Attempting to Apply Cherry-Pick

Learn why a cherry-pick might not work and about the “three-way merge.”

We'll cover the following

A simple branched repository

This should be fairly routine by now. You’re going to set up a Git repository with two branches and some simple changes in them. Once that’s done, you’re going to try to cherry-pick a change from the `abranch` branch and apply it to the `master` branch.

```
mkdir lgthw_patch_and_apply
cd lgthw_patch_and_apply
git init
touch afile
git add afile
git commit -m 'afile added'
echo First change, on master >> afile
git commit -am 'First change, on master added'
git branch abranch
echo Second change, on master >> afile
git commit -am 'Second change, on master added'
git checkout abranch
echo First change, on abranch >> afile
git commit -am 'First change, on abranch added'
echo Second change, on abranch >> afile
echo New file, on abranch >> newfile
git add newfile
git commit -am 'Second change, on abranch added'
git tag abranchtag
git checkout master
```

Terminal 1

Terminal

Click to Connect...

Now that you've typed that in, use `git log` to look at the repository that you've created and all its changes. Make sure you look at it with the `--all`, `--graph`, and `--decorate` flags. Then check the changes at each commit with the `--patch` flag.

It's important to get a grip on the repository that you are looking at before you try to perform surgery on it. This is what we're about to do as we try and cherry-pick a change. Obviously, this repository is simplified but practicing this will help you as you move forward.

A simple cherry pick

Before you type it in, think about what you expect the following `cherry-pick` command to do. You may be surprised by what happens, and the more surprised you are, the more likely you are to learn something useful!

First, take a look at the tag `abranchtag` to see what you're going to cherry-pick:

```
git show abranchtag
```

Terminal 1

Terminal

You should see two changes: the addition of the "Second change, on abranch" line and a new file called `newfile`.

Now you're going to cherry-pick the specific tag referenced by `abranchtag`. Write down what you think will happen, and then type in:

```
git cherry-pick abranchtag
```

Did that do what you expected?

Take a look at the `afile` file. Where did the `First change, on abranch` line come from? That wasn't in the `git show` output!

You might be smarter than me, but I was not expecting to see that in there. I was expecting a conflict but not the `First change, on abranch` within it. I also expected to see the file `newfile` because that file was added on that commit.

In other words, I expected it to just try and apply the contents of `git show` and not introduce anything else to the mix.

Instead, it *did* this::

- Tried to apply the change in the `abranchtag` commit
- Saw that there was a conflict
- Performed a three-way merge

Before showing you how to only apply the changes in the `abranchtag` commit as I originally intended, you will cover three-way merges first.

'Falling back' to a three-way merge

Picture what the commit history looked like before you did the cherry-pick.

```
* 93c07a5 (tag: abranchtag, abranch) Second change, on abranch added
* 7fc9dcc First change, on abranch added
| * 2348f4c (HEAD -> master) Second change, on master added
|/
* 9c3f1af First change, on master added
* 9c01ac0 afile added
```

The `HEAD` is pointed at the `master` branch. The `abranch` branch is pointed at the same commit as the `abranchtag` tag.

Apply the cherry-pick the cherry-pick of the `abranchtag` commit (which has an ID of `93c07a5` here but will be different for you) to the `master` branch (which is at `2348f4c` here).

What Git does is go to the first common ancestor (here, it is `9c3f1af First change, on master added`) and sees what has changed between the `abranchtag` commit you want to apply and the master branch you are applying to. Conflicts are found. Git gives you *all* the changes on both branches from the first common ancestor and asks you to sort it out.

In other words, Git says: “I can’t apply that individual commit here without any conflicts, so I’m going to apply *all* the diffs between these two points, and let you figure out what’s going on.”

This is why it is called a “three-way merge.” The merge compares the changes between the source (`abranchtag`), the target (the `master` branch), and the first common ancestor (`9c3f1af` above).

> **Note**: It *only* does this if there’s a conflict. If there is no conflict in the change, you cherry-pick, and then the three-way merge is not invoked. If you don’t believe me, then start over, but do not run the `git commit -am 'Second change, on abranch added'` line and see what happens.

Back
Introduction: Cherry-Picking and Thre...

Next
Generate and Apply Patch

Mark as Completed

Report an Issue

Ask a Question
(https://discuss.educative.io/tag/attempting-to-apply-cherry-pick__cherry-picking-and-three-way-merges__learn-git-the-hard-way)