Improve consistency of sequencer commands

About Me

Personal Information

Name	Rohit Ashiwal
Major	Computer Science and Engineering
E-mail	<rohit.ashiwal265@gmail.com>
IRC	__rohit
Skype	rashiwal
Ph no	+91 7340 605885
Github	@r1walz
Linkedin	rohit-ashiwal
Address	DG 02, Jawahar Bhawan, IIT Roorkee, Roorkee, Uttrakhand, India
Postal Code	247667
Time Zone	IST (UTC +0530)

Background

I am a sophomore at the Indian Institute of Technology Roorkee, pursuing my bachelor's degree in Computer Science and Engineering. I was introduced to programming at a very early stage of my life. Since then, I've been trying out new technologies by taking up various projects and participating in contests. I am passionate about system software development and competitive programming, and I also actively contribute to open-source projects. At college, I joined the Mobile Development Group (MDG), IIT Roorkee - a student group that fosters mobile development within the campus. I have been an active part of the Git community since February of this year, contributing to git-for-windows.

Dev-Env

I am fluent in C/C++, Java and Shell Scripting, otherwise, I can also program in Python, JavaScript. I use both Ubuntu 18.04 and Windows 10 x64 on my laptop. I prefer Linux for development unless the work is specific to Windows.

VCS : git
Editor: VS Code with gdb integrated

Contributions to Open Source

My contributions to open source have helped me gain experience in understanding the flow of any pre-written code at a rapid pace and enabled me to edit/add new features.

List of Contributions at Git:

Repo	Status	Title
git	Will merge in master	Micro : Use helper Functions in test script
git-for-windows	Merged and released	#2077 : [FIX] git-archive error, gzip -cn : command not found
build-extra	Merged and released	#235 : installer: Fix version of installer and installed file

The Project

Improve consistency of sequencer commands

Overview

Since when it was created in 2005, the `git rebase` command has been implemented using shell scripts that are calling other git commands. Commands like `git format-patch` to create a patch series for some commits, and then `git am` to apply the patch series on top of a different commit in case of regular rebase and the interactive rebase calls `git cherry-pick` repeatedly for the same.

Neither of these approaches has been very efficient though, and the main reason behind that is that repeatedly calling a git command has a significant overhead. Even the regular git rebase would do that as `git am` had been implemented by launching `git apply` on each of the patches.

The overhead is especially big on Windows where creating a new process is quite slow, but even on other Operating Systems it requires setting up everything from scratch, then reading the index from disk, and then, after performing some changes, writing the index back to the disk.

Stephan Beyer <s-beyer@gmx.net> tried to introduce git-sequencer as his GSoC 2008 project which executed a sequence of git instructions to <HEAD> or

<branch> and the sequence was given by a <file> or through stdin. The git-sequencer wants to become the common backend for git-am, git-rebase and other git commands, so as to improve performance, since then it eliminated the need to spawn a new process.

Unfortunately, most of the code did not get merged during the SoC period but he continued his contributions to the project along with Christian Couder <chriscool@tuxfamily.org> and then mentor Daniel Barkalow <barkalow@iabervon.org>.

The project was continued by Ramkumar Ramachandra <artagnon@gmail.com> in 2011, extending its domain to git-cherry-pick. The sequencer code got merged and it was now possible to "continue" and "abort" when cherry-picking or reverting many commits.

A patch series by Christian Couder <chriscool@tuxfamily.org> was merged in 2016 to the master branch that makes git am call git apply's internal functions without spawning the latter as a separate process. So the regular rebase will be significantly faster especially on Windows and for big repositories in the next Git feature release.

Despite the success (of GSoC '11), Dscho had to improve a lot of things to make it possible to reuse the sequencer in the interactive rebase making it faster. His work can be found here.

The learnings from all those works will serve as a huge headstart this year for me.

As of now, there are still some inconsistencies among these commands, e.g., there is no --skip flag in git-cherry-pick while one exists for git-rebase. This project aims to remove inconsistencies in how the command line options are handled.

