Convert “git stash” to builtin

Paul-Sebastian Ungureanu
Politehnica University of Bucharest
ungureanupaulsebastian@gmail.com
[CENSORED]

Name and Contact Information

Hello! My name is Paul-Sebastian Ungureanu. I am currently a first year Computer Science & Engineering student at Politehnica University of Bucharest, Romania.

My email address is ungureanupaulsebastian@gmail.com and my phone number is [CENSORED]. You can also find me on #git IRC channel as ungps.

FLOSS manual recommends students to include in their proposal their postal address and mention a relative as a emergency contact. My permanent address is [CENSORED]. In case of an emergency, you may contact my brother, [CENSORED] by email at [CENSORED] or by phone at [CENSORED].

Synopsis

Currently, many components of Git are still in the form of Shell or Perl scripts. This choice makes sense especially when considering how faster new features can be implemented in Shell and Perl scripts, rather than writing them in C. However, in production, where Git runs daily on millions of computers with distinct configurations (i.e. different operating systems) some problems appear (i.e. POSIX-to-Windows path conversion issues).

The idea of this project is to take “git-stash.sh” and reimplement it in C. Apart from fixing compatibility issues and increasing the performance by going one step closer to native code, I believe this is an excellent excuse to fix long-standing bugs or implement new minor features.

Benefits to community

The essential benefit brought by rewriting Git commands is the increased compatibility with a large number computers with distinct configuration. I believe that this project can have a positive impact on a large mass of developers around the world. By rewriting the code behind some popular commands and making them “built-in”, developers will have a better overall experience when using Git and get to focus on what really matters to them.

As a side effect, there will be a number of other improvements: increased performance, ability to rethink the design of some code that suffered from patching along the time, have the chance to create new features and fix old bugs.

To be specific, in the past, Windows users have reported ‘git stash’ as being very slow, taking several seconds to produce output, while all other commands being fast. Example of these reports:

https://github.com/git-for-windows/git/issues/1235
https://groups.google.com/forum/#!topic/git-users/KJAFNKUQ4ic

https://stackoverflow.com/questions/44159850/git-stash-on-windows-extremly-slow-compared-to-libgit2
https://stackoverflow.com/questions/37529016/git-stash-is-slow-on-windows

In theory, switching from Bash or Shell scripts, Git will be one step closer to native code which should have a positive impact on the performance. Being able to start from a clean slate is a great opportunity to rethink old designs that may have been patched a lot during their lifetime. This way, with the help of my mentors, I can think of new ways to try and remove some limitations of the current system (if there are any).

Moreover, I believe that the community will benefit greatly from new features and bug fixes that I could help with. Even though this is not really one of the main goals of this project, I believe that it would be easier to fix bugs or implement new features while rewriting the code. However, I will have to discuss with my mentors and carefully review issues as I would not want to divagate from the purpose of the project.

As a last point, I believe it is good to have a more homogenous codebase, where the majority of the code would be written in C. This could increase the number of contributions to the project as there are maybe more programmers who are familiar with C, and not so much with Perl or shell scripting.

Deliverables

Deliverable of this project is “git stash” completely rewritten in portable C code. Along with the new code, there will be some additions and changes brought to tests to cover any new behaviour.

Related work

Looking over the list of the other proposed projects, I believe that “Convert interactive rebase to C” and “git bisect improvements” are the most alike with this project and may be stretch goal of this project.

Moreover, there is a chance that other scripts could benefit from this project if it were to be taken as an example for future conversions.

Biographical information

I am a freshman at Politehnica University of Bucharest, which is considered to be one of the most prestigious universities in the Eastern Europe. I consider myself an ambitious software engineer that enjoys competition. This has been proven by my participation to programming competitions and extracurricular projects. As much as I like competitions, I also love working with and meeting people that share my interests for programming and technology.

