System Programming

Week 12: Working with others and Regular Expressions

Seongjin Lee

Updated: 2018/07/06
09_vcs-regex

insight@gnu.ac.kr
http://open.gnu.ac.kr
Systems Research Lab.
Gyeongsang National University

Table of contents

Introduction

Working with others

In this lecture we will cover version control system and Regular Expressions

- Subversion or SVN
- GIT
- Regular Expressions

Terminology

The History

Generation	Networking	Operations	Concurrency	Examples
First	None	One file at a time	Locks	RCS, SCCS
Second	Centralized	Multi-file	Merge Before Commit	CVS, Subversion, SourceSafe, Team Foundation Server
Third	Distributed	Change sets	Commit before Merge	Bazzar, Git, Mecucial

The history of version control is very long (about forty years)

- It steadily moved towards to support more concurrency
- First generation used locks to manage concurrency – one person at a time
- Second generation is more permissive about simultaneous modification – merge before commit
- Third generation separates merge and commit operations

Basic Terminology

Repository is a official place to store the work

- Keeps track of tree of files and directories
- More importantly it contains history
- Create operation makes a new repository

$$\textbf{Repository} = \textbf{File system} \times \textbf{Time}$$

Basic Terminology cnt'd

Checkout creates a working copy of existing repository to local storage

Working copy is current copy of the project in the local stroage

- Records timestamp on the working file
- Records the version number of the repository file (to note the start)
- Keeps complete copy of the retrieved file

```
WorkCycleFromStart:
     make a working copy from repository

WorkCycle:
     modify working copy
     update the repository

GOTO WorkCycle
```

Basic Terminology cnt'd

Commit applies modification in the working copy to the repository as a new change set

- Several others modify the working copy and add an operations to a pending changeset list
- Pending changeset – a place where changes wait to be commited
- Commit operation takes the pending changeset and makes it to create a new version of the tree in the repository
- Operations are atomic (all or nothing)

Basic Terminology cnt'd

`Update` renews the working copy with respect to the repository

- Make working copy up-to-date
- Apply changes from the repository, merge them with any changes on the working copy

Basic Terminology cnt'd

ADD – add a file or directory for version control

- After add they become part of the pending changeset

EDIT – modify a file

- Edit operation does not involve the VCS

DELETE – delete a file or directory

- Remove a file or directory from the repository
- Immediately delete the working copy of the file, but they are left in pending changeset
- File / directory in the repository is not really deleted; just making a new tree w/o them

Basic Terminology cnt'd

RENAME – rename a file or directory

- Some of the earlier tools had no support for it; so, should check how your VCS works

MOVE – move a file or directory

- Move file or directory from one place in the tree to another
- Operation is added to the pending changeset

STATUS – list the modifications that have been made to the working copy

- It shows the list of of pending changeset

Basic Terminology cnt'd

DIFF – shows the details of the modifications that have been made to the working copy

- Status for list and diff for what exactly have been changed
- How it prints out the differences is VCS dependent

REVERT – undo modifications that have been made to the working copy

- Throw away all your pending changeset and the return the working copy to the way it was just after the checkout

LOG – show the history of changes to the repository

- Keeps track of every version and changes made to the project including Who, When, and What

Basic Terminology cnt'd

TAG – associate a meaningful name with a specific version in the repository

- To mark a specific instant in the history of the repository with meaningful name

BRANCH – create another line of development

- To fork off into two different directories

MERGE – apply changes from one branch to another

- Used branch to enable the development to diverge, merge is to converge again

Basic Terminology cnt'd

RESOLVE – handle conflicts resulting from a merge

- VERY IMPORTANT

LOCK – prevent other people from modifying a file

- Not all have this feature

Subversion

Installing SVN

- For linux `apt-get install subversion libapache2-svn`
- For mac - Build using source code or check if it has one

To check version of installed SVN

- `svn --version`

Second Generation: SVN

It is a centralized version control system

```
mkdir projectA
svnadmin create projectA/trunk
svnserver -d --root=/Users/James/projectA
```

Second Generation: SVN cnt'd

```
// checkout a repository
svn checkout http://PROJECT.URL/projectA

// add a file you want to manage in svn
svn add YOURFILE

// see what is changed and managed
svn status

// commit your work to repository
svn commit -m "LOG CONTENT"

// see records of changes
svn log --verbose | more
```

