‹ (../04-remote/)

Introduction to Git and GitHub (../)

› (../06-pull-requests/)

Collaborating with a remote repository

Overview

Teaching: 25 min
Exercises: 15 min
Questions

- How do I update my local repository with changes from the remote?
- How can I collaborate using Git?

Objectives

- Understand how to pull changes from remote repository
- Understand how to resolve merge conflicts

Pulling changes from a remote repository

Now when we have a remote repository, we can share it and collaborate with others (and we can also work from multiple locations: for example from a laptop and a desktop in the lab). But how do we get the latest changes? One way is simply to clone the repository every time– but this is inefficient, especially if our repository is very large! So, Git allows us to get the latest changes down from a repository.

We'll first do a "dry run" of pulling changes from a remote repository and then we'll work in pairs for some real-life practice.

First, let us leave our current local repository,

```
$ cd ..
$ ls
```

```
$ papers
```

And let us clone our repository again, but this time specify the local directory name,

```
$ git clone https://github.com/<USERNAME>/papers.git laptop_papers
Cloning into 'laptop_papers'...
```

So we now have two clones of our repository,

```
$ ls
```

```
$ papers laptop_papers
```

Let's pretend these clones are on two separate machines! So we have 3 versions of our repository - our two local versions, on our separate machines (we're still pretending!) and one on GitHub. So let's go into one of our clones, add a figures section, commit the file and push these changes to GitHub:

```
$ cd papers                    # Switch to the 'papers' directory
$ nano journal.md              # Add figures section
$ git add journal.md
$ git commit -m "Add figures"
$ git push
```

Now let's change directory to our other repository and `fetch` the commits from our remote repository,

```
$ cd ../laptop_papers          # Switch to the other directory
$ git fetch
```

We can now see what the differences are by doing,

```
$ git diff origin/master
```

which compares our `master` branch with the `origin/master` branch which is the name of the `master` branch in `origin` which is the alias for our cloned repository, the one on GitHub.

We can then `merge` these changes into our current repository, but given the history hasn't diverged, we don't get a merge commit — instead we get a *fast-forward* merge.

```
$ git merge origin/master
```

```
Updating 0cc2a2d..7c239c3
Fast-forward
 journal.md | 4 ++++
 1 file changed, 4 insertions(+)
```

We can inspect the file to confirm that we have our changes.

```
$ cat journal.md
```

As a short-hand, we can do a `git pull` which does a `git fetch` then a `git merge`. Next we will update our repo using `pull`, but this time starting in the *laptop_papers* folder (you should already be in the *laptop_papers* folder). Let's write the conclusions:

```
$ nano journal.md              # Write Conclusions
$ git add journal.md
$ git commit -m "Write Conclusions" journal.md
$ git push origin master
$ cd ../papers                 # Switch back to the papers directory
$ git pull origin master       # Get changes from remote repository
```

This is the same scenario as before, so we get another fast-forward merge.

We can check that we have our changes:

```
$ cat journal.md
$ git log
```

📌 Fetch vs pull

If `git pull` is a shortcut for `git fetch` followed by `git merge` then, why would you ever want to do these steps separately?

Well, depending on what the commits on the remote branch contain, you might want to e.g., abandon your local commits before merging.

Fetching first lets you inspect the changes before deciding what you want to do with them.

Conflicts and how to resolve them

Let's continue to pretend that our two local, cloned, repositories are hosted on two different machines. You should still be in the original *papers* folder. Add an affiliation for each author. Then push these changes to our remote repository:

```
$ nano journal.md               # Add author affiliations
$ git add journal.md
$ git commit -m "Add author affiliations"
$ git push origin master
```

Now let us suppose, at a later date, we use our other repository (on the laptop) and we want to change the order of the authors.

