Git 101: Branching

https://developer.cisco.com/learning/lab/git-branching/step/1

Introduction

This Learning Lab provides you with information about using Git branching.

Prerequisites

- Completion of the Git Intro Learning Lab.
- A Git client on your workstation. If you are a DevNet event - such as at CiscoLive! - the workstations are pre-installed with git.

Isolating changes

Let's say you have a new feature that you're working on. You might have to refactor some of your code, and you want to be able to check in your changes as you make progress. You also want to ensure that your changes don't impact the production code. To manage this situation, you need to isolate your changes, and so you should use Git branching.

Next: Git branching

Git branching

Branches are created with the command git branch <branch name>.

That command enables you to make changes in an area that won't affect the main branch. One convention in Git is that the main branch is typically named main or master.

Note: As of 1 October 2020, all newly created GitHub repositories will have main as the default branch name. Repositories created prior to that date or hosted on other servers may still use master. Refer to Renaming the default branch from master for more information.
When you're working in a branch, you're basically operating in a "parallel universe" until you merge your changes back into main.

Branches are often used when implementing new features or hotfixes. They can be submitted for review by team members, and then once verified, can be pulled into the main codebase.

1. In the git-intro directory you created in the git Intro, create a branch:

```
2. $ git branch austen
```

```
3. $ git branch
4. * main
5.   austen
```

6. Now, you need to switch to your branch.

```
7. $ git checkout austen
8. Switched to branch 'austen'
```

Faster Git branching

The `git checkout` command can be used with a `-b` flag to combine the commands in the previous section into a single command:

```
$ git checkout -b austen
Switched to a new branch 'austen'
```

Next: Git branching

Navigating branches in Git

Now that you have a branch, you can start using it to make changes.

1. Add some text to your `first.txt` file with some text from Jane Austen.

   It isn't what we say or think that defines us, but what we do.
2. Save your file, commit your change, and then checkout the `main` branch.

```
3. $ git checkout main
4. Switched to branch 'main'
```

5. Once you've done that, you need to merge the content from the `austen` branch.

```
6. $ git merge austen
7. Merge made by the 'recursive' strategy.
8. first.txt | 2 ++
9. 1 file changed, 2 insertions(+)
```

10. Open `first.txt`, and you'll see that your merges were successful.

Once a branch mas been merged into main, you can delete your working branch with the following command:

```
$ git branch -d austen
Deleted branch austen (was 2bfb35e).
```

Next: Managing merge conflicts

Managing merge conflicts

At some point, you're going to have a merge conflict. This occurs when one or more uses have made overlapping changes to a file, and Git can't automatically merge the changes.

In this exercise, you will deliberately create a merge conflict so that you can learn how to resolve it.

Note: This exercise has you commit a change directly to main, which is not usually considered a Git best practice. This is being done for demonstration purposes only.

1. Create a new branch, lincoln.
2. Make a change to first.txt by adding ca. 1800s to the end of the Emerson quote:

```
3. -Letters and Social Aims. Quotation and Originality. Ralph Waldo Emerson. ca 1800s
```

4. Commit your change (git commit -a -m 'add date').
5. Now switch back to the main branch.
6. Make a change to the same line by adding ca 1820-1830. Commit your change.
7. Try to merge the branches with git merge lincoln, and you should see a message like this:

```
8. $ git merge lincoln
9. Auto-merging first.txt
10. CONFLICT (content): Merge conflict in first.txt
11. Automatic merge failed; fix conflicts and then commit the result.
```

12. Open the first.txt file. You should see some new content in the file:

```
13. <<<<<<< HEAD
14. -Letters and Social Aims. Quotation and Originality. Ralph Waldo Emerson. ca 1800s!
15. =======
```

```
-Letters and Social Aims. Quotation and Originality. Ralph Waldo Emerson. ca 1820-1830.
>>>>>>> lincoln
```

This markup is helping you see where the HEAD version (aka main in this case) is conflicting with the lincoln branch version.

18. You have to fix this manually, generally you'd use a merge tool, but we can do this simply in the text editor. Take out this part:

```
<<<<<<< HEAD
-Letters and Social Aims. Quotation and Originality. Ralph Waldo Emerson. ca 1800s!
=======
```

22. Save your file, and then commit.

```
$ git commit -a -m 'manually merged from branch lincoln'
[main 69a8a00] manually merged from branch lincoln
```

Next: Stashing changes

Stashing changes

Though not directly related to branching, stashing your changes is a useful feature of Git.

For example, while you are working on some changes in a branch, you may need to make a quick fix to a file in the repo before you are finished with your other changes. This exercise will show you how to stash changes.

Note: This exercise has you commit a change directly to main, which is not usually considered a Git best practice. This is being done for demonstration purposes only.

1. Switch to the lincoln branch (git checkout lincoln), and add the famous quote from the Gettysburg Address to first.txt:

```
Four score and seven years ago our fathers brought forth on this continent,
a new nation, conceived in Liberty, and dedicated to the proposition that
all men are created equal.
```

5. Save your work. Now, imagine that a colleague needs you to edit the first.txt file with a different change for immediate publication. To ensure that you don't lose your work, you can stash your changes. First, check the status of your work:

```
$ git status
On branch lincoln
Changes not staged for commit:
 (use "git add <file>..." to update what will be committed)
 (use "git checkout -- <file>..." to discard changes in working directory)

modified:   first.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

To stash your work, use git stash:

```
$ git stash
Saved working directory and index state WIP on lincoln: 237b267 Make date into range
HEAD is now at 237b267 Make date into range
```

Now, check out the main branch.
In first.txt, change 1820-1830 to 1820-1840.
Make the change, and commit your work.
Check out the lincoln branch, and execute git stash pop. This command takes the changes you have stashed, adds them back to your file, and deletes the stash.

```
$ git stash pop
On branch lincoln
Changes not staged for commit:
 (use "git add <file>..." to update what will be committed)
 (use "git checkout -- <file>..." to discard changes in working directory)

modified:   first.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (484bfc5cf0b5d43328e07c398d8a4dd4901f777b)
```

You'll see that the Lincoln quote is back. So, you can now commit your changes.

Using what you learned earlier in this lab, try merge those changes into the main branch.

Summary

Good job! You have learned some of the fundamentals of Git branching, including:

- How to branch
- How to merge
- How to stash work in progress