Git Helper : Git Error Solvers for Beginners

Yihuan Dong
North Carolina State University
ydong2@ncsu.edu

Niki Gitinabard
North Carolina State University
ngitina@ncsu.edu

Linting Xue
North Carolina State University
lxue3@ncsu.edu

Rui Zhi
North Carolina State University
rzhi@ncsu.edu

ABSTRACT

Git is one of the most popular source code management systems for software development. It has many effective features like branches that enable different people to work on the same code easily. However, recent research and our survey show that it is difficult for people to learn Git due to its steep learning curve. Besides, when novice users face errors, they don't know how to solve it efficiently and properly. Hence many users just keep their usage of Git limited to some simple tasks. In this paper, we proposed three tools for helping git users to solve git errors. The first one was a search engine, the second one was an auto-reply email, and the third one was direct help in the shell. We conducted an experiment on 56 participants to evaluate and compare our tools and resulted that the search engine was the best.

Keywords

Git, Novice User, Usability, Version Control System, Search Engine, Auto-reply E-mail System, Decision Tree

1. INTRODUCTION

Version control systems are widely used to keep track of changes in software projects. Using their change-logs, users always have a backup of everything they have done. If there is something wrong at a stage, they can always roll back to any previous stable version. Or if they want to check something in previous versions, they can just go back and take a look.

Git is one of the most popular version control systems. GitHub, a web-based service that hosts software development projects that use Git for version control, has over 3 million users and over 7 million repositories. Git has a good support for distributed work and non-linear workflows. As a result, it is helpful for people who are working in teams. The non-linear workflows' support and branch facilitate several people to change the same line of code simultaneously without losing any of their work. After they finish, they can decide how to merge different features together. They can always check and see who is responsible for each line of the code, so if something isn't working or someone doesn't know how it works, they know whom to ask.

However, Git is not novice-friendly according to our survey and literature review. Beginners need to spend a lot of time learning Git before benefiting from it. The steep learning curve for Git scares away many novice users. For many beginners, they only use the simplest commands in Git and haven't experienced other features of this powerful version control system. Several studies show the problems novice users may face and the steep learning curve of Git[2][10][6]. Isomöttönen et al. did research on finding the challenges and problems students face when learning Git[5]. The survey result of their study shows that many students have difficulties dealing with conflicts when they work on the same part of a software. Cochez et al.[1] analyzed student's Git commit log data in several computing courses. The result shows that students often write trivial or nonsense commit messages. The main reason for the steep learning curve of Git is its complex conceptual model[8]. Jansson et al.[6] found that Git's staging functionality is quite confusing for beginners.

We also conducted a user survey to further identify user problems with Git. The survey[4] contains a total number of 14 questions, with 2 questions asking about user proficiency with Git, 1 question asking about user's purpose to learn Git, 11 questions to identify the problems users were facing with Git. Our survey shows that Git is apparently not easy to pick up for beginners. The main reason is Git's steep learning curve. Novice users cant get familiar with Git concepts in a short time. Without knowing the mechanism behind Git features, it's rather difficult for users to use it appropriately, especially when performing complicated tasks such as using branches and solving conflicts. A scenario might be the case that when they face conflicts, they have no idea what the conflict is and what causes the conflict, which leaves the user in a bad situation. Our survey also confirmed this fact.

In order to reduce the difficulty of using Git and help novice users use Git efficiently, we proposed three tools to

achieve this goal. One is a search engine which is designed specifically for Git errors, another one is an auto-reply email system, the third one is a command line helper expert system based on a decision tree.

In this report, in section one, we present the problems of Git and introduce our three solutions briefly. In section two, we will introduce each of our solutions in detail. In section three, we will talk about how we evaluate our tools and our methods. In section four, we will talk about the results. In section five, we will conclude our work and talk about future work.

2. SOLUTIONS

Based on the problems we found, we proposed three solutions in this study to solve those problems. The first tool is Git Helper search engine. Git Helper search engine is designed specifically for git errors based on Stack Overflow pages. The search engine will return top 20 results. The second tool is Git Helper Auto-reply System. The auto-reply E-mail system provides user the best solution with mail address: git_helper@yahoo.com. The third tool is Command Line Helper. The Command line helper is based on decision tree. It can detect git errors in real time and provide concrete step by step solutions to guide users. In this section, we will introduce the design, features, and implementation of our tools.

