

可持久化本地代码仓库的实现

柯嵩宇 陈乐群

2014 级 ACM 班

上海交通大学

2015 年 5 月 29 日

目录

1	综述	4
2	项目简介	4
2.1	项目名称	4
2.2	支持的功能	4
2.3	主要模块	4
2.4	主要命令	4
3	模块介绍	5
3.1	Color	5
3.1.1	简介	5
3.1.2	实现方法	5
3.1.3	相关文件	5
3.2	Config	5
3.2.1	简介	5
3.2.2	相关文件	5
3.3	Core	5
3.3.1	简介	5
3.3.2	相关文件	5
3.4	Diff	5
3.4.1	简介	5
3.4.2	实现方法	6
3.4.3	相关文件	6
3.5	Hash	6
3.5.1	简介	6
3.5.2	Hash 的使用	6
3.6	FileSystem	6

目录	2
3.6.1 简介	6
3.6.2 主要函数	6
3.6.3 相关文件	7
3.7 Index	7
3.7.1 简介	7
3.7.2 实现细节	7
3.7.3 相关文件	7
3.8 Objects	7
3.8.1 简介	7
3.8.2 实现细节	8
3.8.3 相关文件	8
3.9 Refs	8
3.9.1 简介	8
3.9.2 相关文件	8
3.10 Status	9
3.10.1 简介	9
3.10.2 实现细节	9
3.10.3 相关文件	9
3.11 Util	9
3.11.1 简介	9
3.11.2 相关文件	9
4 命令介绍	10
4.1 add	10
4.1.1 Introduction	10
4.1.2 Usage	10
4.1.3 Options	10
4.2 config	10
4.2.1 Introduction	10
4.2.2 Usage	10
4.2.3 Options	10
4.3 checkout	10
4.3.1 Introduction	10
4.3.2 Usage	10
4.3.3 Options	10
4.4 commit	10
4.4.1 Introduction	10
4.4.2 Usage	10
4.4.3 Options	10
4.4.4 diff	10

4.4.5	Introduction	10
4.4.6	Usage	10
4.4.7	Options	10
4.5	gc	10
4.5.1	Introduction	10
4.5.2	Usage	10
4.6	init	10
4.6.1	Introduction	10
4.6.2	Usage	10
4.7	log	10
4.7.1	Introduction	10
4.7.2	Usage	10
4.8	status	10
4.8.1	Introduction	10
4.8.2	Usage	10
4.9	reset	10
4.9.1	Introduction	10
4.9.2	Usage	10
4.9.3	Options	10

1 综述

代码仓库在计算机科学的发展过程中扮演了一个非常重要的角色。一个优秀的代码仓库可以帮助程序员高效率地完成工作。这个学期的数据结构大作业，我们二人参考了 git 的设计编写了一个可持久化的本地代码仓库。

2 项目简介

2.1 项目名称

sit(a Simplified Git)

