When we rebased main previously, we made all the commits from my-branch reachable from main. That means main *contains* all those changes
- Merging is another way get a given branch to *contain* all the changes from another
- Merging makes a new commit with **two** (or more) parents. That new commit is the textual union of all the changes from both parent branches

Merging, inside Git

Live demo, using ready-to-merge repo

Get a graphical view of where main is right now

Show files on each branch by switching between

Checkout main and merge my-branch

Refresh the graphical view; show the files

Collaboration & workflows

Collaboration & workflows

- One person, entirely local, developing entirely on main
- One person, entirely local, developing several different features at once (and fixing bugs)
- A small team, pushing-to and pulling-from a central repo, developing different features at once (some with just one engineer, some shared among more than one), and fixing bugs
- Using main as your stable branch, and maintaining a develop branch where ongoing work is merged. Feature and bug-fix branches come and go
- Big teams, external contributors, merge requests
- More! See `git help workflows` for a description of how git itself is managed

Resources

- `git help topic`
  - `branch, merge, rebase, workflows`
- https://learngitbranching.js.org/ (strongly recommended)
- Pro Git: https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell
- Pro Git: https://git-scm.com/book/en/v2/Git-Tools-Advanced-Merging
- Fugitive: https://github.com/tpope/vim-fugitive
- Tig: https://github.com/jonas/tig

learngitbranching.js.org too bad the trees are all upside-down