2.1 Database

Our searching engine and auto-reply email system share the same database. In order to build our own database, we implemented a web crawler based on Python and google-search package to crawl pages from Stack Overflow. The crawler sends error messages to Google and gets the result links back. When user searches git errors on Google, they may find empty links since Google's server will save a webpage eventhough it cannot be accessed to later. We wrote a parser to filter those empty links. Afterwards, the crawler sends HTTP requests to those links to get corresponding HTML pages. We crawled about 2,000 pages which are related to Git errors from the Stack Overflow website. Each page mainly contains one question, multiple answers, and separate votes for the question and answers. After crawling the raw HTML pages, we implemented a parser to get the question, answers, votes, titles and links from each page. After removing the duplicate pages, we obtain 1,311 pages in total. All the parsed contents are organized and stored in a CSV file. Later, we upload the data into the Google Cloud server.

For the git helper search engine, we create a document class[3] and build indexes for those contents. The Google App Engine provides a powerful search API which can do full text matching on strings for indexing documents.

2.2 Solution 1: Git Helper Search Engine

In this section, we will introduce our Git Helper Search Engine(`http://git-helper-2016.appspot.com`). The Git Helper search engine is designed specifically for Git errors (see Figure 1). Just like Google, users can find solutions for Git errors by searching the error message. The Git Helper search engine will return top 20 relevant results for the error message within two seconds. Each result has a feedback button to receive user's feedback. We record the data automatically and we can improve our search engine later based on those feed-backs. Each time when user clicks on a link, the click data will be recorded in the cloud data store. The main advantages of our search engine are that it's easy to use and users can get reliable results immediately. Users don't need to learn how to use this tool if they have experience on using search engines. The disadvantage is that user may not find relevant results if they don't input appropriate key words. Another problem is that maybe that our database is too small to contain the information user needs.

2.2.1 Design

Before implementing the search engine, we studied the features of several popular search engines including Google, Bing, DuckDuckGo and Baidu. We decided to use a simple style. The whole search engine website has only one input box and a search button. The design follows Fitts' law [7] as we made the input box as wide as possible and user can click on the input box easily. For our result page, we also included a search input box on the top of the page. Besides, our feedback button event is based on AJAX. Thus user can interact with the system without unnecessary page jumping. Besides, our search engine is adaptive to smart phones, user can also get good experience through using our search engine on a smart phone.

2.2.2 Techniques

The back-end of the Git Helper search engine is implemented by Python within the Jinja2 template and powered by Google App Engine. The front-end uses JavaScript, AJAX, Bootstrap and CSS. We implemented a web crawler using Python. All the data of the search engine are crawled from Stack Overflow website.

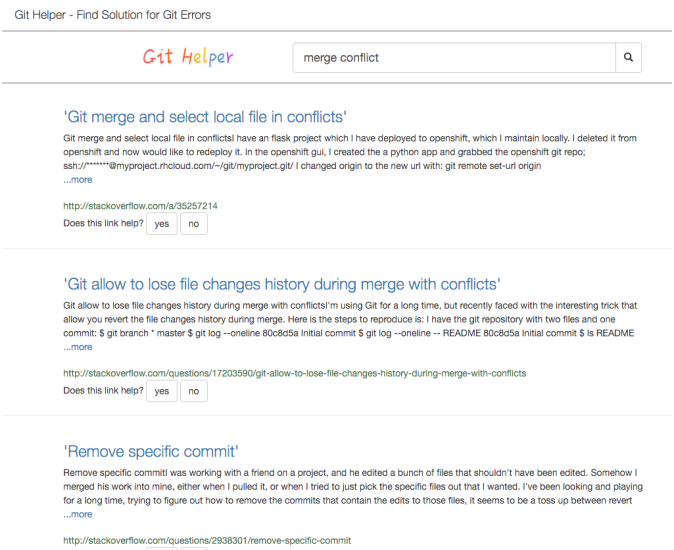

Figure 1: Git Helper Search Engine

2.3 Solution 2: Git Helper Auto-reply System

We have a mail server with address git_helper@yahoo.com listening for emails seeking help. After we receive an email, we tokenize the email content by removing the stop words

and Unicode characters to extract the key error words. Based on these key words, we sorted our database by the Term frequency-inverse document frequency (Tf-idf) values, which will be explained in the next paragraph. Then we respond to request email with the top related content that includes top ranked answer on Stack Overflow as well as corresponding link.

