Refactoring `git rebase --interactive`

Quint Guvernator
College of William and Mary,
Williamsburg, VA

`quintus.public@gmail.com`
`http://qguv.github.io/`

March 16, 2014

1 Synopsis

I propose to refactor much of the messy `git-rebase--interactive.sh` script and to add more options to the list of options the script understands. With the guidance of the mailing list and of my mentor, I could make informed decisions as to what additional functionality is useful and what would be clutter.

2 Discrete Sub-projects

There are a number of directions one could go with this project. I aim to accomplish the following four major tasks.

2.1 Improve code readability and clarity

Right now, some hunks of code are pretty messy. For example:

```
# lines 239 through 243
ff=--ff

case "$1" in -n) sha1=$2; ff= ;; *) sha1=$1 ;; esac
case "$force_rebase" in '') ;; ?*) ff= ;; esac
output git rev-parse --verify $sha1 || die ...
```

Not only does this hunk diverge from the style guide[1], it obscures the purpose of the section, making further development difficult.

In an open-source project, it is vital for every contributor to understand clearly the purpose of the code she is editing. In my contributions, I will stick to the syntax permitted in the bash programming style guide.

2.2 Add comments

Much of the 1000-line source lacks comments. I will be using `git blame` and reading commit messages to try to grok precisely what the uncommented portions of the code do.

This will most likely require digging through old posts in the mailing list around the time when the changes were produced. With my microproject[2], I feel I have become familiar with our mailing list and would be up to the challenge.

2.3 Add features (judiciously)

While my goals include feature additions, I will stick close to the original intention of `rebase --interactive` as described in section 5. The first features I would consider adding are those described on this[3] mailing list thread.

I have experience using Git to manage the source history of my own projects[4]. I am actively developing or maintaining the vast majority of these projects, so I would not have trouble finding "itches" to scratch with new features.

[1]`Documentation/CodingGuidelines`, lines 33–107

[2]The newest version of the patch at the time of writing is available at: `http://thread.gmane.org/gmane.comp.version-control.git/244159`.

[3]`http://marc.info/?t=139340200700001&r=1&w=2`

[4]A list of projects can be found at `http://qguv.github.io`.

I also understand that new features increase the size of the project and make it harder to maintain. I will make sure the benefits of the features I add outweigh the costs associated with maintaining them. I will be in constant contact with the community to decide what additions are worth the time to develop and maintain.

2.4 Improve documentation

Although not as important as the other two sub-projects due to the quality of our existing documentation, the documentation of interactive mode could be improved. I am a native English speaker with sufficient command of the language to write clear, concise documentation.

If `rebase --interactive` does not take all summer, improving English-language documentation could be a good way to continue contributing to Git. At my university, I copy-edit our student newspaper and would be happy to spend my time on documentation.

3 Procedure for Making Changes

Feedback is vital to the success of this project. New features, especially as quickly as they may come with this project, need lots of eyes to make sure they're ready for production. As such, I will need to be in constant contact with both the mailing list and the IRC channel. My workflow for this project would look something like this:

1. Decide what to improve: refactoring a hunk, a new feature, or documentation
2. Start writing a mockup or early draft of the feature or improvement
3. Communicate with my mentor as to my implementation
4. Improve the implementation
5. Repeat 3 and 4
6. Decide with my mentor that the feature is ready for the mailing list and post it
7. Follow up
8. Improve the implementation
9. Repeat 7 and 8
10. Rinse and repeat the whole procedure with a new task

It will be important to stay in contact with the community. I have found the mailing list to be informative, prompt, and reasonable in discussing patches to Git. I would look forward to taking part in development discussion.

4 Benefits to the Community

The intent of `rebase --interactive` was to make sanely applying patching easier. Especially considering that development work is carried out by submitting, improving, and applying patches, the Git community would also benefit from additional functionality in interactive mode.

5 Some History of `git-rebase --interactive`

In June 2006, Eric Biederman wrote on the mailing list:

> I'm trying to sort out a sane work flow for developing with a large number of highly related patches on the development side. I care because the way I think I usually fix the root cause of problems.

Johannes Schindelin had a similar problem, which he posted on the *Uni-Würzburg* mailing list. He discusses the solution below:

> While I was hacking on another issue, I realized just how often I would like to stash away a fix which is unrelated (but often triggered) by the theme of the current topic branch. Or I fix an earlier commit, which is not the tip of the branch, so I cannot `--amend` it.
>
> My common practice is to commit it nevertheless, and sort the topic branches out later, by cherry-picking my way through the commits.
>
> This is a tedious and error-prone procedure, and I often wished I knew how to use StGIT. But then, StGIT is overkill for me: on some machines I work on, there is no Python installed, I do not really need to have a history on the order and version of patches, and I do not need to preserve author and committer information.
>
> Therefore, I wrote this extremely simple script to sort out the order of commits, and possibly merging some. The script lets

> you edit the commit list upon start (reordering it, or removing commits), and then works on that list.

Soon after the proto-script was released, Johannes made the first commit in the git history to include `git-rebase--interactive.sh`:

> Don't you just hate the fact sometimes, that git-rebase just applies the patches, without any possibility to edit them, or rearrange them? With `--interactive`, git-rebase now lets you edit the list of patches, so that you can reorder, edit and delete patches.
>
> Such a list will typically look like this:
>
> ```
> pick deadbee The oneline of this commit
> pick fa1afe1 The oneline of the next commit
> ...
> ```
>
> By replacing the command `pick` with the command `edit`, you can amend that patch and/or its commit message, and by replacing it with `squash` you can tell rebase to fold that patch into the patch before that.

The shell code at this point[5] was elegant and terse. I will try to mimic the style of this early form of the script to improve readability and make the script easier to maintain.

[5] `https://github.com/git/git/blob/1b1dce4bae760248a1fc3e29548a72c446e77270/git-rebase--interactive.sh`