A new way to edit a stack of commits

JUN WU · FRIDAY, AUGUST 26, 2016

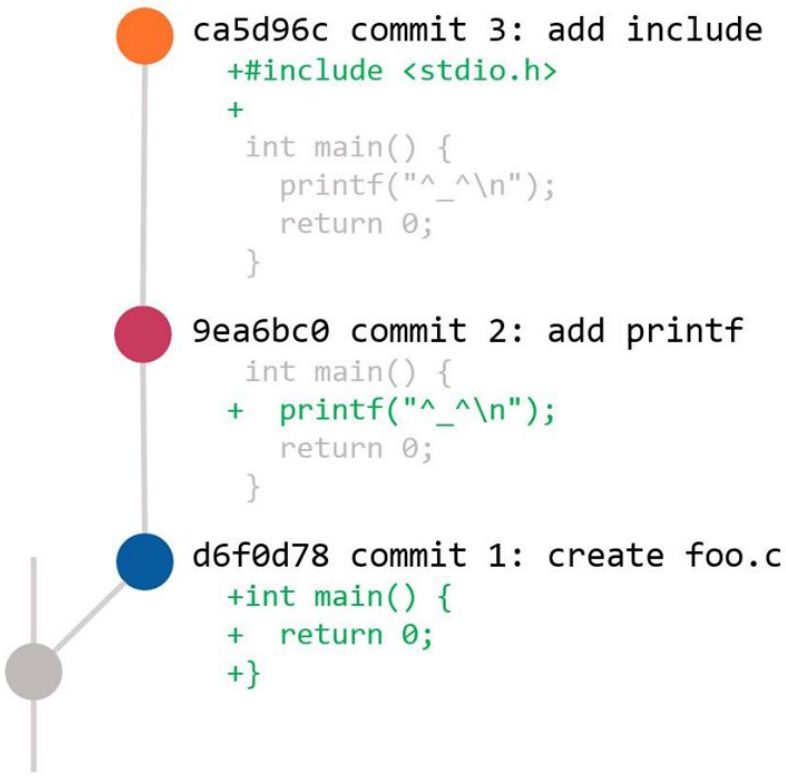
tl;dr The new `hg absorb` (or `hg amend --stack`) command can incorporate changes to correlated commits in a stack.

The problem

Imagine there is a stack of commits. What will you do if you want to fix a typo in the bottom or the middle of the stack?

Let's look at an example shown in the picture to the right. The stack consists of 3 commits:

- The first commit adds a `foo.c` with an empty `main` function.
- The second one adds a `printf`.
- The third one adds the missing `#include`.



An example stack of 3 commits

You `arc diff` the stack and get back review comments:

- Prefer `puts` to `printf`, as it's a plain string without formats.
- Use `EXIT_SUCCESS` instead of `0`, and add `#include <stdlib.h>`.

How do you change `printf` to `puts` in commit 2 (`9ea6bc0`), while your working copy is on top of commit 3 (`ca5d96c`)? And what about other changes?

The old solutions

Consider the fact that `hg amend` changes the "current" commit, the most direct way is to run `hg update 9ea6bc` to move the "current" commit there, make changes, and run `hg amend` to get a new commit. Let's name the new commit `e7cb488` . Then we need `hg rebase -s ca5d96c -d e7cb488` to move commit 3 to the new place. The rebase operation may also be done by using `--rebase` during `hg amend` , or running `hg amend --fixup` or `hg evolve` afterwards depending on your config.

Using histedit, there are other choices: Run `hg histedit` , replace `pick 9ea6bc0` to `edit 9ea6bc0` to make histedit stop there and edit the commit. Then use `histedit --continue` which actually does the rebase job implicitly.

Alternatively, just edit the working copy directly at commit 3, and use `hg commit -i` to make small fixup commits, then use `fold` or `roll` provided by `histedit` to merge those fixup commits into previous commits.

Note that no matter which approach you choose, merge conflicts can happen - the `printf` change doesn't, but the `EXIT_SUCCESS` change will, because it touches a line that's part of the diff context in commit 2. Merge conflicts are annoying but it would be a separate topic.

To address review comments to all 3 commits, it's at least 4 commands, 1 merge conflict resolution, and you have to figure out what changes belong to what commits manually.

