Merge Commands

coursera.org/learn/git-distributed-development/supplement/0Lhqw/merge-commands

Merging the **devel** branch into the current main branch is as simple as doing:

```
1
```

```
2
```

```
$ git checkout main
```

```
$ git merge devel
```

where the first step is only necessary if you are not already working on the main branch.

While not absolutely mandatory, the best practice is to tidy up before merging. This means committing any changes that been staged, getting rid of junk, etc. This can help minimize later confusion. You can check your current state in this regard with the command:

```
1
```

```
$ git status
```

If you had no conflicts, the preceding **merge** command will have given a happy report, and your main branch now includes everything that was in the **devel** branch; it has been synced up with the ongoing development. However, let's see what happens if the merge senses a conflict.

To examine this, we create **main** and **devel** branches that are identical except that in the **main** branch we create **file1** which has one line, saying **file901**. In the devel branch this file has the line **file701**. Following the above merge procedure, we get the result:

```
1
```

```
2
```

```
3

4

5

$ git merge devel


Auto-merged file1

CONFLICT (content): Merge conflict in file1

Automatic merge failed; fix conflicts and then commit the result.
```

Listing the files in the working branch with the command:

```
1

2

3

4

5

6

7

$ git ls-files


file1

file1

file1

file2

file3
```

shows the funny result of **file1** being listed three times! If we look at the contents of this file after the attempted merge, we see the curious result:

```
1
2
3
4
5
6
7
$ cat file1


<<<<<<< HEAD:file1
file901
=======
file701
>>>>>>> devel:file1
```

a three-way diff showing the problems.

Note that all other changes resulting from the merge have gone through successfully. The merge command will tell you specifically about each problem.

There are two basic approaches you can take to fixing the problems. The first is revert the merge with **git reset**, work on the conflicts in either of the two branches until no conflict is expected to result from a merge, and try again.

If you have made a mess of things with a premature or accidental merge, it is easy to revert back to where you were with:

```
1
$ git reset --hard main
```

The second approach is to work on the attempted merge file, the third copy that has all the funny markers in it. You can edit it to your heart's content to reflect what should be the product of the merge, and then simply commit, as in:

```
1

2

3

4

5

6

7

$ git add file1

$ git commit -m "A message for the merge"

$ git ls-files


file1

file2

file3
```

Note the odd appearance of three versions of **file1** is gone and the merge is now complete.

Which approach you take is a judgment call that depends on the size of the merge set, the number of conflicts that develop, etc., and it is totally up to you. The end result should be the same.