2.2 支持的功能

链式版本（即每个版本只有一个父版本）的保存与历史版本的检出以及对历史版本的修改。

2.3 主要模块

Diff 用于比较版本及文件间的差异。

FileSystem 对 boost::filesystem 的封装，用于对文件的读写，复制及删除。

Index 文件的索引相关的数据结构，用于保存某版本的文件的 hash 值。

Objects 对版本的操作（保存文件，检出和重置版本）的数据结构的实现。

2.4 主要命令

add 把文件加入 index。

checkout 把指定的版本的文件（可以是单个文件，也可以是整个版本）复制到工作区。

commit 将当前的 index 信息作为一个版本的全部文件信息并提交到仓库。

diff 比较特定版本（全版本或特定文件）之间的差异。

init 在当前的工作目录初始化一个版本仓库。

log 输出已保存的版本的版本信息。

status 输出当前工作区的文件信息。

reset 将 index 或 index 中一个特定文件的信息重置为某个特定的版本。

rm 将文件从 index 中移除。

3 模块介绍

3.1 Color

3.1.1 简介

用于实现对输出信息的高亮，使得输出信息的可读性更佳。

3.1.2 实现方法

Windows 使用 Windows API 实现，颜色数量较少，实际效果较差。

Unix 及其他 **POSIX** 标准系统 使用 ANSI 颜色序列实现，色彩丰富，实际效果较好。

3.1.3 相关文件

Color.hpp Color.cc

3.2 Config

3.2.1 简介

sit 的配置模块，用来保存作者信息（包括作者名字和 Email 地址）

3.2.2 相关文件

Config.hpp Config.cc

3.3 Core

3.3.1 简介

sit 的核心模块，sit 的所有命令都调用 Core 中相关函数实现

3.3.2 相关文件

Core.hpp Core.cc

3.4 Diff

3.4.1 简介

用于比较两个文件之间或两个版本之间的差异

3.4.2 实现方法

版本之间 先对两个版本的 index 进行 diff, 等到两个版本的文件列表的差异信息 (Add, Rm, Modify, Same), 然后对差异信息为 Modify 的文件调用针对文件的比较算法。

文件之间 对两个文件逐行 Hash, 对得到的 Hash 值列表执行 LCS 近似算法 (时间复杂度 $O(len * d)$, 其中 len 为两个列表中较长的那个的值, d 为输入数据的差异个数)。

3.4.3 相关文件

Diff.hpp Diff.cc MurmurHash3.hpp MurmurHash3.cc

3.5 Hash

3.5.1 简介

Hash 在本项目中扮演了非常重要的角色, 包括版本的管理和 Diff 算法的实现。

3.5.2 Hash 的使用

使用 SHA1 算法的 Hash 值作为文件名来保存文件。虽然效率上会受到一定的影响, 但为了保证跨平台的兼容性, SHA1 算法直接使用了 boost 库中的实现而不是 git 中使用了大量内嵌汇编的 SHA1 算法。

在针对文件的比较算法中, 先使用了 MurmurHash3 的 Hash 算法, 并且针对 32bit 和 64bit 的系统做出了针对性优化。

3.6 FileSystem

3.6.1 简介

由于 Windows 和 Linux 对文件操作的 API 各不相同, 因此使用 boost 库中的 filesystem 进行文件操作。同时为了方便代码的编写和实现一些必要的操作, 对 boost::filesystem 进行了再封装。

3.6.2 主要函数

CompressCopy 对源文件进行压缩后进行复制, 主要用于 add 命令的实现。

CompressWrite 将一个字符串压缩后写入文件, 主要用于版本信息的压缩与写入。

DecompressCopy 对源文件进行解压后进行复制, 主要用于需要从版本库中复制文件的命令的实现。

DecompressRead 对一个被压缩的文件使用, 返回文件原来的信息, 主要用于对版本信息的解压与读取。

GetRelativePath 得到路径 A 相对于路径 B 的相对路径，用于保证在仓库的子目录中调用 sit 是可以得到正确的执行结果。

ListRecursive 返回包含某一个目录下的所有文件的 **vector**，主要用于实现需要文件夹递归的命令 (add, checkout, etc)

3.6.3 相关文件

FileSystem.hpp FileSystem.cc

3.7 Index

3.7.1 简介

程序运行时用于处理版本的文件列表信息的数据结构,其本质是对 `std::unordered_map` 的再封装。

3.7.2 实现细节

考虑到在程序运行过程中需要处理三种类型的文件列表信息：

1. 保存在“.sit/index”中的已暂存的文件信息
2. 保存在“.sit/objects”中以树形结构储存的文件信息
3. 当前工作目录的所有文件的文件信息

其中所有的文件列表都使用 < 路径, SHA1 Hash 值 > 的键值对保存。

在项目中编写了基类 **IndexBase**，以及其针对文件中的 **index** 信息的派生类 **CommitIndex**、针对“.sit/index”中的文件信息的派生类 **Index**、针对当前工作目录的所有文件的派生类 **WorkingIndex**。