Second Generation: SVN cnt'd

Merge changes to the working copy : `svn update`

```
svn update

Select: (p) postone, (df) diff-full,  (e) edit,  (r) resolved,
        (mc) mine-conflict,  (tc) theirs-conflict,
        (s) show all options:
```

- Postpone – deal with the conflict later
- Resolved – mark it as solved
  - `svn resolve accept=working`
- Mine-conflict – use my version as new
- Theirs-conflict – use the repository as new

Second Generation: SVN cnt'd

```
// see what is changed in your file
svn diff -r 1 YOURFILE
```

Example of text with conflict

```
<<<<<<<< .mine
Some text is introduced in this line
This is what James wrote
========
Some different contents in this line
This is what Abraham wrote
>>>>>>>> .r4
```

Second Generation: SVN cnt'd

`svn help` gives minimal usage of commands

The commands are

- create
- delete
- rename
- move
- revert
- tag
- branch
- merge
- resolve

GIT

Third Generation Background: GIT

It is distributed or decentralized version control system

Synchronizing the local and the remote

- **PUSH** – copy changesets from a local repository instance to a remote one
- **PULL** – copy changesets from a remote repository instance to a local one
- Note that not all changes on the local is same as that on the remote

Backgrounds

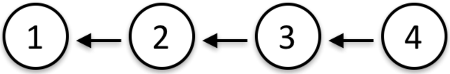
Directed Acyclic Graph (DAGs)

- Ability to push and pull changesets between multiple instances of the same repository comes from a design model called DAG
- Consists of Node, directed edge, root node, leaf node
- Node – represents one revision of the entire repository tree
- Directed edge – shows relationship between nodes

Backgrounds: DAGs

Repository history as a line

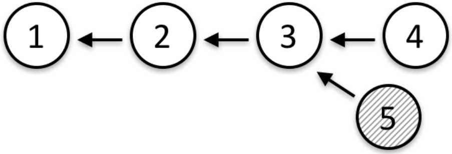
- Fork latest version
- Modify
- Check back in


$$\begin{tikzcd} 1 & 2 \arrow[l] & 3 \arrow[l] & 4 \arrow[l] \end{tikzcd}$$

Backgrounds: DAGs

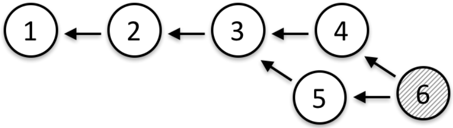
Somebody else does the same

- Before I make change to version 3 somebody else already made version 4



Backgrounds: DAGs

REMEMBER to commit before merge



Benefit of having DAG model

- Everything is not linear
- Flexible and expressive

Advantages of Distributed Version Control System

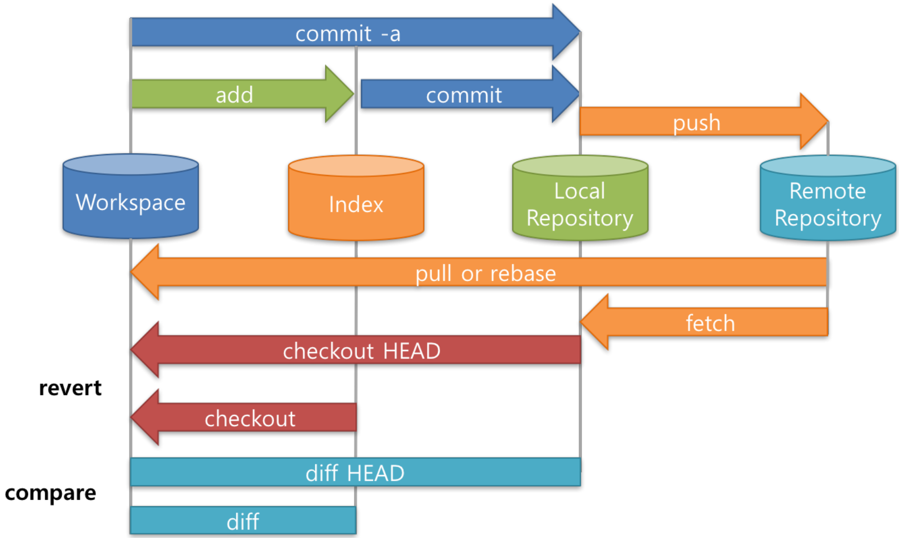
It gives private workspace for the whole repository