Points to work on:

1. Suggest relevant flags for operations that have such a concept like git cherry-pick --skip
2. Unify the suggestive messages of git (cherry-pick|rebase-i) with git (am|rebase)
3. Implement flags that am-based rebases support, but not interactive, in interactive rebases, e.g.:
    - --ignore-whitespace
    - --committer-date-is-author-date or --ignore-date
    - --whitespace=...
    - -C
4. Test and Documentation
5. [Bonus] Make a flag to allow rebase to rewrite commit messages that refer to older commits that were also rebased
6. [Bonus] Performance run on different backends of rebasing, if everything agrees, deprecate am-based rebases

"The Plan"

1. Start by introducing `git cherry-pick --skip`, this will help in step 2 of "the plan" since we are required to unify the advice messages that show during an interrupt of `git (cherry-pick|rebase -i)` when the incoming commit has become "empty" (no change between commits).

   Files Changed:
   ➔ revert.c: Introduce option `--skip` under `run_sequencer()`

2. There are two backends available for rebasing/cherry-picking, *viz*, the `am` and the `interactive`. Naturally, there shall be some features that are implemented in one but not in the other. One such quality is suggestive messages. The am-based rebases (and am itself) will give advice to the user to use `git rebase --skip` (or `git am --skip`) when a patch isn't needed. In contrast, interactive-based and cherry-pick will suggest the user `git reset`. Change this to match the message of `am` backend, so that everything appears symmetric.

   Files Changed:
   ➔ rebase.c: change flow so that `--skip` calls `git reset` while interactive rebasing
   ➔ commit.c: change messages
   ➔ sequencer.c: change suggestive message under `create_seq_dir()`

3. Now that I'm familiar with the code, I'll start picking the pace now. And start implementing the meat of the project. The flags. I'll start implementing the flags in the following order as Elijah suggested:
   1. --ignore-whitespace
   2. --committer-date-is-author-date
   3. --ignore-date
   4. --whitespace=...

   Files Changed:
   ➔ rebase: introduce the flag
   ➔ builtin/rebase--interactive: introduce the flag under `cmd_rebase__interactive()`
   ➔ change messages wherever required

4. Testing and Documentation will go in sync with implementation. I intend to follow `Test Driven Development` but let's see how it turns out.

5. `[Bonus]` As familiarity with the code increases, I might be able to implement a flag to allow rebase to rewrite commit messages that refer to older commits that were also rebased in time.

6. `[Bonus]` If everything goes well and time permits, discuss with the mentor(s)the possibility of deprecating the am backend of rebase. This point is the last to work on as it provides no "cosmetic" difference on the user side. Elijah mentioned the possibility of a "social" problem that might occur which shall be discussed then.

Proposed Timeline

- Community Bonding (May 6th - May 26th):
  - Introduction to community
  - Get familiar with the dataflow
  - Study and understand the workflow and implementation of the project in detail

- Phase 1 (May 27th - June 23rd):
  - Start with implementing `git cherry-pick --skip`
  - Write new tests for the just introduced flag(s)
  - Analyse the requirements and differences of am-based and other rebases flags

- Phase 2 (June 24th - July 21st):
  - Introduce flags of am-based rebases to other kinds.
  - Add tests for the same.

- Phase 3 (July 22th - August 19th):
  - Act on [Bonus] features
  - Documentation
  - Clean up tasks

Relevant Work

Dscho and I had a talk on how a non-am backend should implement `git rebase --whitespace=fix`, which he warned may become a large project (as it turns out it is a sub-task in one of the proposed ideas), we were trying to integrate this on git-for-windows first.

Keeping warning in mind, I discussed this project with Rafael and he suggested (with a little bit uncertainty in mind) that I should work on implementing a git-diff flag that generates a patch that when applied, will remove whitespace errors which I am currently working on.

Availability

My vacations start on 7 May and end on 15 July. The official GSoC period is from 6 May to 19 August. I can easily devote 40-45 hours a week until my college reopens and 35-40 hours per week after that. I'm also free on the weekends and I intend to complete most of the work before my college reopens.

Other than this project, I have no commitments/vacations planned for the summer. I shall keep my status posted to all the community members on a weekly basis and maintain transparency in the project.

After GSoC

Even after the Google Summer of Code, I plan on continuing my contributions to this organization, by adding to my project and working on open issues or feature requests. With the community growing continuously, I feel responsible for all the projects I'm a part of. Having picked up a lot of developing skills, my major focus would be to develop mentorship skills so that I can give back this community by helping other people navigate around and reviewing Contributions.