Git Basic

Software Requirements Specification

Prepared by Matthew Ristine

January 27, 2018

COMP 495

Contents

[1. Introduction 3](#_Toc504828478)

[1.1. Note to the Professor 3](#_Toc504828479)

[1.2. Overview 3](#_Toc504828480)

[1.3. Purpose 3](#_Toc504828481)

[1.4. Scope 3](#_Toc504828482)

[1.5. Definitions, Acronyms, and Abbreviations 4](#_Toc504828483)

[1.6. Resources 5](#_Toc504828484)

[2. Product Overview 5](#_Toc504828485)

[2.1. Product Perspective 5](#_Toc504828486)

[2.1.1. System Interfaces 5](#_Toc504828487)

[2.1.2. User Interfaces 5](#_Toc504828488)

[2.1.3. Software Interfaces 5](#_Toc504828489)

[2.2. Product Functions 6](#_Toc504828490)

[2.3. User Characteristics 6](#_Toc504828491)

[2.4. Constraints 7](#_Toc504828492)

[3. Product Requirements 7](#_Toc504828493)

[3.1. External Interfaces 7](#_Toc504828494)

[3.1.1. Git SCM 7](#_Toc504828495)

[3.2. Functions 7](#_Toc504828496)

[3.2.1. Installer 7](#_Toc504828497)

[3.2.2. Command Line Interface 7](#_Toc504828498)

[3.2.3. Review and Diffs 7](#_Toc504828499)

[3.2.4. Productivity Assitance 8](#_Toc504828500)

[3.3. System Requirements 8](#_Toc504828501)

[3.3.1. System 8](#_Toc504828502)

[3.3.2. Resources 8](#_Toc504828503)

# 1. Introduction

## 1.1. Note to the Professor

This SRS was adapted from my COMP 325 project. As explained before, I started on the concepts for this software in that class. I have modified and refined this document to better reflect the software application we are going to build.

## 1.2. Overview

This document contains the following:

[Section 1](#_1.__Introduction) – (this section) introduction and context.

[Section 2](#_2._The_Overall) – an overview and description of the product.

[Section 3](#_3._Product_Requirements) – specific product requirements.

## 1.3. Purpose

This document serves two purposes:

1. It defines scope for the software.
2. It lays out the requirements (both system and functional) for the software.

This requirements document should guide the design and development efforts of the team.

This document is to be checked into source control with the rest of the project. It should be modified as necessary and kept up to date. Changes made to it are to be approved by the technical lead (Matthew Ristine). The technical lead is also responsible for determining which features will be developed as part of each milestone. (Note: we will most likely not have time to complete all these features as part of the practicum. Therefore, we will focus on core functionality first. The project plan will detail which features are selected for completion.)

## 1.4. Scope

The software product’s name is *Git Basic*. Its name is reflective of its essence. It is a git client that aims to present to the user a simple graphical and textual interface for git version control.

The following problem statement helps understand the purpose of Git Basic.

Git clients tend to fall to one extreme or the other. On the one hand, Git can be run entirely from the command line console. While this is a powerful solution it has the following drawbacks:

* The user must know all the git commands.
* Lengthy commands (for example, checking out a branch with a long name) are difficult to type out quickly.

On the other hand, Git graphical user interfaces exist which, while they solve the above problems, they present drawbacks of their own:

* They often have cluttered UIs which can make certain options hard to find.
* They often present the user with more feature than he/she wants or needs.

In his book, *The Design of Everyday Things*, design expert Don Norman (2013) speaks about two contrasting types of knowledge that the user is faced with in a product’s design: knowledge in the head and knowledge in the world. The command line versus GUI approaches contrast these two main methods of knowledge. The Git console requires all knowledge to exist in the user’s head. They must know the commands to type. The Git GUI tries to place all knowledge in the world. This is done by creating buttons, menu items, and other graphical controls which signify to the user what operations can be performed.

Git Basic aims to bridge the gap between these two worlds. To do this, it will integrate a console together with a minimal graphical user interface. The command line console integrated into the application will be the main source of user input. This means that Git Basic will provide all the same benefits of running Git straight from the command line. However, in addition to the console, a user interface will provide shortcuts to the most common commands with the aim of speeding up user input. Also, the interface will provide a means for displaying information which is difficult to view in a console – for example, diffs and file status (staged versus unstaged files).