Tf-idf is a statistic method that is used to evaluate how important a word is o a document in a corpus [9]. It's the production of term frequency and inverse document frequency. The ways for determining the exact values of both statistics are various. In our work, we use the Sklearn package in Python to calculate it. Each document contains the web information that includes web title, question and a set of answers. Corpus indicates all 1,311 entries in data bases. After obtain Tf-idf value for each key word, we simply add these values together as the score of corresponding document, then sort all the documents based on Tf-idf scores and return the best one.

This solution might be better than the search tool if we can find the best answer properly, because then the user doesn't need to read the different possible answers and analyze them and find the one that matches their problem. But if the answer we find is not a good match, we lose the chance, because there is no other try. There is a waiting time of at most 20 seconds which most probably will not be annoying for users because first of all, it might not happen, they might get an immediate response. On the other hand email is usually not considered as an immediate solution. So, 20 seconds for getting an email is not usually counted a long time. It might be better than the decision tree, because it is more dynamic and has a wider range of support, though the solutions are not that straight forward and clear.

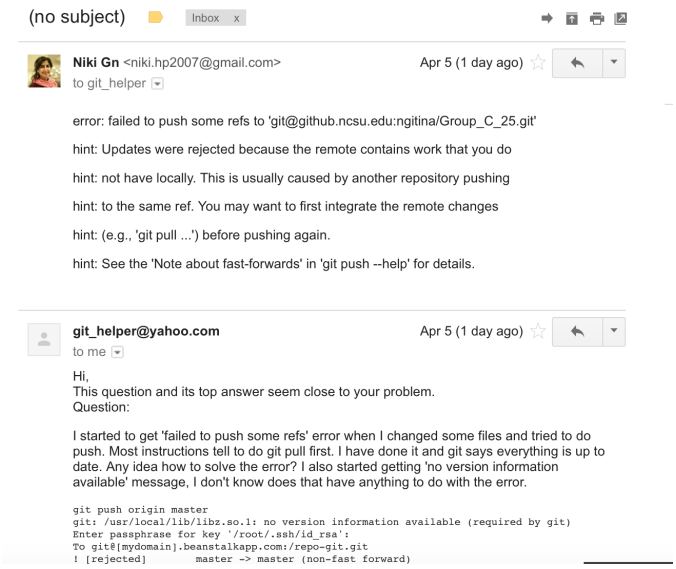

Figure 2: Auto-reply System Example

2.4 Solution 3: Command Line Helper

In this solution, a decision tree is integrated into a customized shell to provide instant solutions to the errors that users might come across when using Git commands. An integrated solution provider has an advantage of being able to present a solution to an error on screen. It will save the effort for users to search the error message online and try to identify a proper solution from a large volume of text that might or might not be helpful. What's worse, this process can be really hard especially for novice users. One other advantage of decision tree is that it can be very easy to add a new solution to a specific problem in the decision tree. One can add a solution to a certain error simply by stating the criteria to determine when this solution applies and what explanation and solution to provide when seeing the corresponding error. For other solution like Search Engine, it is very hard to control if a newly added solution can be seen by users when searching for the problem with different keyword.

Aside from the advantages, there are also downsides of this solution. One obvious problem is that it is very hard to enumerate all the possible errors that a user might see in Git and it takes much of expert time and knowledge to create solutions. However, considering that our goal is to provide support for novice users and the hypothesis that a large portion of the novice users will only see a small portion of all Git errors, which are the most common errors, it is possible to cover these common errors that a novice user might encounter when using Git. Another potential problem for this solution is that one need to be very careful when determining their rules to identify a specific error that their solution is written for. Otherwise, it could mislead the novice user and make the problem more complex. When designing the rules, people needs to be as specific as possible and try to narrow down to solve one problem at a time, instead of trying to generalize the problem.

