Merging

Contents

Overview

Joining the history of two or more branches through `git merge` incorporates changes into the current branch. This command is
used by `git pull` to incorporate changes from a different repository, as well as to merge changes from one branch into
another.

If we assume that we have a history like this, and the current branch is **main** :

```
      E---F---G issue
     /
A---B---C---D main
```

Executing `git merge issue` would add changes E, F and G into main and result in a new commit.
This new commit would end up in the following history.

```
      E---F---G issue
     /         \
A---B---C---D---H main
```

Merge conflicts

Merge conflicts happen when more than one person has edited a file, and the line numbers that were edited are the same
. It can also happen if someone deleted a file that another person was working on.

This conflict only affects the person performing the merge, and the rest of the team wouldn't be affected by it.

If a merge conflict happens, *Git* will automatically halt the merge process and mark the file, or files, that are
conflicting. It is then up to the developer to resolve them.

Tutorial

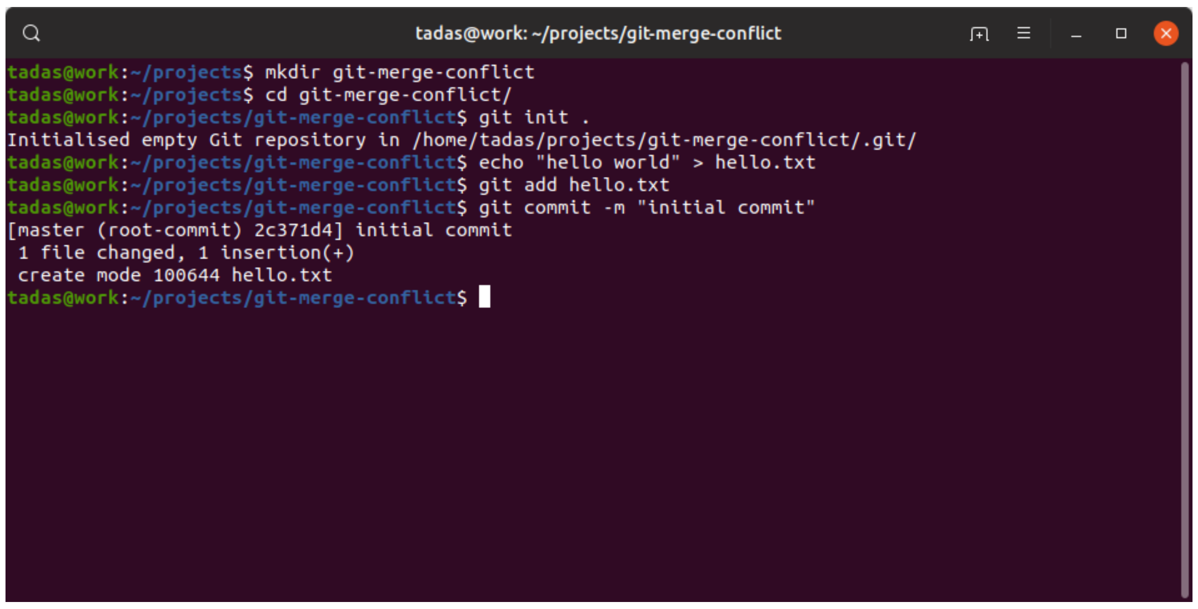
Handling Merge Conflicts

You will now go through the steps required to cause a merge conflict.

Initialise a Repository for Testing Merge Conflicts

1. Open a terminal
2. Create a new directory by executing `mkdir git-merge-conflict`
3. Change directory by executing `cd git-merge-conflict`
4. Initialise this directory as a git repository by executing `git init .`

5. Create a new text file *hello.txt* in the directory
6. Add some text, such as "hello world", to the *hello.txt* file. Save and close the file
7. Now you need to make git track changes for the *hello.txt* file, which can be done by executing the `git add hello.txt` command
8. Now you need to create a save point, which is known as a commit. This will have the current state of the *hello.txt file in it. Execute the `git commit -m "initial commit"` command to achieve this



```
tadas@work:~/projects$ mkdir git-merge-conflict
tadas@work:~/projects$ cd git-merge-conflict/
tadas@work:~/projects/git-merge-conflict$ git init .
Initialised empty Git repository in /home/tadas/projects/git-merge-conflict/.git/
tadas@work:~/projects/git-merge-conflict$ echo "hello world" > hello.txt
tadas@work:~/projects/git-merge-conflict$ git add hello.txt
tadas@work:~/projects/git-merge-conflict$ git commit -m "initial commit"
[master (root-commit) 2c371d4] initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 hello.txt
tadas@work:~/projects/git-merge-conflict$
```

Create a Branch with a Conflict

Now that you have a repository and a main branch with a file on it, the next step is to create a new branch to
use to cause the merge conflict.