Even though, in the last two years I found myself to be more interested in Android software development rather than competitive programming, I still take part in most of the competitions, such as contests organized by Google HashCode, Google KickStart and ACM.

I have a good grasp of programming languages such as C and C++ and I consider both of them to be my favorites. In the past, I intensively used shell scripting and Git for some of my personal projects.

One of the facts that motivates me the most is that I get the chance to improve my abilities by getting the code reviewed by professionals that have been in the industry for a long period of time. This way, I get first-hand experience regarding software management and increase my programming skills in a professional environment.

Ultimately, I believe that being part of a community such as Git would be a great confidence boost for me knowing that I gave something back to an excellent piece of software that I am using on a daily basis.

Project Details

Plan for converting each command

1. Understand the command well: read documentation, discover any features I were not aware of, find out how it works and why it works that way.
2. Find out all edge and corner cases: these should be quite obvious as soon as I have a good understanding of how the command works.
3. Find out the history of the command and what future improvements can be made: history of the command will help me know what issues affected the command, how those issues were fixed and will help me avoid them in the future. As I mentioned before, I believe that reimplementing those commands is a great excuse to fix bugs and if there are any, I need to know about them.
4. Write tests to cover all possible cases. My process of development is going to be similar to Test-Driven Development where a good amount of tests is required.
5. Convert function: this step basically makes up the goal of this project. In order to convert a command, all the functions which are used by the command must be converted first. The conversion will start with the bottom-level functions, which do not have any dependencies. The new C code will be found in stash-helper.c and will be used by git-stash.sh until the full conversion is complete. As soon as the entire conversion is done, stash-helper.c will be promoted to stash.c. Any functionality that will be implemented, but is not strictly related to git stash will reside in the appropriate files (for example if I had to implement similar to get_oid, which is not related to git stash, but to Git, then I would not implement it

in stash-helper.c; anyway, I do not believe I will encounter this situation that often).
For example, to convert "git stash list", the parser will call “list_stash”, which will call “have_stash”. The conversion of these functions will be made in reverse order they were mentioned (have_stash first and then list_stash).
It is very important to know the Git source well in order to avoid reimplementing functionality. In this case “have_stash()” is somehow already implemented as “get_oid(ref_stash, &obj)”.

6. Ensure that no regression occurred: considering that there are plenty of tests (tests were written at step 4) and that I have a good understanding of the function, this should be a trivial task. To make sure that the code is really leak free, I will use Valgrind, which is already integrated with the testing framework.
7. Write more tests to ensure the best code coverage. This step will be almost non existent due to step 4.
8. In the end, an analysis based on performance can be made. Benchmarking the script will be done by recording the running time given a large set of diversified tests that cover all the functionalities, before and after conversion. The new commands should run faster just because they were written in C, but I expect to see even more improvements. The performance tests will be found in t/perf/ and will make use of the existent testing framework to compare the running time of multiple revisions and repositories.

Potential difficulties during conversion

- The rewritten code may introduce new bugs that may be delaying the project. I believe there is a small chance this will happen as long as I spend some time first to fully understand how it works and why it works that way. Considering the amount of documentation already available and the discussions from the mailing list, this should be a trivial task. Moreover, I am already familiar with “git stash”.
- There may be old bugs, from the Shell scripts that would cause issues while porting because they would probably require a discussion about what implications the bug has and what is the best way to fix it.
- Most of the times, Shell scripts tend to be a lot more smaller and hide a lot of complexity. Moreover, new data structures or utility functions may be needed, some which exist and need changes or do not exist and need to be implemented from scratch.

Wrapping-up the project

In the end, the old Shell script will serve only to call the stash-helper that I would be implementing. The final step is to remove the Shell script and promote the stash-helper to stash builtin.