2.4.1 Shell

To provide an integrated solution to errors, we need to build a customized shell that can take in user command, perform the command, grab error message if an error occurs, analyze the error message and provide the corresponding solution to user in a reasonable format. The shell we built used Python 2.7 and is capable of doing all the things mentioned above in Linux Bash. Figure 3 shows the user interface of our final Command Line Helper.

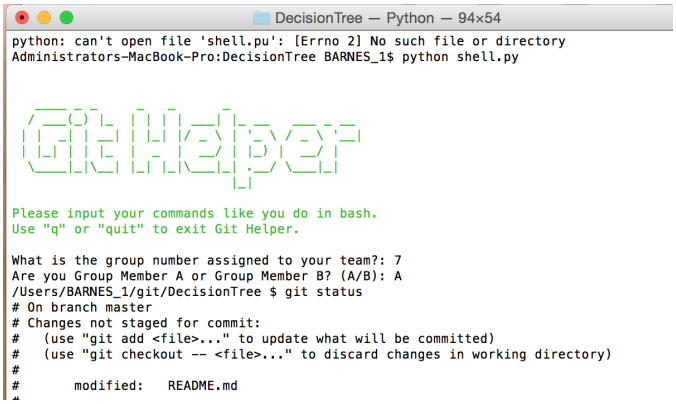

Figure 3: The UI of Command Line Helper

2.4.2 Solution Preparation

To gather error messages, explanations and solutions to build our decision tree, first, we came up with a list of the most basic commands that a novice user needs in a simple group project setting. Then we searched on Stack Overflow website for common error messages and solutions for each of the commands. One of the reasons why we use Stack Overflow to look for common errors is that Stack Overflow is a very well know Q&A platform for computer related problems. The large population of both novice and expert users provides us with many error messages from novices and detailed answers from experts. Another reason why we use Stack Overflow is that it has a voting system for all the questions and answers. This provides us with a relatively reliable way to see which questions are common and which answer is the best. Specifically, the questions that have more votes usually means that more people are having the same issue, thus is more likely to be seen by novice users. Answers that have higher votes typically means it has solved the problem for more people, thus is more likely to contain the most reliable solution and explanation to the problem. After we have a list of most common errors for each command, we extracted a unique combination of keywords from each of the error messages for error identification purpose. We also collected a step by step solution for each of the errors with an explanation of the causes that can be easily understood by novice users.

2.4.3 Decision Tree

With this information mentioned in the last subsection, we are ready to build the decision tree. In our approach, we build our tree into two layers. The first layer is a command extractor. When it gets a command, it will extract the command name from the whole command. Then, it will pass the command information, including command name and error message, to a module called solution provider. The solution provider will use the command name to determine which Error Message Analyzing Module in the second layer to distribute the information to. Upon getting the information passed down from the first layer, the second layer will look for keywords in the error message, identify which solution was written for the error, and return corresponding explanation and solution to the shell to present to the user.