文件列表的读取和写入都是通过 **FileSystem** 模块来实现的。

3.7.3 相关文件

Index.hpp Index.cc

3.8 Objects

3.8.1 简介

作为文件与 **Index** 之间的中间数据结构，在 **Index** 与 **Commit** 文件的转换中具有重要作用。**tree** 类型的 **object** 以树形结构保存了 **index** 中的文件列表，清楚的记录了文件和路径的关系。

3.8.2 实现细节

包含三种类型的 Objects:

commit 这个类型的 Objects 表示该文件记录的是一个版本信息，包含文件的信息和该 commit 的信息（作者，时间，以及 COMMIT_MSG）。

tree 表示这个文件是一个文件列表，记录了同一个文件夹下所有文件的信息，每行一个 object，分别是，文件权限，该 object 的类型，SHA1 Hash 值，以及 object 的名字（如果是 blob 类型则是文件名，如果是 tree 类型，则表示这个文件夹的名字）。

blob 一个基本的 object 类型，表示它是路径树上的叶子节点，记录的是相应的文件（可以是文本文件，也可以是二进制文件）。

保存 Object 的结构体:

```
1 struct TreeItem {
2     int mode; // Linux 文件权限，如10644
3     ObjectType type; // 这个Object的类型：tree or blob
4     std::string id; // 文件的SHA1值，其实等价于文件的路径
5     boost::filesystem::path filename; // 文件路径
6 };
```

保存 Commit 信息的结构体

```
1 struct Commit {
2     std::string tree; // 以REPO_ROOT为根的目录树的SHA1 Hash值
3     std::string parent; // 该提交的父版本
4     std::string author; // 作者的名字，Email地址，时间戳，时区
5     std::string committer; // 提交者的名字，Email地址，时间戳，时区
6     std::string message; // Commit Messages
7 };
```

3.8.3 相关文件

Objects.hpp Objects.cc

3.9 Refs

3.9.1 简介

由于处理与版本指针相关的引用信息，如，当前工作区的版本（HEAD 指针），当前分支的版本指针（指向当前分支的最后一个版本，由于目前的 sit 不支持分支功能，故只有一个默认生成的 master 分支）。该模块的目的主要是为了减少以后需要增加分支功能的时候的工作量。

3.9.2 相关文件

Refs.hpp Refs.cc

3.10 Status

3.10.1 简介

用于处理工作区和 index 以及版本库之间的关系，输出当前工作区的状态。

3.10.2 实现细节

使用了 Index 模块中的 WorkingIndex 类，会与保存在“.sit/index”中的 Index 以及 HEAD 指针指向的 commit 的 Index 比较，得出哪些文件是新出现且没有加入 index 的，那些文件已经被 index 记录，那些文件被删除。如果 WorkingIndex 中的所有文件记录与 HEAD 指向的 commit 的文件记录都相同那么认为工作区是 Clean 的。checkout 操作在检出 commit 的时候要求工作区必须是 Clean 的。

3.10.3 相关文件

Status.hpp Status.cc

3.11 Util

3.11.1 简介

重要的一个模块，作为整个工程的工具模块，其中包含了对 boost 库中 SHA1 算法的封装，以及根据当前的 objects 目录提供的 SHA1 值的补全功能，增强了 sit 的易用性。

在这个模块里还定义了 sit 的异常类 SitException，用于对用户的非法操作给出提示与警告。

3.11.2 相关文件

Util.hpp Util.cc

4 命令介绍

4.1 add

4.1.1 Introduction

4.1.2 Usage

4.1.3 Options

4.2 config

4.2.1 Introduction

4.2.2 Usage

4.2.3 Options

4.3 checkout

4.3.1 Introduction

4.3.2 Usage

4.3.3 Options

4.4 commit

4.4.1 Introduction

4.4.2 Usage

4.4.3 Options

4.4.4 diff

4.4.5 Introduction

4.4.6 Usage

4.4.7 Options

4.5 gc

4.5.1 Introduction

4.5.2 Usage

4.6 init

4.6.1 Introduction

4.6.2 Usage

4.7 log

4.7.1 Introduction

4.7.2 Usage

4.8 status