Application for Google Summer of Code, 2008

Sverre Rabbelier

March 25, 2008

Abstract

Git, 'the stupid content tracker'[1], is actually quite intelligent. It contains a lot of information about whatever content it is tracking, and using the right tools this information can be extracted and presented to the user. Statistics on the tracked content can be very useful to the user. Currently, git lacks almost all forms of statistic gathering.

1 Introduction

The purpose of this document is to describe in detail my proposal for this years Google Summer of Code[2]. First I will describe my project goals, giving an overview of what I would like to contribute to git. This description is suitable to be used as a idea description. Second I will list what functionality will be provided after the project is completed. Thereafter a description of the used interfaces will follow. As fourth a quick description of a proposed implementation plan followed by milestones in the implementation of this project. Lastly I will give some information about myself.

2 Project Goals

Consider Ohloh[3], an external tool that provides commit information about contributors to a project. It provides with a quick overview of all contributors to a project, and what their contribution has been so far. At the moment git does not have anything similar, even though all the data needed for such an analysis is present. Integration with gitk and git-web would allow the data to be presented in a clear and informative way.

Interesting information would be 'who is maintaining this code?' and 'how well are they doing in maintaining it'. Such information is especially useful when trying to decide whom to send a copy of a patch, or whom to ask to fix a particular bug. In a more broad sense it might be interesting to determine what part of the code is most actively worked on, and what part of the code is most stable. Information on how actively part of the code is edited could be used to find 'edit wars'. (In which a part of the code is changed over and over again, this could indicate that that particular part of the code is hard/difficult/error prone.)

Even more generally, code could be analyzed to detect 'bugfixes' based on what they edit, and how often that code is edited thereafter. After completing the analysis all commits are marked as 'introducing a bug', 'enhancement' or 'bugfix' on a per-file basis. Based on such information it is possible to gather statistics about users and their performance. Using these statistics steps can be taken to remedy whatever is causing a contributor to introduce many bugs, or to compliment a maintainer that fixes many.

[1] `http://www.kernel.org/pub/software/scm/git/docs/`
[2] `http://code.google.com/soc/2008`
[3] `http://www.ohloh.net/`

My plan for this summer is to create a 'statistics' feature for git.

3 Provided functionality

Upon completion of this proposed project git would be extended with the following functionality.

- Display statistics on how many changes a user has made, for example in the form of 'lines changed' or a 'total diff'.
- Show which contributors have contributed to the part of the code that a patch modifies.
- Find out what part of the code a maintainer is working on the most.
- Analyse what type a commit is ('introducing a bug', 'enhancement' or 'bugfix') and perform analysis with this data.

4 Interfaces

To implement these features heavy usage can be made of existing git commands. For example, getting the total amount of commits from a maintainer can be achieved with the less-than-intuitive 'git shortlog -s -n -c master', providing an alias to this command would make it easier to use this functionality. Since other git commands will be used a lot, performance may suffer as a result of piping/parsing results from one command to another. When a feature is converted to C later on, attention could be given to directly passing the result from one function to another.

Most, if not all, of these features can be implemented without changing any of the existing git code. Gathering the statistics will mostly consist of consulting other git commands and aggregating the information retrieved. It might be needed to speed up user response time by pre-calculating statistics in the background. This might require creating hooks that will trigger an incremental update of the statistics whenever the user commits/fetches. These hooks exist already in git, and as such require no changes to the existing code.

5 Implementation

Implementation would probably start out with scripts since those are easy to modify and combine with other scripts. As milestones are reached in time, or ahead of time, attention could be shifted to converting these to C and combining them with the rest of git. When the other milestones are finished time could be spent on using the newly added features in gitk and/or git-web.

Determining which users have been active on a file git's built in 'blame' functionality can be used. Git blame is very fast it would be no problem to

make extensive use of it in determining maintainer focus. In a similar way it can be used to determine who has worked on a file recently.

A algorithm will have to be devised that is capable of tracking a specific line throughout commits. This algorithm should work even if lines are added or deleted in the file. The algorithm should be able to detect changes to the line, keeping track of it despite this change. Such an algorithm could make use of 'git diff', which lists inserted and deleted lines with a '+' and a '-'. An example of such an implementation are side-by-side diff viewers, these associate two lines from separate revisions. Since several of these viewers are open source, the algorithm in those viewers can be used as a starting point.

For a more detailed description on implementation please see my website[4].

6 Milestones

During implementation the following milestones will be completed. It would seem likely that they will be completed in the order listed considering how that most milestones are dependent on those listed above it.

- Creating a simple statistics command with basic functionality (such as lines changed, editors in a file, in other words, data that can be aggregated easily).
- Build on these commands, providing more sophisticated features such as 'maintainers that contributed code that was edited in a specific changeset'.
- Develop an algorithm that tracks a line throughout commits.
- Use this algorithm to track down how actively a specific part of the code is edited, and by whom.
- Provide a way for 'git-format-patch' to use this information to decide who to cc: a patch to.
- Develop an algorithm that figures out what type a specific commit is ('introducing a bug', 'enhancement', 'bugfix').
- Parameterize the above algorithm to allow the user to specify ways to detect what type a commit is.
- Develop an algorithm that gathers statistics based on the data mined with the above algorithms.
- Should it be needed to maintain fast user-response: perform analysis in the background (for example, every time a commit is made) and retrieve this data later.

[4] `http://alturin.googlepages.com/gsoc2008`

7 About me

I am a Dutch student, doing my Bachelor at 'Delft University of Technology'. I study 'Technische Informatica', Dutch for 'Computer Science'. Even before starting fourth grade in High school I learned C++ so that I could help out as a coder on a MUD (Multi User Dungeon). In grade four through six I followed the optional "Informatica" (a High school version of 'Computer Science') course. We learned Java and SQL, nothing too difficult, but it got me wanting to learn more. I learned to learn other languages on my own, probably the most valuable thing I learned.

My main project lately has been 'Unsigned Byte', dubbed as such for unclear reasons. As said above I learned C++ so that I could help out as a coder on a MUD. Several years later, I decided that hacking on someone else's source code just wouldn't do. Inspired by a group of active writers and designers I set out to write my own code base. Making use of Sqlite and a Sockets library allowed me to get to coding on the more interesting right away. I started out tracking it with Subversion but soon found this to be unsatisfactory. When a friend advised me to try out git I decided to switch VCS soon after. Progress can be followed online [5]. Within the project we primarily used a forum to communicate, as well as IM.

I have used git on a few other projects so far, although some of its more elaborate features I am not yet familiar with. I have looked through the initial git source (from the initial commit), but aside from that have not worked on git itself. My motivation for this particular idea I have described above. Enjoying working with git made me want to work on it as my Google of Summer project. Knowing that an original idea has more chance of being selected I spent a lot of time looking for ways to improve git worth a GSoC of coding.

In order to get more feedback on my proposal I mailed the git mailing list [6] and asked around on #gsoc. I've spoken with several people, mostly Shawn O. Pearce, to refine my application. I'm really looking forward to coding for git and I think GSoC would be an awesome introduction to it's code base but also to contributing to a large project.

[5] `http://repo.or.cz/w/UnsignedByte.git`
[6] `http://kerneltrap.org/mailarchive/git/2008/3/21/1216854`