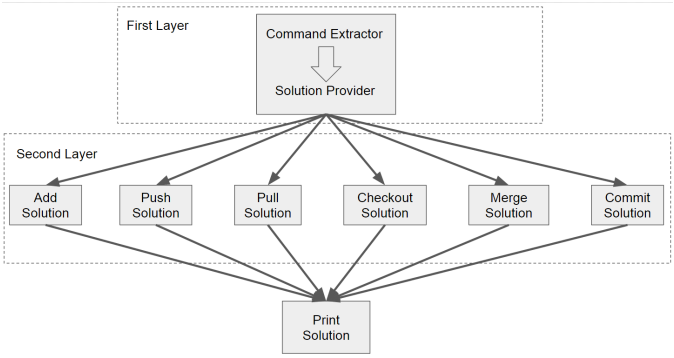

Figure 4: Decision tree structure used in Solution 3 the Command Line Helper

2.4.4 Example

Here is a live example of how the decision tree solution works. When a user use "git pull origin test" command and face with a Conflict error, upon receiving the error message, our GitHelper shell will send both the whole command and error message into the first layer of our decision tree. The first layer will extract the command name "pull" and pass the command name and error message to the solution provider. The solution provider will then pass the message to the "pull command error message analyzing module" in the second layer to analyze the error message. The module will find the keyword "Conflict" in the error message and decide that it is a conflict error. At last, the module will return the explanation, commands needed and step by step instruction to solve the problem back to the shell to display to the user. Figure 5 shows an example of the solution provided by Command Line Helper. From the figure, we can clearly see the three parts of each solution: Explanation, Comm

```
########################################################################
  ___ _ _ _ _
 / __| ___| |_ _| |_(_)___ _ _
 \__ \/ _ \ | || |  _| / _ \ ' \
 |___/\___/_|\_,_|\__|_\___/_||_|

########################################################################
Explanation:

        The remote server has some work that you do not have on your local machine. You can do
 a git pull command to get the work you do not have locally.

Command:

        1. git pull origin <branch-name>

Solution:

        1. Please use git pull origin <branch-name> command to get the work that you don't hav
e locally.


########################################################################
Solution Ended
########################################################################
```

Figure 5: An example of the format for the solution provided by Command Line Helper

2.4.5 Extension

Note that we are fully aware that just by analyzing user command and the error message is not enough to identify a problem. There are cases that an error message produced with the same command that is caused by multiple reasons based on previous actions. It will require additional information like the status of the repository to more accurately determine the cause and corresponding solution to an error. However, considering that our target users are novice users who will only use the most basic commands and produce the most common error, and the scope and difficulty of analyzing repository status and log files, we decide not to put this analysis into our course project and save it for future work. In the future, we could add a third layer to look at the repository status and log files before giving a solution.

3. METHOD AND EVALUATION

3.1 Method

In this section, we will talk about our methods for evaluating our tools. The experiment design will be presented in detail.

3.1.1 Experiment Participants

All of our experiment participants are from CSC 216 course. CSC 216 is an introductory programming course in Java in which they use simple commands of Git to submit their projects and work in teams. Most of the students were novice users of Git commands and were only exposed to Git concepts taking this course. In total, we had 56 participants

in our experiment, of which 14 were grouped using Search Engine solution, 14 were grouped using Auto-reply solution, 14 were using Command Line Helper solution and 14 were grouped using whatever help they can find, as baseline solution.

3.1.2 Pre-survey

We designed a pre-survey which we asked users to fill it one session before the experiment session. We asked about their Git proficiency in that survey. Asked how familiar they are with Git, and also which commands of Git they have used and know how to use. We got 75 responses which we used for grouping purposes.

Table 1: Questions in Pre-Survey

Questions
1.Which of following git commands do you frequently use?
2.How familiar are you with git command line?
3.Do you have version control experience?

3.1.3 Grouping

We put students in 4 different groups which we tried to distribute the same amount of knowledge across them. We decided to use a number of Git commands they know as their level of knowledge and have similar groups for testing our solutions. One of these groups is used as a baseline and the students in that group are free to use any tool they want to solve the errors they face. The other three groups are provided with 3 different shells that will allow them each to use one of our solutions. The shell is used for guiding them to the solution and also for saving logs on their actions so that we can find out how much time they spent on the tasks. After General grouping, due to the fact that it is almost impossible to create an error in a repository by a single person, we decided to organize students into 2 person teams. In this teaming we tried to put people with similar knowledge together so that their help to each other does not affect our results.

3.1.4 Experiment Instruction

For our experiment, we designed instructions of one-page length for participants to follow. We design two different instructions for team members to collaborate. Both of the two instructions have the same "before experiment" reading section. In this section, we tell the students how to use our shell, what to expect in the experiment and how to collaborate within teams. The difference between the two instructions is the step by step instruction part. To create git errors in a clean repository, we asked students to follow the scripts we wrote in the step by step instruction in a certain order. Our instruction has steps which say "Wait until Team Member B to finish step N." For these steps, Team Member A should make sure that Team Member B finished step N before carry on with next step.

To make both of the members of a team have roughly equal amount of work to do and get more useful data from participants, we designed the step by step instruction that each of team member will face with one error, which means each of them need to find solution for one error during the experiment. This setting will allow us to get problem-solving log from all the participants in the experiment, instead of one problem-solving log for each team, which will be half the number of participants.

