GOMYCODE

Current Skill Branching

Branching

So far in our Git workflow we've only been working on a single branch. But when you're working with a team, this isn't usually desirable. What if you want to go off on your own and work on some experimental new feature? It would be nice if you could do so without worrying about breaking the code for everyone else, or conflicting with things that other people are working on.

In most modern work flows, we do not do all of our work on a single branch. Instead, we usually have many different branches for certain use cases (bug fixes, new features, deployment), so it's essential to understand how to create, delete, and merge branches.

Before creating a branch, let's first type `git branch` in the terminal. You should see a list of all your branches; right now, there should just be a single branch called `master`. This is the default branch for all Git repositories.

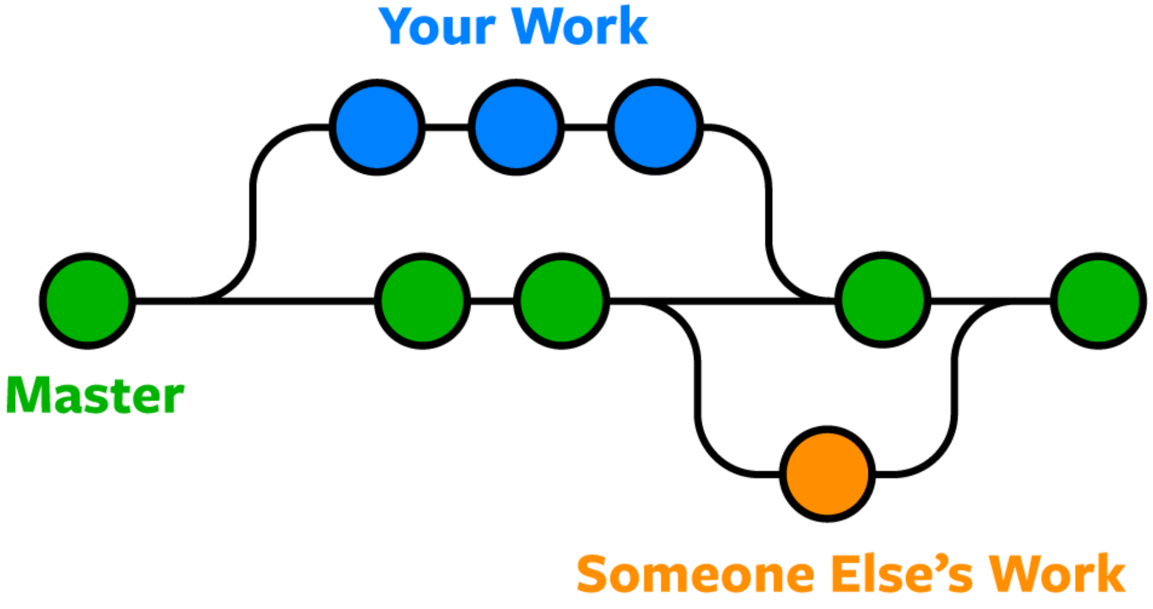
To create a new branch we use the `git checkout` command with the `-b` flag and then pass in a name of a branch. This looks like `git checkout -b NAME_OF_BRANCH`.

If we want to move to another branch that has been previously created we use the `git checkout` command and then specify the name of a branch. This looks like `git checkout NAME_OF_BRANCH`

To delete a branch we make sure we are not on that branch and then run `git branch -D NAME_OF_BRANCH`

To see all of the branches we have, we can type `git branch -a`. The `-a` flag will include remote branches (branches on GitHub or other remote locations). The flag does not matter right now, but it's good to get in the habit of using it with the `git branch` command.

Try creating a branch called `second` branch. When you type `git branch -a`, you should now see two branches; your current branch will have an asterisk next to it. You can now add and commit files to the two branches completely independently of one another! Try this out by adding separate files to each branch.



Merging

With a branch workflow, we usually create a new branch for something we are working on (a new feature, a redesign, etc.). When we are done with that modification, we need to put our code back on the `master` branch. Traditionally, the `master` branch is reserved for production code and immediate bug fixes. In order to put our code back on the `master` branch we need to merge our code in. Here's what that looks like:

Let's:

1. Create a folder called `learn_branching` and `cd` into it ⇒ `mkdir learn_branching && cd learn_branching`.
2. Initialize a `git` repository ⇒ `git init`.
3. Create a file called `first.txt` ⇒ `touch first.txt`.
4. Add that file `git add .`.
5. Commit that file `git commit -m "initial commit"`.
6. Create a new branch called `feature` ⇒ `git checkout -b feature`.
7. Now that you are on the `feature` branch, create a file called `new.txt` ⇒ `touch new.txt`.
8. Add that file ⇒ `git add .`.
9. Commit that file ⇒ `git commit -m "adding new.txt"`.
10. Create another file called `another.txt` ⇒ `touch another.txt`.
11. Add that file ⇒ `git add .`.
12. Commit that file ⇒ `git commit -m "adding another.txt"`.
13. Change back to the master branch ⇒ `git checkout master`. Note that this branch has no awareness of `new.txt` or `another.txt`!

14. Merge our changes from the feature branch into the `master` branch ⇒ `git merge feature`
15. Delete our branch called `feature` ⇒ `git branch -D feature`

Now if you take a look at `git log --oneline --decorate` you'll see that the commit history on feature has ben merged into `master`! (`--decorate` gives you nice coloring around branches and where they are in the commit history.)

Your Turn

Practice makes perfect. Walk through the following steps to get more experience with the branching and merging workflow.

1. Create a folder called `branch_time`.
2. `cd` into that folder.
3. Initialize an empty git repository.
4. Create a file called `first.txt`, then add and commit the file.
5. Create a new branch called `amazing_feature`.
6. Create a file called `best.txt`.
7. Add the file.
8. Commit the file with the message -m "added best.txt".
9. Switch back to the `master` branch.
10. Merge your changes from the `feature` branch into `master`.
11. Delete the feature branch.

‹ Previous

next ›