Creating a merge conflict

In order to get real familiar with merge conflicts, the next section will simulate a conflict to later examine and resolve. The example will be using a Unix-like command-line Git interface to execute the example simulation.

$ mkdir git-merge-test  
$ cd git-merge-test  
$ git init .  
$ echo "this is some content to mess with" > merge.txt  
$ git add merge.txt  
$ git commit –am "we are commiting the inital content"  
[main (root-commit) d48e74c] we are commiting the inital content  
1 file changed, 1 insertion(+)  
create mode 100644 merge.txt

This code example executes a sequence of commands that accomplish the following.

* Create a new directory named git-merge-test, change to that directory, and initialize it as a new Git repo.
* Create a new text file merge.txt with some content in it.
* Add merge.txt to the repo and commit it.

Now we have a new repo with one branch main and a file merge.txt with content in it. Next, we will create a new branch to use as the conflicting merge.

$ git checkout -b new\_branch\_to\_merge\_later  
$ echo "totally different content to merge later" > merge.txt  
$ git commit -am"edited the content of merge.txt to cause a conflict"  
[new\_branch\_to\_merge\_later 6282319] edited the content of merge.txt to cause a conflict  
1 file changed, 1 insertion(+), 1 deletion(-)

The proceeding command sequence achieves the following:

* create and check out a new branch named new\_branch\_to\_merge\_later
* overwrite the content in merge.txt
* commit the new content

With this new branch: new\_branch\_to\_merge\_later we have created a commit that overrides the content of merge.txt

git checkout main  
Switched to branch 'main'  
echo "content to append" >> merge.txt  
git commit –am "appended content to merge.txt"  
[main 24fbe3c] appended content to merge.tx  
1 file changed, 1 insertion(+)

This chain of commands checks out the main branch, appends content to merge.txt, and commits it. This now puts our example repo in a state where we have 2 new commits. One in the main branch and one in the new\_branch\_to\_merge\_later branch. At this time lets git merge new\_branch\_to\_merge\_later and see what happen!

$ git merge new\_branch\_to\_merge\_later  
Auto-merging merge.txt  
CONFLICT (content): Merge conflict in merge.txt  
Automatic merge failed; fix conflicts and then commit the result.

BOOM 💥. A conflict appears. Thanks, Git for letting us know about this!

How to identify merge conflicts

As we have experienced from the proceeding example, Git will produce some descriptive output letting us know that a CONFLICT has occcured. We can gain further insight by running the [git status](https://www.atlassian.com/git/tutorials/inspecting-a-repository) command

$ git status  
On branch main  
You have unmerged paths.  
(fix conflicts and run "git commit")  
(use "git merge --abort" to abort the merge)  
  
Unmerged paths:  
(use "git add <file>..." to mark resolution)  
  
both modified:   merge.txt

The output from git status indicates that there are unmerged paths due to a conflict. The merge.text file now appears in a modified state. Let's examine the file and see whats modified.

$ cat merge.txt  
<<<<<<< HEAD  
this is some content to mess with  
content to append  
=======  
totally different content to merge later  
>>>>>>> new\_branch\_to\_merge\_later

Here we have used the cat command to put out the contents of the merge.txt file. We can see some strange new additions

* <<<<<<< HEAD
* =======
* >>>>>>> new\_branch\_to\_merge\_later

Think of these new lines as "conflict dividers". The ======= line is the "center" of the conflict. All the content between the center and the <<<<<<< HEAD line is content that exists in the current branch main which the HEAD ref is pointing to. Alternatively all content between the center and >>>>>>> new\_branch\_to\_merge\_later is content that is present in our merging branch.

How to resolve merge conflicts using the command line

The most direct way to resolve a merge conflict is to edit the conflicted file. Open the merge.txt file in your favorite editor. For our example lets simply remove all the conflict dividers. The modified merge.txt content should then look like:

this is some content to mess with  
content to append  
totally different content to merge later

Once the file has been edited use git add merge.txt to stage the new merged content. To finalize the merge create a new commit by executing:

git commit -m "merged and resolved the conflict in merge.txt"

Git will see that the conflict has been resolved and creates a new merge commit to finalize the merge.

Git commands that can help resolve merge conflicts

General tools

git status

The status command is in frequent use when a working with Git and during a merge it will help identify conflicted files.

git log --merge

Passing the --merge argument to the git log command will produce a log with a list of commits that conflict between the merging branches.

git diff

diff helps find differences between states of a repository/files. This is useful in predicting and preventing merge conflicts.

Tools for when git fails to start a merge

git checkout

checkout can be used for *undoing* changes to files, or for changing branches

git reset --mixed

reset can be used to undo changes to the working directory and staging area.

Tools for when git conflicts arise during a merge

git merge --abort

Executing git merge with the --abort option will exit from the merge process and return the branch to the state before the merge began.

git reset

Git reset can be used during a merge conflict to reset conflicted files to a know good state