It is fast

It works offline

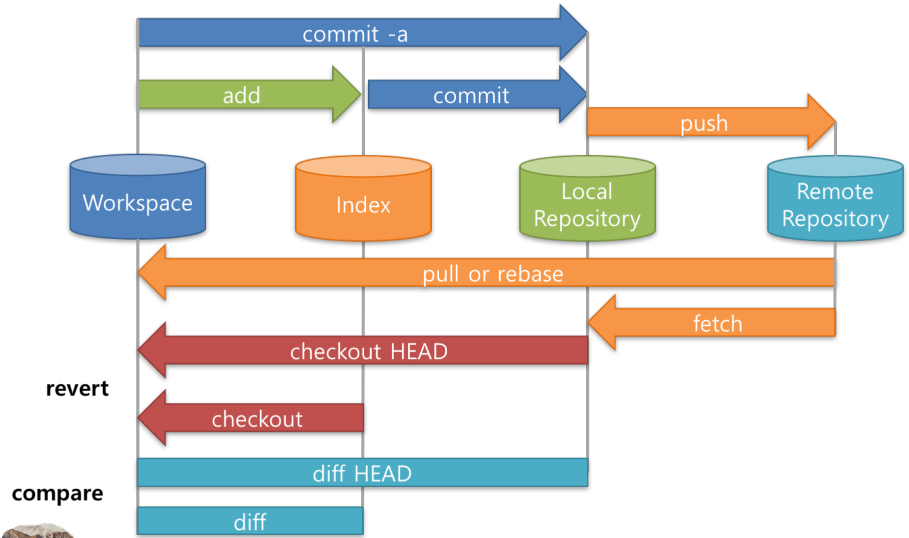
It scales out and up

Work Flow


commit -a
add
commit
push
Workspace
Index
Local Repository
Remote Repository
pull or rebase
fetch
checkout HEAD
revert
checkout
diff HEAD
compare
diff

File Management

It keeps the delta of the object



Installing GIT

- For linux `apt-get install git`
- For mac http://git-scm.com/download/mac

To check version of installed GIT

- `git --version`

Basics

If you have a git server already installed on the local PC

```
mkdir Project-with-git
cd Project-with-git
git init -bare Project-with-git
```

Basics

If you have an account in github.com `vi  /.gitconfig`

```
[user]
   name = YOURID
   email = YOUREMAIL
```

```
git clone address
```

Basics

```
git pull

git add YOURFILE

git commit -m "LOG"

git push
```

Basics

Example text with conflicts

```
<<<<<<<< HEAD
Some text is introduced in this line
This is what James wrote
========
Some different contents in this line
This is what Abraham wrote
>>>>>>>> b30hf32hfaohf8dhaf8a
```

Regular Expression

Search for Strings: grep Overview

Usage

```
grep[1] pattern filename
```

Examples

- `grep hello world`
- `grep ‘‘hello world’’`
- `grep ‘‘h.llo’’`
- `grep ‘‘h*llo’’`
- `grep ‘‘hello \| world’’`

`grep -E` for extended `grep` with regular expression

Download sample text from
http://www.ats.ucla.edu/stat/examples/chp/p176.txt (use `wget` to download)
Examples

- 2 to 3 vowels: `grep -E ‘‘[aeiou]{2,3}’’ brain.txt`
- OR cases
  - `grep -E ‘‘A[sf]*’’ brain.txt`
  - `grep -E ‘‘Asian \| African’’`

`grep -E` for extended `grep` with regular expression

Download a sample text from https://www.gnu.org/licenses/gpl.txt

yet another example of OR search

- `grep -E ‘‘(GPL\| General Public License)’’ gpl.txt`

Meta characters: begin with capital letter and end with period

- `grep -E ‘‘^[A-Z].*\.$’’ gpl.txt`

Examples cnt'd

Optional group of string

- `grep -D ‘‘([cC]opy)?right’’ gpl.txt`

Words with 16 to 20 characters

- `grep -E ‘‘[[:alpha:]]{16,20}’’ gpl.txt`

Groups are used in

- repeating set:
  - `(Love){5}` matches `LoveLoveLoveLoveLove`
  - Love{5} matches: `Loveeeee`
- Back referencing usually used in replacing. It is also called capturing group
  - `grep -E ‘‘foo(bar)?(baz)(quz)’’`

Review with interactive excercises

Interactive excercises: https://regexone.com/lesson/introduction_abcs

Using regular Expression in Vim http://vimregex.com/