Git merge conflicts

Version control systems are all about managing contributions between multiple distributed authors (usually developers). Sometimes multiple developers may try to edit the same content. If Developer A tries to edit code that Developer B is editing a conflict may occur. To alleviate the occurrence of conflicts developers will work in separate isolated branches. The `git merge` command's primary responsibility is to combine separate branches and resolve any conflicting edits.

Understanding merge conflicts

how to automatically integrate new changes.

Conflicts generally arise when two people have changed the same lines in a file, or if one developer deleted a file while another developer was modifying it. In these cases, Git cannot automatically determine what is correct. Conflicts only affect the developer conducting the merge, the rest of the team is unaware of the conflict. Git will mark the file as being conflicted and halt the merging process. It is then the developers' responsibility to resolve the conflict.

Types of merge conflicts

A merge can enter a conflicted state at two separate points. When starting and during a merge process. The following is a discussion of how to address each of these conflict scenarios.

Git fails to start the merge

A merge will fail to start when Git sees there are changes in either the working directory or staging area of the current project. Git fails to start the merge because these pending changes could be written over by the commits that are being merged in. When this happens, it is not because of conflicts with other developer's, but conflicts with pending local changes. The local state will need to

```
error: Entry '<fileName>' not uptodate. Cannot
```

Git fails during the merge

A failure DURING a merge indicates a conflict between the current local branch and the branch being merged. This indicates a conflict with another developers code. Git will do its best to merge the files but will leave things for you to resolve manually in the conflicted files. A mid-merge failure will output the following error message:

```
error: Entry '<fileName>' would be overwritten
```

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
```

```
 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 `master` 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 lat
$ git commit -am"edited the content of merge.t
[new_branch_to_merge_later 6282319] edited the
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`

With this new branch: `new_branch_to_merge_later` we have created a commit that overrides the content of `merge.txt`

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

This chain of commands checks out the `master` 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 `master` 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
```

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

How to identify merge conflicts

Git will produce some descriptive output letting us know that a CONFLICT has occcured. We can gain further insight by running the `git status` command

```
$ git status
On branch master
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`
- `=======`

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 master 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
```

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

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

Merge conflicts can be an intimidating experience. Luckily, Git offers powerful tools to help navigate and resolve conflicts. Git can handle most merges on its own with automatic merging features. A conflict arises when two separate branches have made edits to the same line in a file, or when a file has been deleted in one branch but edited in the other. Conflicts will most likely happen when working in a team environment.

There are many tools to help resolve merge conflicts. Git has plenty of command line tools we discussed here. For more detailed information on these tools visit stand-alone pages for git log, git reset, git status, git checkout, and git reset. In addition to the Git, many third-party tools offer streamlined merge conflict support features.

Ready to try branching?

Try this interactive tutorial.

Get started now

Next up:

START NEXT TUTORIAL

Ready to try branching?

Try this interactive tutorial.

Get started now

Powered By

Atlassian

Enter Your Email For Git News