Requirements Specification

1. Introduction

This section gives an introduction and overview of the product included in the Requirement Specification document.

1.1 Overview

This document gives a specific description of requirements of the Distributed Version Control System(DVCS) named **Kron**. There are three parts of the document: the introduction and scope of this software, the overall description of the software functions and it's constrains and risks and the detailed description of the software user interface. This document is intended to be reviewed, discussed and modified with customer and to be provided as a draft user manual.

1.2 Scope

The product is intended to provide a basic DVCS system to users' local file system to assist their version control activity. It will provide user with functions to create a repository, modify and synchronize a repository and merge between different revisions of repositories. The elaborated desired functions are specified by customer and depicted in part three. The software package should be installed before using the user can use the software through command line.

This software needs permission to read and write the file system of user. Some of the functions of this software will make changes to the file system or the files themselves. User should be aware of the content of changes each time using the functions.

2. Software description

2.1 Product functions

This product will enable user to control among different versions of the file system. User can create a repository to track the chosen file system with the product. When user is making changes to the tracked file system, this product can assist user to track the changes and refer to the changes made at a certain time point. User can also use this product to handle changes from different repository or different revisions of a repository for the purpose of cooperation. This product also provides some functions for user to retrieve the log or content of previous

revisions.

There are mainly 2 parts of this product: the repository and the version control system.

The repository is a special file system initialized by the DVCS. It represents the current working system the version control system is working on. Each time user makes changes to a repository and checkout that change to the file system on the computer, the content of folder corresponding to the repository will be changed. Repository also provides interfaces to interact with user in the command line. User can adopt specified commands with passing legal parameters to act correctly on a repository.

The version control system is composed of manifest and changeset. Manifest represents the snapshot of all the files in the repository at a certain time. Changeset represents the changes made by the user each time has commited to the DVCS. Both the content change and the user action will be recorded by changeset. These two objects together can save and retrieve the certain version of a repository thus enable version control on files.

The design of the product has fully considered the future possibilities of functionality extension. When new functions are designed, new modules will be added to the module hierarchy of current ones.

2.2 User characters

The user of DVCS needs to be able to use the command line and understand the file system of personal computer. User should be familiar with the ability of each function after reading the manual of product and be aware of what will happen to the local file system and the repository after executing each command.

2.3 Constrains and risks

This product can only perform version control locally on personal computer and currently not designed to support remote interactions with other users or repositories. The product is also designed to support merging only two repositories. For multiple repository merging, user may need to merge repositories by two each time. Due to the design of data structure, some actions may not be efficient and need to be valued in the testing stage.

The information of crucial change to the version control system is encrypted but still have risk to be decrypted thus content of the change will be disclosed. Changes made to the file system will also possibly induce loss of information due to the action of user.

3. Requirement specification

3.1 Basic function requirement

`init` : Initialize an empty repository to user's specified path.

`clone` : Copy a target repository to current path. The current path will be changed due to the clone operation. It will take the target repository as a parameter.

`add` : Add files to the track list of DVCS for further commit. User needs to specify the file names and paths want to be tracked in the command line as parameters.

`remove` : Remove files that the DVCS are tracking. User needs to specify the file names in the command line to indicate which file to remove.

`status` : Check the status of current repository. The status information will be printed to command line console, including the branch information, the changes made to the repository and the current tracking status of these changes.

`heads` : Show the current heads of current repository. The heads will be printed to the command line console, including information of current heads.

`diff` : Check the differences between the current directory with the latest commit. This command will differentiate all the contents in current file system and contents in the file system last committed to DVCS. This difference information will be printed to the command line console, including differences in file system structures and differences in file contents.

`checkout` : Checkout a specific version to the current working directory. The files added to track list will be retrieved to the certain version, while files not added to track list will not be changed. User needs to specify the version number to checkout in hashcode form.

`cat` : Inspect a file of a certain revision. User needs to specify the file name and the version number in hashcode to the command line.

`commit` : Commit the changes to the DVCS, create a new version for the changes committed. There will be no change be made to the file system. User will need to specify the commit messages for each commit.

`log` : View the changeset of the current repository. The most recent changeset will be printed to the command line, including the information of the content of change, the type of change action, user name and time of the change.

3.2 Basic function interface

For the following commands, if there is no specification of the command line output, it should work out as follows:

- If `$ kron command <parameters>` works normally, nothing will be printed to the command line.
- If `$ kron command <parameters>` fails, there will be a warning message printed to the command line.

`init` : `$ kron init`

`clone` : `$ kron clone <repository>`

`add` : `$kron add <file paths>`

`remove` : `$kron remove <file paths>`

`status` : `$ kron status`

- Output: If command works normally, the status information will be printed as follows:

```
<the branch information>
<change information>
<change details>
<the untracked files>
```

`heads` : `$ kron heads`

- Output: If command works normally, the heads information will be printed as:

```
<heads>:<heads details>
```

`diff` : `$ kron diff`

- Output: If command works normally, the differences will be printed as:

```
<the overview of file differences>
<the details of file differences>
```

`checkout` : `$ kron checkout <hashcode of a version>`

`cat` : `$ kron cat <file path> <file version hashcode>`

- Output: If command works normally, the content of the file will be printed as:

```
<file name>
<detailed file content>
```

`commit` : `$ kron commit <commit message>`

- Output: If command works normally, the command line prints the following message:

```
<files committed>: <file change type>
<the number of total changes made>
```

`log` : `$ kron log`

- Output: If command works normally, the command line prints:

```
<change index>
<author>
<time of change>
<change details>
```

3.3 Advanced function requirements

`pull` : Pull the changes from another repository to the current repository. User will need to specify the name of repository to be pulled. No change will be made to the current directory.

`push` : Push the changes to another repository from the current repository. User will need to specify the name of repository to be pushed to. No change will be made to the current directory.

`merge` : Merge two revisions of a repository. The changes of these two different revisions will be applied to the current working directory. User needs to specify the revision of repository to be merged to the current repository.

3.4 Advanced function interface

For the following commands, if there is no specification of the command output, it should works out as follows:

- If `$ kron command <parameters>` works normally, nothing will be printed to the command line.
- If `$ kron command <parameters>` fails, there will be a warning message printed to the command line.

`pull` : `$kron pull <repository>`

`push` : `$kron push <repository>`

`merge` : `$kron merge <revision of repository>`

- Output: If command works normally, command line will print:

```
<merging status>
<the change applied> : <change details>
```

4. Summary and acknowledgment

This document defines the general usage, detailed functions and the user interface of the product. There are some risks and constraints that are also demonstrated in this document. There are possibilities of function extension for more abilities depending on the schedule and progress of customer and developer.