In summary, Git Basic is a Git version control client with a minimal graphical interface providing the user with productivity boosting features.

## 1.5. Definitions, Acronyms, and Abbreviations

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Version Control | Software used for the management and control of files. Version control manages file history allowing one to retrieve the file at any point of its history. A typical use of version control is in the software development process in order to manage changes to a code base by multiple members of a team. |
| Git | Git is a free open-source distributed version control solution. It is one of the most popular version control systems. See <https://git-scm.com/> |
| Repository | This is directory structure under version control – it holds all the files being tracked. Since Git is distributed, you can have a shared remote repository which can be cloned down locally by all the developers using it. |
| Branch, merge | Version history in Git is a directed acyclical graph often simply referred to as a tree. To facilitate simultaneous development on a common code-base, developers branch off of some main trunk (often master) and merge their changes back into the trunk when they are complete.  Branch and merge are both Git commands for these operations. |
| Commit | This Git command is used to check-in changes on the current branch. Commit creates a new node in the version tree. |
| HEAD | This is a reference to the currently checked-out node (commit). |
| Push | This Git command sends all of the changes (commits) of the current branch to the shared remote repository. |
| Pull | This Git command retrieves all of the changes (commits) of the current branch from the shared remote repository. |

## 1.6. Resources

To download Git SCM and for more information about Git, go to: <https://git-scm.com/>

Refer to <https://git-scm.com/docs/git> for Git documentation.

For a 15 minute interactive Git tutorial go to: <https://try.github.io/levels/1/challenges/1>

The following Git tutorial from Tutorialspoint may be useful for learning Git: <https://www.tutorialspoint.com/git/index.htm>

# 2. Product Overview

## 2.1. Product Perspective

### 2.1.1. System Interfaces

The application is to be developed in C# using the .NET framework. The UI framework will be WPF. As such the target platform is Microsoft Windows.

The product will be supported on the following operating systems:

* Windows 7 (32 and 64 bit)
* Windows 8 (32 and 64 bit)
* Windows 10 (32 and 64 bit)

### 2.1.2. User Interfaces

The user interface is to consist of a command line interface integrated into a graphical user interface.

The command line interface will be the primary control for inputting Git commands. The graphical user interface aims to make certain Git commands and common version control operations easier for the user.

The following is a list of the interface elements to be included in the product:

* Integrated command line.
* Diff viewing window.
* File status window (display for tracked and untracked files)
* Command bar for rapid access to Git commands.
* Status bar.

### 2.1.3. Software Interfaces

Since Git Basic is a Git client, Git will need to be installed on the user’s machine. When Git Basic is installed, the installer will check if Git is installed on the user's machine. If not it will suggest that this is done and provide a link to <https://git-scm.com/>.

## 2.2. Product Functions

Git Basic is a Git client which aims to make version control tasks easier for software developers or other users. The software will consist of a command line interface integrated into a graphical user interface in order to present the benefits of both environments.

The command line interface will support the execution of all Git commands (as this is the basic and typical way of running Git). The graphical user interface aims to remove the difficulty associated with running particular Git commands in a console. For example, the console is not an ideal interface for viewing file diffs, so the graphical user interface will present an alternative to do this in a more convenient view.

The primary features of Git Basic are the following:

* Execute any Git command.
* View file diffs.
* View file status, stage, and unstage from the graphical interface.
* Select predefined Git commands for fast execution.
* Switch between repositories and branches.

In addition to the above features, the product should provide the users with productivity-boosting features such as easy keyboard navigation of controls, mappable hotkey button, and autocomplete for quick text entry.

## 2.3. User Characteristics

While the product can be used by any person using version control, its target user base is software developers. This user group is understood to already have a basic understanding of version control systems and be familiar with the common Git commands.

The tool aims to be a version control “power tool” to make the job of the software developer easier. The tool should support the commonly used workflows of the software developer and make executing Git commands quicker and easier.

The decision to integrate the command line into this tool as the common form of input is partially due to the familiarity of this kind of control to the software developer.

Here are some basic user stories driving the development of this product.

As a software engineer using version control, I want to:

* Execute any Git command from the command line, because it is faster for me to type short commands then to find them in a software menu.
* See and review the diff of my changes before I commit, to make sure I’m committing the right thing.
* Easily stage and unstage files for a commit.
* Switch between branches quickly without having to type out or remember long branch names.
* Have access to features like hotkeys, shortcuts, and text autocomplete for executing Git commands rapidly.