Example of conversion (for “git stash list”)

To test my capabilities and to be sure that I am able to work on a project of this kind, I tried to convert “git stash list” into a built-in. The result can be found below in text-only version, at the Github link or at mailing list link.

https://github.com/ungps/git/commit/3ffdece881525ee021a0f057a797e0717e101094

https://public-inbox.org/git/20180324182313.13705-1-ungureanupaulsebastian@gmail.com/

```
From 3ffdece881525ee021a0f057a797e0717e101094 Mon Sep 17 00:00:00
2001
From: Paul-Sebastian Ungureanu <ungureanupaulsebastian@gmail.com>
Date: Wed, 21 Mar 2018 20:06:23 +0200
Subject: [PATCH] git-stash: convert git stash list from Shell script
to C
 builtin

Currently, because git stash is not fully converted to C, I
introduced a new helper that will hold the converted commands.
---
 Makefile                 |  1 +
 builtin.h                |  1 +
              builtin/stash--helper.c              |              52
++++++++++++++++++++++++++++++++++++++++++++++++++
 git-stash.sh             |  7 ------
 git.c                    |  1 +
 5 files changed, 56 insertions(+), 6 deletions(-)
 create mode 100644 builtin/stash--helper.c

diff --git a/Makefile b/Makefile
index a1d8775adb4b3..8ca361c57a8eb 100644
--- a/Makefile
+++ b/Makefile
@@ -1020,6 +1020,7 @@ BUILTIN_OBJS += builtin/send-pack.o
 BUILTIN_OBJS += builtin/shortlog.o
 BUILTIN_OBJS += builtin/show-branch.o
 BUILTIN_OBJS += builtin/show-ref.o
+BUILTIN_OBJS += builtin/stash--helper.o
 BUILTIN_OBJS += builtin/stripspace.o
 BUILTIN_OBJS += builtin/submodule--helper.o
```

```
 BUILTIN_OBJS += builtin/symbolic-ref.o
diff --git a/builtin.h b/builtin.h
index 42378f3aa471e..2ddb4bd5c76fa 100644
--- a/builtin.h
+++ b/builtin.h
@@ -220,6 +220,7 @@ extern int cmd_show(int argc, const char **argv,
const char *prefix);
 extern int cmd_show_branch(int argc, const char **argv, const char
*prefix);
  extern int cmd_status(int argc, const char **argv, const char
*prefix);
 extern int cmd_stripspace(int argc, const char **argv, const char
*prefix);
+extern int cmd_stash__helper(int argc, const char **argv, const char
*prefix);
 extern int cmd_submodule__helper(int argc, const char **argv, const
char *prefix);
 extern int cmd_symbolic_ref(int argc, const char **argv, const char
*prefix);
 extern int cmd_tag(int argc, const char **argv, const char *prefix);
diff --git a/builtin/stash--helper.c b/builtin/stash--helper.c
new file mode 100644
index 0000000000000..61fd5390d7d61
--- /dev/null
+++ b/builtin/stash--helper.c
@@ -0,0 +1,52 @@
+#include "builtin.h"
+#include "cache.h"
+#include "parse-options.h"
+#include "argv-array.h"
+
+enum {
+    LIST_STASH = 1
+};
+
+static const char * ref_stash = "refs/stash";
+
+static const char * const git_stash__helper_usage[] = {
+    N_("git stash--helper --list [<options>]"),
+    NULL
+};
+
```

```
+static int list_stash(int argc, const char **argv, const char
*prefix)
+{
+	struct object_id obj;
+	struct argv_array args = ARGV_ARRAY_INIT;
+
+	if (get_oid(ref_stash, &obj))
+		return 0;
+
+	argv_array_pushl(&args, "log", "--format=%gd: %gs", "-g",
"--first-parent", "-m", NULL);
+	argv_array_pushv(&args, argv);
+	argv_array_push(&args, ref_stash);
+	return !!cmd_log(args.argc, args.argv, prefix);
+}
+
+int cmd_stash__helper(int argc, const char **argv, const char
*prefix)
+{
+	int cmdmode = 0;
+
+	struct option options[] = {
+		OPT_CMDMODE(0, "list", &cmdmode,
+			N_("list stash entries"), LIST_STASH),
+		OPT_END()
+	};
+
+	argc = parse_options(argc, argv, prefix, options,
+					git_stash__helper_usage,
PARSE_OPT_KEEP_UNKNOWN);
+
+	if (!cmdmode)
+		usage_with_options(git_stash__helper_usage, options);
+
+	switch (cmdmode) {
+		case LIST_STASH:
+			return list_stash(argc, argv, prefix);
+	}
+	return 0;
+}
diff --git a/git-stash.sh b/git-stash.sh
index fc8f8ae6401dd..a5b9f5fb60bba 100755
--- a/git-stash.sh
```

```
+++ b/git-stash.sh
@@ -380,11 +380,6 @@ have_stash () {
      git rev-parse --verify --quiet $ref_stash >/dev/null
 }

-list_stash () {
-     have_stash || return 0
-     git   log   --format="%gd:   %gs"   -g   --first-parent   -m   "$@"
$ref_stash --
-}
-
 show_stash () {
      ALLOW_UNKNOWN_FLAGS=t
      assert_stash_like "$@"
@@ -695,7 +690,7 @@ test -n "$seen_non_option" || set "push" "$@"
 case "$1" in
 list)
      shift
-     list_stash "$@"
+     git stash--helper --list "$@"
      ;;
 show)
      shift
diff --git a/git.c b/git.c
index 96cd734f12821..6fd2ccd9a81bc 100644
--- a/git.c
+++ b/git.c
@@ -466,6 +466,7 @@ static struct cmd_struct commands[] = {
      { "show-branch", cmd_show_branch, RUN_SETUP },
      { "show-ref", cmd_show_ref, RUN_SETUP },
      { "stage", cmd_add, RUN_SETUP | NEED_WORK_TREE },
+     { "stash--helper", cmd_stash__helper, RUN_SETUP },
      { "status", cmd_status, RUN_SETUP | NEED_WORK_TREE },
      { "stripspace", cmd_stripspace },
      {   "submodule--helper",   cmd_submodule__helper,   RUN_SETUP   |
SUPPORT_SUPER_PREFIX},
```