1. Checkout a new branch by executing `git checkout -b new-branch`
2. Open the text file *hello.txt* and add a new line of text to it, such as "making a change to the file". Your text
   file should now look like this:

```
hello world
making a change to the file
```

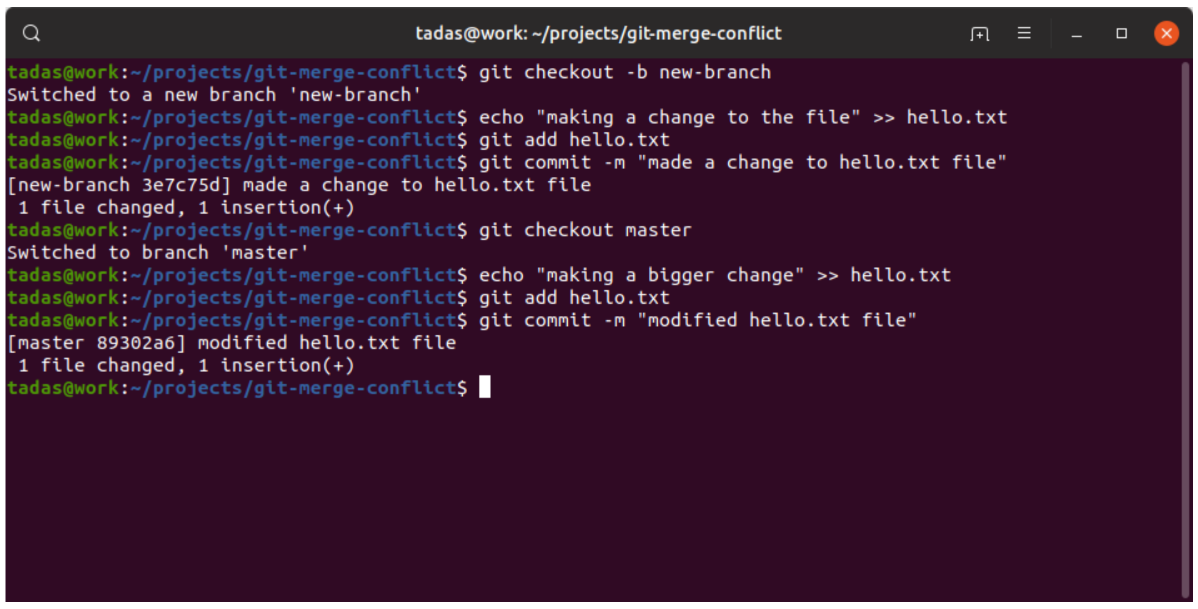
3. Let's ensure Git keeps track of the hello.txt file that we made a change to. This can be done by executing the `git add hello.txt` command. Now you need to commit again, but this time with a message that reflects the change made - `git commit -m "made a change to hello.txt file"`. The change we made will now try to override the changes in main branch, for the text file *hello.txt*.
4. A change to the text file *hello.txt* is now required before we can cause a merge conflict. Let's go back to
   our *main* branch, by executing the `git checkout main` command
5. Open the text file *hello.txt* and add a new line of text to it, such as "making a bigger change". Your
   text file on main should now look like this:

```
hello world
making a bigger change
```

   The *hello.txt* on the *new-branch* branch should look like this:

```
hello world
making a change to the file
```

   Let's commit the change we made to the *hello.txt* file to the main branch, by executing `git add hello.txt` followed by
   `git commit -m "modified hello.txt file"`.