The obvious

Take a look at the output of `hg annotate foo.c` , and the changes we are going to make: inserting `#include <stdlib.h>` and replacing `printf` and `0` .

```
          ca5d96c: #include <stdio.h>
Insert →  ca5d96c:
          d6f0d78: int main()
          d6f0d78: {
Replace   9ea6bc0:   printf("^_^\n");
Replace   d6f0d78:   return 0;
          d6f0d78: }
```

hg annotate foo.c

It seems if we just make changes (at the top of the stack), it's

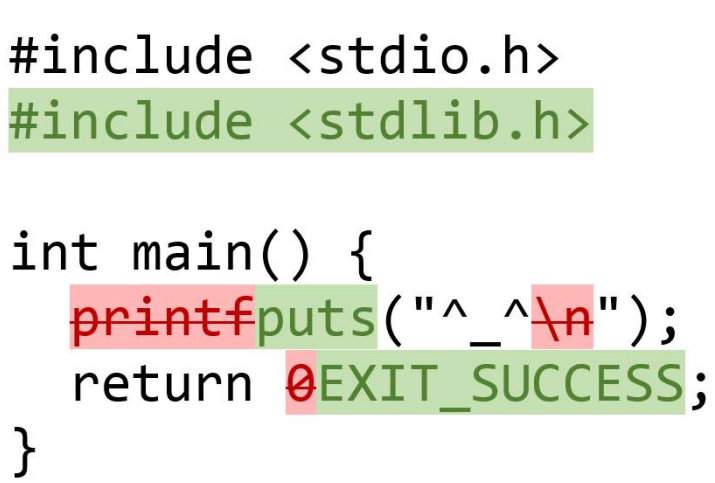
obvious where the changes should actually go, unambiguously:

- If a line is inserted between two lines introduced by `ca5d96c`, that line should probably just go to `ca5d96c`.
- If a line introduced by `9ea6bc0` is replaced (or deleted), the change should probably just go to `9ea6bc0`. Same with `d6f0d78`.

It's so obvious! Why do we need 4 commands and have to manually resolve the merge conflict.... It should be just one single command!

The new command

And that command is `hg absorb`. Think of a commit as a sponge which can absorb changes to its modified area. `hg absorb` makes your commits absorb changes in the working copy. Changes absorbed will disappear from `hg status` or `hg diff` output.



Making changes in the working copy

Using the new command, you just stay at `ca5d96c`, change `foo.c` in your working copy to address all comments, then simply run `hg absorb`.

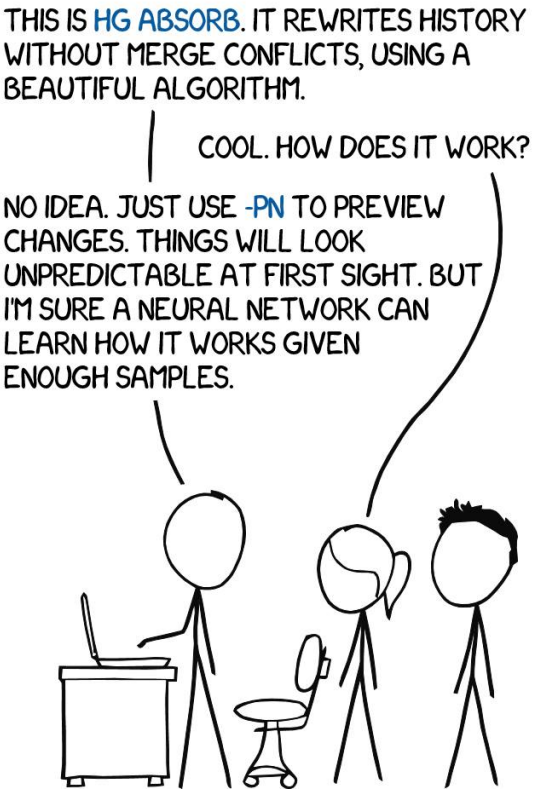
It will tell you that all modified chunks are applied (to commits). And it's done. No merge conflicts. Cheers!

Caveats

`hg absorb` is not a silver bullet. Changes can be ambiguous - unclear which "sponge" to use. The command would just leave those changes in the working copy, untouched.

The ignore-ambiguous-change behavior is different from the `amend` command which always takes all requested changes unconditionally. Despite the difference, some people still prefer `amend` as the command name. We have got that covered - you can use `amend --stack` for the same feature.

To precisely define what is "ambiguous" requires a long text - there are tricky cases. However, you can always use `hg absorb -pn` to preview what changes will be made to which commits. And just like `commit`, `split`, and `revert`, you can use `-i` to interactively select files, chunks or lines to be used.



Unpredictable algorithm
(modified from xkcd: Git)

Unlike `histedit`, the `absorb` implementation never writes to the working copy (even if you use `-i`). So your working copy is safe even if the command terminates unexpectedly.

If you have made it this far, thank you for your time. Hope this little command could make stack editing more enjoyable.