Branching

Overview

Teaching: 10 min
Exercises: 10 min
Questions

- What is a branch?
- How can I merge changes from another branch?

Objectives

- Know what branches are and why you would use them
- Understand how to merge branches
- Understand how to resolve conflicts during a merge

What is a branch?

You might have noticed the term *branch* in status messages:

```
$ git status
```

```
On branch master
nothing to commit (working directory clean)
```

and when we wanted to get back to our most recent version of the repository, we used `git checkout master` .

Not only can our repository store the changes made to files and directories, it can store multiple sets of these, which we can use and edit and update in parallel. Each of these sets, or parallel instances, is termed a `branch` and `master` is Git's default branch.

A new branch can be created from any commit. Branches can also be *merged* together.

Why are branches useful?

Suppose we've developed some software and now we want to try out some new ideas but we're not sure yet whether we'll keep them. We can then create a branch 'feature1' and keep our `master` branch clean. When we're done developing the feature and we are sure that we want to include it in our program, we can merge the feature branch with the `master` branch. This keeps all the work-in-progress separate from the `master` branch, which contains tested, working code.

When we merge our feature branch with master git creates a new commit which contains merged files from master and feature1. After the merge we can continue developing. **The merged branch is not deleted.** We can continue developing (and making commits) in feature1 as well.

Branching in practice

One of our colleagues wants to contribute to the paper but is not quite sure if it will actually make a publication. So it will be safer to create a branch and carry on working on this “experimental” version of the paper in a branch rather than in the master.

```
$ git checkout -b paperWJohn
```

```
Switched to a new branch 'paperWJohn'
```

We’re going to change the title of the paper and update the author list (adding John Smith). However, before we get started it’s a good practice to check that we’re working on the right branch.

```
$ git branch                    # Double check which branch we are working on
```

```
  master
* paperWJohn
```

The * indicates which branch we’re currently in. Now let’s make the changes to the paper.

```
$ nano journal.md               # Change title and add co-author
$ git add journal.md
$ git commit                    # "Modify title and add John as co-author"
```

If we now want to work in our `master` branch. We can switch back by using:

```
$ git checkout master
```

```
Switched to branch 'master'
```

Key Points

- `git branch` creates a new branch
- Use feature branches for new ideas and fixes, before merging into `master`
- merging does not delete any branches

(../02-history/)

(../04-remote/)