## 2.4. Constraints

The product should be functional without a network connection. However, in the absence of a network connection Git commands will only work on local repositories. This is not a limitation of the product per-se, but rather due to the distributed nature of Git version control.

# 3. Product Requirements

## 3.1. External Interfaces

### 3.1.1. Git SCM

* 3.1.1.1. Git Basic will interface with Git repositories, as such it requires that Git SCM be installed on the user’s computer.

## 3.2. Functions

### 3.2.1. Installer

* 3.2.1.1. If Git is not installed, the installer will notify the user and provide a link to <https://git-scm.com/>

### 3.2.2. Command Line Interface

* 3.2.2.1. The integrated console (command line interface) will support the execution of all Git commands.
* 3.2.2.2. The console will support selection and copying of text.
* 3.2.2.3. The console will support pasting text input into the input region.
* 3.2.2.4. The console will output color coded text for easy interpretation.
* 3.2.2.5. Error output in the console window will be displayed in red text.
* 3.2.2.6. The console will display the current working directory by the text input region.
* 3.2.2.7. The font size in the console may be adjustable.
* 3.2.2.8. Double click in the console should select the word underneath the cursor.
* 3.2.2.9. Triple click in the console should select an entire line.
* 3.2.2.10. Right click on the console should display a context menu with the following options: Copy, Paste, Select All
* 3.2.2.11. The console window will be resizable.
* 3.2.2.12. Pressing tab in the console input region, should toggle through the working directories contents (subdirectories and files).
* 3.2.2.13. Pressing the up/down arrow keys in the console input region, should cycle through command history.
* 3.2.2.14. Pressing Ctrl + Space in the console input region, should open auto-complete for branch names.

### 3.2.3. Review and Diffs

* 3.2.3.1. The diff viewer will support unified and dual viewer comparisons.
* 3.2.3.2. The diff viewer will only compare text based files.
* 3.2.3.3. The diff viewer is resizable.
* 3.2.3.4. The text in the diff viewer is selectable and can be copied.
* 3.2.3.5. Additions are highlighted in green in the diff viewer.
* 3.2.3.6. Removals are highlighted red in the diff viewer.
* 3.2.3.7. Line numbers should be displayed in the diff viewer.
* 3.2.3.8. The name of the file being diffed should be displayed in the diff viewer.
* 3.2.3.9. The file to be displayed in the diff viewer should be selectable from the file status control.
* 3.2.3.10. The diff viewer will be collapsible.
* 3.2.3.11. The file status control will display all modified files.
* 3.2.3.12. The file status control will display which files are staged and which files are unstaged.
* 3.2.3.13. The files status control will allow the user to select a file and stage or unstage it.
* 3.2.3.14. The file status window will support a show in Windows Explorer option for the selected file.
* 3.2.3.15. The file status window is resizable.

### 3.2.4. Productivity Assitance

* 3.2.4.1. Ten buttons will be mapped to common Git commands.
* 3.2.4.2. Clicking on the buttons will execute their corresponding commands in the console window.
* 3.2.4.3. The buttons will be mapped to default commands, but the user will have the ability to change these commands.
* 3.2.4.4. The buttons will have the hotkeys Ctrl + 0-9 mapped to them so that the user can invoke the commands without clicking on the buttons. The first button will be 1 and the last will be 0, so as to spatially map the keyboard to the layout.
* 3.2.4.5. The status bar should display the current repository.
* 3.2.4.6. Switching to a different repository should be possible by clicking on the repository name in the status bar.
* 3.2.4.7. The status bar should display the current branch.
* 3.2.4.8. Switching to a different branch should be possible by clicking on the branch name in the status bar.
* 3.2.4.9. When typing “Git branch” into the console control an autocomplete box should be displayed which allows the user to complete a branch name without fully typing it.
* 3.2.4.10. The working directory of the application should be persisted between sessions.

## 3.3. System Requirements

### 3.3.1. System

* 3.3.1.1. The software should function on Windows 7, 8, and 10.

### 3.3.2. Resources

Since Git Basic simply acts as a client on top of Git SCM, it should not be resource demanding.

* 3.3.2.1. Excluding Git’s time to execute a command, no Git Basic specific action should take more than 1 second to perform.
* 3.3.2.2. The application should use no more than 250 MB of memory at any given time.

Exceptions to the above resource requirements should be listed in this document.