Useful resources

There has been a lot of progress made in this direction already and I believe it will serve of great help. However, from my understanding it is not yet mergeable and it requires changes.

The patches about converting ‘git stash’ are a great starting point and I will definitely use them. Each time I start converting a new command I will first take a look at what other patches there are, evaluate them and if I consider the patch to be good enough I will continue my work on top of that patch. Otherwise, I will start from scratch and that patch will only serve for comparison.

https://public-inbox.org/git/20170608005535.13080-1-joel@teichroeb.net/T/#m8849c7ce0ad8516cc206dd6910b79591bf9b3acd

One other resource that is of great importance is git itself. I can learn how a builtin is structured and how it is built by considering examples the other Git commands. Having a similar coding standard used, the codebase will be homogeneous and hopefully easier to maintain.

Another important resource are commands that are Google Summer of Code projects from previous years (2016 and 2017) which had as purpose to convert “git bisect” and “git submodule”. I can always take a look at the patches they submitted and read their code reviews to avoid making same mistakes they did.

Project Schedule

I can spend about 25 hours during the exam period (last half of May and the first half of June) and about 40 hours a week as soon as I am done with exams. I hope that by the start of the program (14th of May), I will have completed a good part of the project and that will compensate for the exam period.

After Google Summer of Code ends, I will do my best to keep contributing to Git. Hopefully, all the work I proposed over the summer will be done and I will be able to move towards other areas, maybe by continuing converting other scripts to builtins or something else, depending on what are the priorities of the Git team.

Google Summer of Code starts on 14th of May and lasts 13 weeks. I propose to convert one command per week. I actually do not expect each command to take exactly one week, some may be easier to convert and other may be more challenging, but I believe that the average time of conversion is around 1 week.