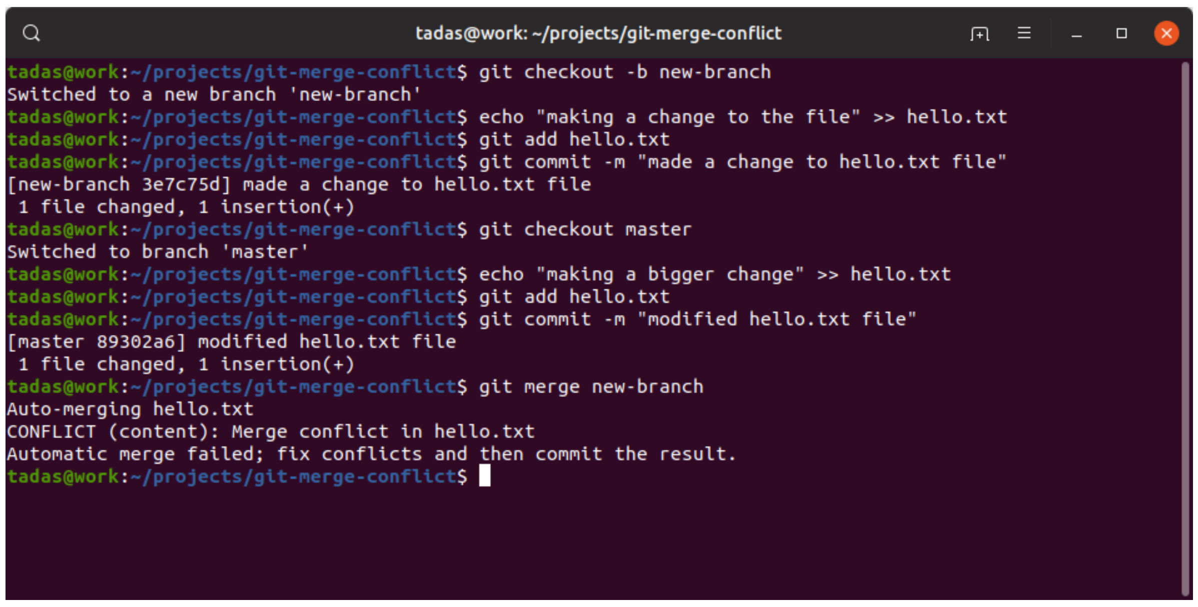


```
tadas@work:~/projects/git-merge-conflict$ git checkout -b new-branch
Switched to a new branch 'new-branch'
tadas@work:~/projects/git-merge-conflict$ echo "making a change to the file" >> hello.txt
tadas@work:~/projects/git-merge-conflict$ git add hello.txt
tadas@work:~/projects/git-merge-conflict$ git commit -m "made a change to hello.txt file"
[new-branch 3e7c75d] made a change to hello.txt file
 1 file changed, 1 insertion(+)
tadas@work:~/projects/git-merge-conflict$ git checkout master
Switched to branch 'master'
tadas@work:~/projects/git-merge-conflict$ echo "making a bigger change" >> hello.txt
tadas@work:~/projects/git-merge-conflict$ git add hello.txt
tadas@work:~/projects/git-merge-conflict$ git commit -m "modified hello.txt file"
[master 89302a6] modified hello.txt file
 1 file changed, 1 insertion(+)
tadas@work:~/projects/git-merge-conflict$ 
```

Attempt to Merge the New Branch

Now, we want to merge changes from the *new-branch* branch to the *main* branch. However, Git won't be able to figure out which version of the second line to use - it will create a merge conflict, which the developer will be responsible for resolving.

1. Let's go ahead and actually cause the merge conflict, by executing the `git merge new-branch` command. You should get an output similar to this:



```
tadas@work:~/projects/git-merge-conflict$ git checkout -b new-branch
Switched to a new branch 'new-branch'
tadas@work:~/projects/git-merge-conflict$ echo "making a change to the file" >> hello.txt
tadas@work:~/projects/git-merge-conflict$ git add hello.txt
tadas@work:~/projects/git-merge-conflict$ git commit -m "made a change to hello.txt file"
[new-branch 3e7c75d] made a change to hello.txt file
 1 file changed, 1 insertion(+)
tadas@work:~/projects/git-merge-conflict$ git checkout master
Switched to branch 'master'
tadas@work:~/projects/git-merge-conflict$ echo "making a bigger change" >> hello.txt
tadas@work:~/projects/git-merge-conflict$ git add hello.txt
tadas@work:~/projects/git-merge-conflict$ git commit -m "modified hello.txt file"
[master 89302a6] modified hello.txt file
 1 file changed, 1 insertion(+)
tadas@work:~/projects/git-merge-conflict$ git merge new-branch
Auto-merging hello.txt
CONFLICT (content): Merge conflict in hello.txt
Automatic merge failed; fix conflicts and then commit the result.
tadas@work:~/projects/git-merge-conflict$ 
```

2. Let's take a look at the contents of the *hello.txt* file now, which should look similar to this:

```
hello world
<<<<<<< HEAD
making a bigger change
=======
making a change to the file
>>>>>>> new-branch
```