3.2 Experiment

Our experiment conducted in a large classroom setting where teammates can sit together to work on our experiment. At the beginning of the experiment, the participants were asked to sit with their teammate. After all the participants were in teams, we distributed the two versions of instructions to each team. The participants get to decide which version they want to use within their teams. Then, we went through the "Before Experiment" reading part with the participants to make sure they know to follow the order in the instruction to perform each command and the experiment began. The whole experiment went for 1 hour. We were walking around the room to provide technical help and observe participants behaviors throughout the experiment.

3.3 Data Collection

To get better understanding of user behaviour and analyze our tools, we collected two types of data from participants - user interaction log and user satisfaction survey.

We also have a Google form provided in the emails and "Was this helpful" buttons beside each result shown by our search engine tool to ask users how that specific solution was, but since the users didn't end up using them enough to analyze, we decided to ignore that data.

3.3.1 User Interaction Log

To collect user interaction log during the experiment, we built a clean shell by removing the solution provider part from Command Line Helper solution and added logging functionality to it. We request all of the participants to use our clean shell, instead of Git bash, to run our experiment. When the user starts running the clean shell, it will first ask for team number, user identity in the team (team member A or team member B), and it will create a log file for him with a file-name in the format of "TeamNumber-TeamMember-Solution-log.csv". This will allow us to organize all the log files in the same folder for later analysis. When the user face with an error, the clean shell will give instruction about how to use our tools to find solutions to the error based on which solution is assigned to the user.

In the log file, we record information like the timestamp when user runs a command, the whole command the user used, is the command a git command or bash command, the command name, is the result an error and if it is, the corresponding error messages. With this log file, we will be able to get statistics on how long it took a participant to solve an error, how long it took the participants to complete the experiment and what commands did the participants use during the experiment.

3.3.2 User Satisfaction Survey

Apart for user interaction log, we also required participants to fill out a post survey about their user experience using the tool. In the user survey, we asked questions from a different perspective to evaluate our solutions, like the helpfulness of each of the tools, how relevant is the solution given by our

tool, how easy to find solutions using our tools, etc. We hope to combine the results from the log files and post survey to help us learn the strength and weakness of each tool and make our decision on which tool performs best.

Table 2: Questions in User Satisfaction Survey

Questions
1. How helpful do you think this tool is? _ Very helpful _ Somewhat helpful _ A little helpful _ Not helpful
2. Will you use this tool or recommend this to your friend in the future? _ Yes _ No
3. How easy could you find solution to your error using this tool? _ Very easy _ Kind of easy _ Kind of hard _ Very hard
4. How relevant do you think the suggested solutions were? _ Very relevant _ Somewhat relevant _ A little relevant _ Not relevant
5. How do you think this tool could be better?

4. RESULT

Table 3: Experiment Statistics: Number of participant users, useful log files, available satisfaction surveys.

Solutions	Users	Log Files	Satisfy Survey
Search Engine	14	6	**10**
Email Server	14	4	**13**
Decision Trees	14	5	**12**
Baseline	14	7	10
Total	56	22	45

After the experiment, we collected 22 interaction log files and 35 responses about our solutions from post survey. Table 3 shows the distribution of the log files and post survey collected among four experiment groups.

As the table suggests, the number of log files and post-survey responses are fairly even among different groups. However, there is a noticeable difference between the number of participants assigned to each team, the number of log files collected and the number of post-survey responses we receive. According to our observation during the experiment, there are several reasons that may have caused this difference. First, some students did not show up, even though they said they agree to participate in the experiment in their pre-survey. Second, some participants were having trouble getting the shell running due to multiple unexpected reasons. We tried to fix the known compatibility problem of our shell and test it both on Mac and Windows system before the experiment. However, there still were unpredicted errors from participants that couldn't be fixed on scene. Third, some participants failed to follow the instruction given to produce the errors, or failed to push their log files to remote repositories at the end of the experiment. Thus, we couldn't use/get these log files in our analysis. Last, the reason why we got more post-survey responses than log files is most likely be participants who have the tools running but could not fix their problems. They did not end up submitting their log files, however, decided to kindly fill out the post survey to express their feelings about the tool.

4.1 Interaction Log Files Analysis

Table 4: Tool Efficiency Evaluation: Average time for completing the test and solving git error.