I am expecting to submit a patch for every command that is converted or for a group of similar commands, depending on the difficulty of the code and what mentors prefer.. This way, I have well set milestones and results will be seen as soon as possible. Moreover, seeing my contributions getting merged will be a huge confidence boost to myself.

There will be a weekly report to inform about the status of the project, progress made in the past week and future plans. I am not sure whether this should be a weekly email send to the mailing list or a blog.

Each “convert” phase of the project below is not only about converting from Shell to C, but also ensuring that everything is documented, there are enough tests and there are no regressions. Also, the command order presented in the schedule below may suffer modifications.

14th May - 20th May - Convert "git stash list"
21st May - 27th May - Convert "git stash show"
28th May - 3rd June - Convert "git stash save"
4th June - 10th June - Convert "git stash push"
11th June - 17th June* - Convert "git stash apply"
18th June - 24th June - Convert "git stash clear"
25th June - 1st July - Convert "git stash create"
2nd July - 8th July - Convert "git stash store"
9th July - 15th July* - Convert "git stash drop"
16th July - 22nd July - Convert "git stash pop"
23rd July - 29th July - Convert "git stash branch"
30th July - 5th August - Remove old Shell script.
6th August - 12th August* - Wrap-up the project by writing more tests and documentation.
12th August - forever - Pick up another project.

* 1st, 2nd and 3rd mentor and student evaluation

Git contributions

My first contribution to Git was to help making *“git tag --contains <id>” less chatty if <id> is invalid*. Looking over the list of available microprojects, there were a couple of microprojects which got my attention, but I picked this up because it seemed to be a long-standing bug (I noticed it was approached by students in 2016, 2017 and now in 2018). Thanks to the code reviews from the mailing list, after a few iterations I succeeded in coming up with a patch that not only fixed the mentioned problem, but also a few others that were having the same cause.

It is now in the ‘next’ branch of the Git repository.

https://github.com/git/git/commit/3bb1dcd506e022ef9929a7c7c2d3970fcc56bd36#diff-c47fded0b0929f6dcd1b707b3c9a6f4b

I worked on only one project because I tried to achieve the best quality I could. Even after I submitted one patch that was good enough [8], I decided to try another approach that worked better in the end [9].

First of all, I weighed the ideas mentioned at [10] and considered the second one to be the best. After a few iterations [1], [2], [3], [4], [5], [6] and after Junio's review I decided to try another approach and submitted another patch [7].

I changed the approach, because this issue was affecting not only "git tag" command, but also "git for-each-ref", "git branch" and many more. The new submission was based on Junio's ideea and fixed the cause at a lower level than the previous patch.

In order to make sure that the new code was 100% correct and familiarize myself with the testing system, I also write a new set of tests.

[1]
https://public-inbox.org/git/20180219212130.4217-1-ungureanupaulsebastian@gmail.com/

[2]
https://public-inbox.org/git/20180223162557.31477-1-ungureanupaulsebastian@gmail.com/

[3]
https://public-inbox.org/git/20180303210938.32474-1-ungureanupaulsebastian@gmail.com/

[4]
https://public-inbox.org/git/20180306193116.23876-1-ungureanupaulsebastian@gmail.com/

[5]
https://public-inbox.org/git/20180319185436.14309-1-ungureanupaulsebastian@gmail.com/

[6]
https://public-inbox.org/git/20180320175005.18759-1-ungureanupaulsebastian@gmail.com/

[7]
https://public-inbox.org/git/20180322184351.2392-1-ungureanupaulsebastian@gmail.com/

[8]
https://public-inbox.org/git/xmqqpo4ne8ud.fsf@gitster-ct.c.googlers.com/

[9]
https://public-inbox.org/git/xmqqefkm6s06.fsf@gitster-ct.c.googlers.com/

[10]
https://public-inbox.org/git/20160118215433.GB24136@sigill.intra.peff.net/