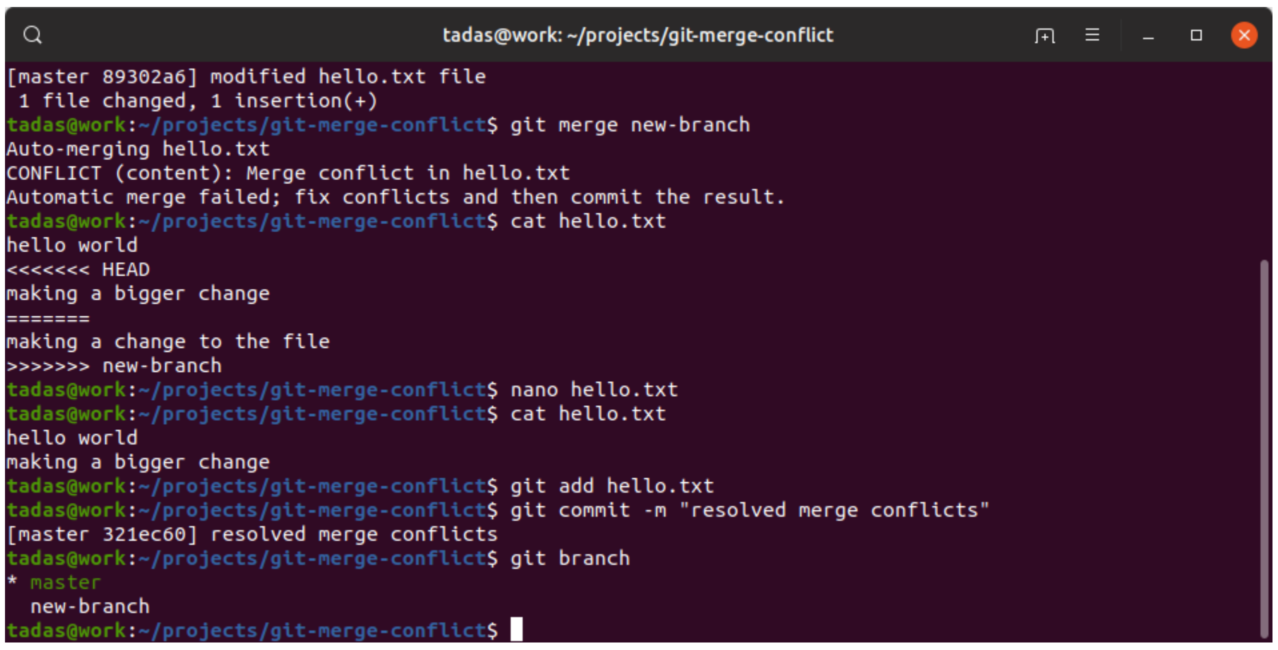
   The main thing to recognise is that the first line (`hello world`) doesn't have a conflict, but there is a conflict between the second line of the *main* and *new-branch* branches.

Resolving the Conflict

To resolve the merge conflict, there are a couple of steps needed:
Firstly, decide which second line to keep out of `making a bigger change` or `making a change to the file`. You could also choose something entirely different, such as keeping or deleting both lines.
Secondly, delete the lines *Git* added, to show where the merge conflict is happening `<<<<<<< HEAD`, `=======`, `>>>>>>> new-branch`.
Let's say you decide to keep the second line that's currently in *main* branch. After cleaning up the file, it should look like this:

```
hello world
making a bigger change
```

Next, you need to save the changes made. This can be done by executing the `git add hello.txt` command, followed by the `git commit -m "resolved merge conflict"` command.



```
[master 89302a6] modified hello.txt file
 1 file changed, 1 insertion(+)
tadas@work:~/projects/git-merge-conflict$ git merge new-branch
Auto-merging hello.txt
CONFLICT (content): Merge conflict in hello.txt
Automatic merge failed; fix conflicts and then commit the result.
tadas@work:~/projects/git-merge-conflict$ cat hello.txt
hello world
<<<<<<< HEAD
making a bigger change
=======
making a change to the file
>>>>>>> new-branch
tadas@work:~/projects/git-merge-conflict$ nano hello.txt
tadas@work:~/projects/git-merge-conflict$ cat hello.txt
hello world
making a bigger change
tadas@work:~/projects/git-merge-conflict$ git add hello.txt
tadas@work:~/projects/git-merge-conflict$ git commit -m "resolved merge conflicts"
[master 321ec60] resolved merge conflicts
tadas@work:~/projects/git-merge-conflict$ git branch
* master
  new-branch
tadas@work:~/projects/git-merge-conflict$ 
```

Running `git status` should now indicate that there are no longer any conflicts to resolve.

Exercises

There are no exercises for this module.