Solutions	Average Spending Time	
	Complete Test	**Solve Error**
Search Engine	$17'54''$	$5'56''$
Email Server	$11'30''$	$2'45''$
Command Line	$\mathbf{11'24''}$	$\mathbf{1'49''}$
Baseline	$17'22''$	$5'42''$

⋆ ‘′’ indicates mins; ‘ ′′’ indicates secs;
⋆ Command line helper is the most efficient one.

In this section, we will talk about our analysis on 22 useful log files to evaluate the efficiency of each solution. We use the time users spent on solving git errors as main criteria to determine efficiency. By useful log files, we mean those logs that contain at least the error that we designed for our experiment and the user actually found a solution. Otherwise, we will not be able to determine the time to solve the error and the time to complete the experiment.

Table 4 shows the average time participants needed to complete the whole test (the "Complete" column) and solve errors ("Solve" column) for each group. In particular, we count the time between participants see the git error and resolve the error as the time they used to solve the error. From the result, we can see that the Command Line Tool is the most efficient, which has both the least "Complete" time and "Solve" time. This result is actually what we had expected, because the Command Line Tool shows the solution immediately after users encounter errors. It does not require additional action to look for solutions to the error. Email Server is the second efficient tool, with the second best "Complete" time and "Solve" time. This is because users only get to the most relevant solution from the server within 20 seconds. They don't need, or have the option to look at more than one solution which would save some time. However, they do need to spend time on sending and receiving email.

In contrast, Search Engine and Baseline groups are relatively slower. For the Search Engine group, participants were presented with many solutions. They had to spend more time to read through at least the first few web pages before making decisions on which one is more helpful. Among the seven people working in the baseline group, 4 of them seek help from his/her friend, 2 of them solved problem with their own knowledge. It may be the case that users who seek help from their friends and solved error by themselves took less time, so the average completing time and solving problem time are shorter than Search Engine Group. It also reflects one weakness of our search engine that we have limited database, so it would take users longer time to find the correct answer.

4.2 Satisfaction Survey Analysis

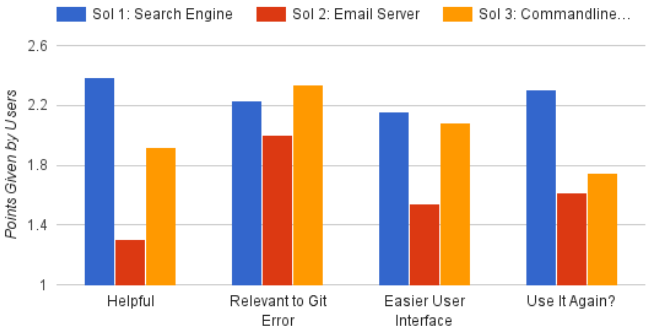

Figure 6: Satisfaction Survey Comparison Between Three Solutions

In this section, we will focus on evaluating three solutions base on 25 satisfaction survey. In the satisfaction survey, for Q1, Q3 and Q4, we assign score from 0 to 3 for each answers. For Q2, we assign 3 to the answer Yes, 0 to the answer No. Then we calculate the average points of each question base on these scores. Figure 6 visualizes comparison of average points between three solution. From figure 6 we can see that the search engine wins the highest point for helpfulness and easiness, but Command Line Helper wins in relevance. In terms of question 5 in user satisfactory survey, we summarize participants response and our discussion as follow:

- Search engine gets the most people who want to use it again, even though it's not the most efficient and relevant one. Because after browsing the websites that search engine return, the users learn about the reasons that causes the conflict and the mechanism of git feature
- Command Line Helper provides the most relevant result. The most important reason why people like Command Line Helper is that it provides user with solution immediately after users encounter errors. However, post survey result shows that users didn't enjoy the interface and somehow the Command Line Helper "hijack" their command lines. Some bash command doesn't work very well under our Command Line Helper shell due to compatibility of the python API and the system environment. This is a known issue to this solution and will take much time to improve in the future.
- Email server gets the worst result. Because users don't like the email user-interface. Instead of waiting for the email response, they would rather search the problem on-line or send the error message in the shell. The email server actually hinders the effectiveness of Stack Overflow. Because it only provides one solution. If the solution is not helpful, then the user can't solve problems. In addition, there are two users that didn't get email response because they didn't use gmail. Although some students stated that it was a good idea to show them that Stack Overflow has the answer to their error but they mostly didn't like the idea of having a middle-man. What's more, our email server have limited database.