The remote branch `origin/master` is now ahead of our local `master` branch on the laptop, because we haven't yet updated our local branch using `git pull` .

```
$ cd ../laptop_papers           # Switch directory to other copy of our repository
$ nano journal.md               # Change order of the authors
$ git add journal.md
$ git commit -m "Change the first author" journal.md
$ git push origin master
```

```
To https://github.com/<USERNAME>/papers.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://github.com/<USERNAME>/papers.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Our push fails, as we've not yet pulled down our changes from our remote repository. Before pushing we should always pull, so let's do that…

```
$ git pull origin master
```

and we get:

```
Auto-merging journal.md
CONFLICT (content): Merge conflict in journal.md
Automatic merge failed; fix conflicts and then commit the result.
```

As we saw earlier, with the fetch and merge, `git pull` pulls down changes from the repository and tries to merge them. It does this on a file-by-file basis, merging files line by line. We get a **conflict** if a file has changes that affect the same lines and those changes can't be seamlessly merged. We had this situation before in the *branching* episode when we merged a *feature* branch into *master*. If we look at the status,

```
$ git status
```

we can see that our file is listed as *Unmerged* and if we look at *journal.md*, we see something like:

```
<<<<<<< HEAD
Author
G Capes, J Smith
=======
author
J Smith, G Capes
>>>>>>> 1b55fe7f23a6411f99bf573bfb287937ecb647fc
```

The mark-up shows us the parts of the file causing the conflict and the versions they come from. We now need to manually edit the file to *resolve* the conflict. Just like we did when we had to deal with the conflict when we were merging the branches.

We edit the file. Then commit our changes. Now, if we *push* …

```
$ nano journal.md               # Edit file to resolve merge conflict
$ git add journal.md            # Stage the file
$ git commit                    # Commit to mark the conflict as resolved
$ git push origin master
```

… all goes well. If we now go to GitHub and click on the "Overview" tab we can see where our repository diverged and came together again.

This is where version control proves itself better than DropBox or GoogleDrive, this ability to merge text files line-by-line and highlight the conflicts between them, so no work is ever lost.

We'll finish by pulling these changes into other copy of the repo, so both copies are up to date:

```
$ cd ../papers                  # Switch to 'papers' directory
$ git pull origin master        # Merge remote branch into local
```

Collaborating on a remote repository

In this exercise you should work with a partner or a group of three. One of you should give access to your remote repository on GitHub to the others (by selecting `Settings -> Collaborators`).

Now those of you who are added as collaborators should clone the repository of the first person on your machines. (make sure that you **don't clone into a directory that is already a repository**!)

Each of you should now make some changes to the files in the repository. Commit the changes and then push them back to the remote repository.
Remember to pull changes before you push.

Creating branches and sharing them in the remote repository

Working with the same remote repository, each of you should create a new branch locally and push it back to the remote repo.

Each person should use a different name for their local branch. The following commands assume your new branch is called `my_branch`, and your partner's branch is called `their_branch` — you should substitute the name of your new branch and your partner's new branch.

```
$ git checkout -b my_branch            # Create and check out a new branch.
                                       # Substitute your local branch name for 'my_branc
h'.
```

Now create/edit a file, and then commit your changes.

```
$ git push origin my_branch            # Push your new branch to remote repo.
```

The other person should check out local copies of the branches created by others (so eventually everybody should have the same number of branches as the remote repository).

To fetch new branches from the remote repository (into your local `.git` database):

```
$ git fetch origin
```

```
Counting objects: 3, done.  remote:
Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 2 (delta 0) Unpacking objects: 100% (3/3), done.
From  https://github.com/gcapes/papers
9e1705a..640210a master -> origin/master
* [new branch] their_branch -> origin/their_branch
```

Your local repository should now contain all the branches from the remote repository, but the `fetch` command doesn't actually update your local branches.

The next step is to check out a new branch locally to track the new remote branch.

```
$ git checkout their_branch
```

```
Branch their_branch set up to track remote branch their_branch from origin.
Switched to a new branch 'their_branch'
```

Key Points

- `git pull` to integrate remote changes into local copy of repository

(../04-remote/)

› (../06-pull-requests/)

Edit on GitHub (https://github.com/emdupre/git-course/edit/gh-pages/_episodes/05-remote-collaboration.md) / Contributing (https://github.com/emdupre/git-course/blob/gh-pages/CONTRIBUTING.md) / Source (https://github.com/emdupre/git-course/) / Cite (https://github.com/emdupre/git-course/blob/gh-pages/CITATION) / Contact (mailto:elizabeth.dupre@mail.mcgill.ca?subject=Git course)