# Merging and Merge Conflicts

When we want to move changes from one branch to another, we use the `git merge` command. Depending on the history of our commits, we can merge two different ways:

1. Fast forward
2. Recursive

In the previous section, we saw a fast forward merge, which is when git can easily tell when the commits happened and "put" one set of commits on top of another chronologically.

When different commits happen at different times on two branches, and git can not easily determine what order these commits happened in, a `recursive` merge needs to happen. You don't need to know the details of how the `recursive` strategy works; just know that it's an algorithm git uses to try to merge branches when a simple fast-forward merge won't suffice. (If you'd like to learn more, type `man git-merge` to check out the documentation.)

Things get even worse when you commit changes to the same file on two different branches. In that case, Git does not know which commit to go with so it creates a merge conflict. This is basically Git's way of saying "Hey human, you're asking me to put conflicting files in the commit history; I don't know how to resolve these conflicts, so you take care of it and let me know when you're done."

Let's see an example by creating our very own merge conflict! Starting in our home directory:

```
mkdir merge_conflicts
cd merge_conflicts
git init
echo first > first.txt
git add .
git commit -m "first commit"


git checkout -b new_branch
```

```
echo second > second.txt
git add .
git commit -m "adding second.txt"

git checkout master
echo something_else > second.txt
git add .
git commit -m "adding second.txt on the master branch"

git merge new_branch
```

This final command should output

```
Auto-merging second.txt
CONFLICT (add/add): Merge conflict in second.txt
Recorded preimage for 'second.txt'
Automatic merge failed; fix conflicts and then commit the result.
```

Looks like we have an issue because when we tried to merge the `new_branch`, git does not know which `second.txt` file to use! Both branches have a file with the same name, but different contents.

If we take a look at our `second.txt` file (`cat second.txt`) we will see this

```
something_else
=======
second
>>>>>>> new_branch
```

What we see is the contents in HEAD, where master is, followed by the contents in `new_branch`. Let's open up this file in our text editor and change it so that it looks like this:

```
second
```

This is our way of letting Git know that we want to keep the text from `new_branch` and discard the text from `master`.

Then let's go back to the terminal and run

```
git add .
git commit -m "fixing merge conflict"
```

From here you should be good to go.

Git is a great tool, but there are times when it won't know how to merge things for you. The first time you see a merge conflict it can look a little intimidating, but Git will put the two conflicting pieces of text right on top of one another; the only thing you need to do is decide what text you want to keep, delete the character separating the two options (`=======` and `>>>>>>>`), and add and commit the results of your manual merge.

Previous next