4.3 Overall Evaluation

Choosing the best solution was highly dependant on which features we cared most about. People believed that search engine is more helpful and is easier to use than the Command Line Helper (with a really small difference). But they also thought that Command Line Helper gives more relevant results. Looking at the time it took people to solve the problems, we see that the ones using Command Line Helper were much faster than the others.

Considering the fact that our original goal of this project was to help novice user find the solution and solve the error faster, we decide that **Command Line Helper** is our best solution because it has both the shortest "Complete" time and "Solve" time and it provides the most relevant solution to user. However, we need to realize that Command Line Helper has its limitation. If the solution is not in its decision tree, it will not be able to provide a single solution. The best it can do in this situation is to point to other resources, for example, our other solutions.

5. CONCLUSION AND FUTURE WORK

In this work, we proposed three tools for helping git users to solve git errors. We also conducted experiments on 56 participants to evaluate those three tools. Based on our user interaction log file analysis and user satisfaction survey, we conclude that the **Command Line Helper** is the optimal tool. Because it provides the users more relevant solution, even though However, there are limitations to our tools, future work will need to improve each tool from the following perspectives.

- For the database, the crawled data we used for git helper search engine and email system are relatively small. We should work on expanding it to include more data from Stack Overflow as well as other related forums.
- For the email system, firstly, currently, we use a CSV file as our database. Because we only have small dataset now and querying from CSV file doesn't slow down our system. In the future, we should replace it with a NoSQL database such as MongoDB. However, for search engine, we already drop a database for our search engine on Google App Engine. Secondly, the answer returned by email might not be the best-related solution depending on the error message they put. For the future work, we should add the feature that the user can request the next answer if the first one doesn't work. Thirdly, Email solution currently only supports emails sent from Gmail because of formatting differences. Here, we assumed students will use their NCSU email, which was mostly but not completely true. In the future, more formats might be supported.
- For the Command Line Helper solution, we need to add a third layer to analyze repository status and log files in order to provide more accurate solution. And we also need to try to find an easier way for people to collaborate on building the decision tree.

- For experiment setup, we can have a pre-session to setup participants' computers. In our current experiment, a lot of time was spent on installing Python and setting environment variables and many subjects didn't get to the main experiment part.

6. ACKNOWLEDGEMENT

Special thanks to Dr. Heckman for letting us use her class time as our experiment.

We thank Professor Tim Menzies for all the useful suggestions and feed-backs about this project.

7. REFERENCES

[1] M. Cochez, V. Isomöttönen, V. Tirronen, and J. Itkonen. How do computer science students use distributed version control systems? In *Information and Communication Technologies in Education, Research, and Industrial Applications*, pages 210–228. Springer, 2013.

[2] J. Feliciano. *Towards a Collaborative Learning Platform: The Use of GitHub in Computer Science and Software Engineering Courses*. PhD thesis, University of Victoria, 2015.

[3] Google. `https://cloud.google.com/appengine/docs/python/search/`.

[4] G. H. `https://goo.gl/w6OHI0`.

[5] V. Isomöttönen and M. Cochez. Challenges and confusions in learning version control with git. In *Information and Communication Technologies in Education, Research, and Industrial Applications*, pages 178–193. Springer, 2014.

[6] E. Jansson. An investigation of git in an xp project. 2013.

[7] I. S. MacKenzie. Fitts' law as a research and design tool in human-computer interaction. *Human-computer interaction*, 7(1):91–139, 1992.

[8] S. Perez De Rosso and D. Jackson. What's wrong with git?: a conceptual design analysis. In *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*, pages 37–52. ACM, 2013.

[9] Wiki. `https://en.wikipedia.org/wiki/Tf-idf`.

[10] Z. Xu. Using git to manage capstone software projects. In *ICCGI 2012, The Seventh International Multi-Conference on Computing in the Global Information Technology*, pages 159–164, 2012.