Source Control Method & Approach

Jmr Reference

Summary Source control is a central element in secure file storage, transfer, sharing and use. It is also really damn easy and quick when practiced right, cross platform too. The following doc illustrates this with the intent of promoting wide & uniform use.

Author Justin Reina

Date 10/26/17

Platform Coverage
- Windows
- Linux
- Macintosh

Repository Location
- Local, stored only on your computer
  - e.g. on 'C:\' drive
- Remote, stored on an external server
  - e.g. GitHub

Software Selection
- terminal (Win: Cygwin/Linux:tty/
- git Mac:Terminal)
- gitk

Auxillary Interface
- GitHub Desktop
- GitHub Web
- IOS CodeHub

Useful & Recommended
- Cygwin (Windows Bash Terminal)
- Eclipse (Git & Team Viewer Views)

Base Intent
- Track all revisions & changes
- Tag & track releases
- Tag & track development
- Dev ideas (e.g. a new feature, debugging an issue, etc.)

Example (Jmr, ASK Ref Project)
The example shown in Figure 1 illustrates:
- Tags(yellow) – tracking
  - e.g. 'r1' for rev 1, released to team
  - e.g. '6-28_handoff' for last handoff
- Branches (green) – development
  - e.g. 'stat_lib' for the statistics dev
- Form & Structure – type of commit
  - '(+)' – "Addition"
  - '(C)' – "Change"
  - '(B)' – "Bug"
  - '(M)' – "Misc."
  - '(U)' – "Update"
  - '(*)' – "Unknown"

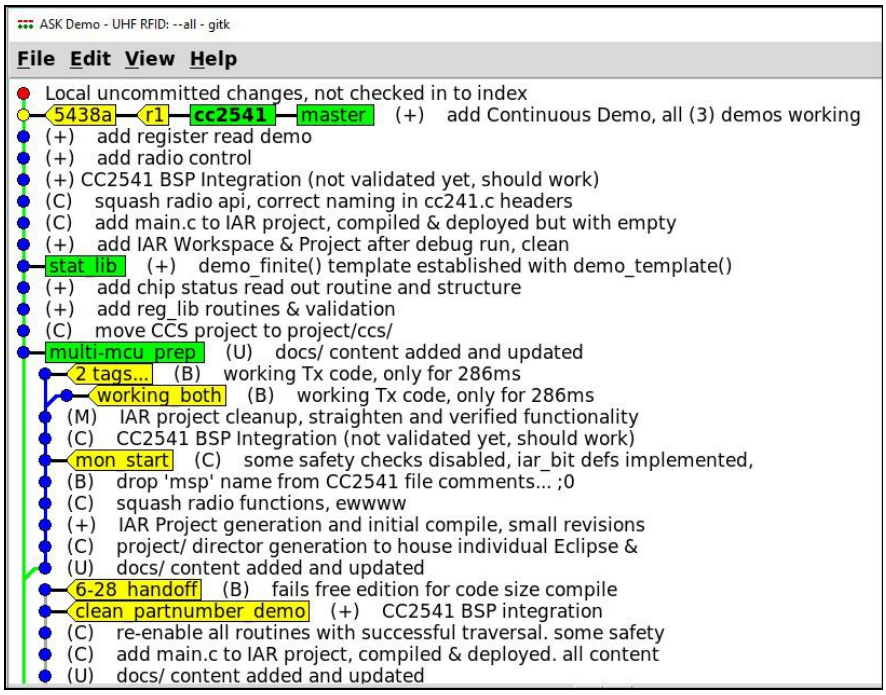

Figure 1: Example Repository

Reference

- Git – About Version Control
- Wiki – Git

About Version Control (VCS)

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

It allows you to revert selected files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more. Using a VCS also generally means that if you screw things up or lose files, you can easily recover. In addition, you get all this for very little overhead.

Local Version Control Systems

Many people's version-control method of choice is to copy files into another directory (perhaps a time-stamped directory, if they're clever). This approach is very common because it is so simple, but it is also incredibly error prone. It is easy to forget which directory you're in and accidentally write to the wrong file or copy over files you don't mean to.

To deal with this issue, programmers long ago developed local VCSs that had a simple database that kept all the changes to files under revision control.

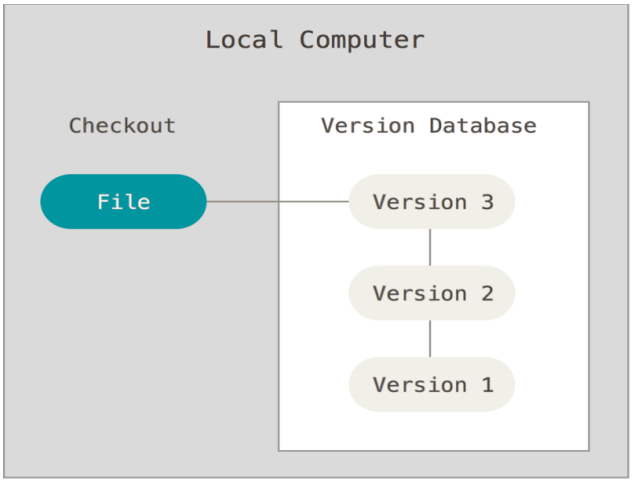

Figure 2: Version Control Example

Vocabulary

git

primary software used to implement version control. Created by Linus Torvalds to enhance & empower the linux movement, in which it has become central and primary in all aspects of development and use. Makes nodes of file contents, strung together into branches, forming a large tree of progress & activity, with the output of releases and products along the way

repository

A file archive where a large amount of source **code** is kept. Includes changes (commits), versions (branches) and releases (tags)

commit ('node')

A point along the path within a repository, a snapshot along a branch with full description of the repository at that moment along a branch

*e.g. at commit #ABCD file *sarah.txt* read "is happy", etc., for all contents in the repo at #ABC

branch

a path through a repository from one point to another

*e.g. for the new idea of popscile headlights we created the popscicle branch. On the first commit here we defined them, 2-5 prototyped, 5-10 created them and 6-20 debugged & finalized them, completing the idea and PoC

merge

any time two branches come together and join

*e.g. branch 'new_feature' was merged into 'master', updating the headlights

head ("HEAD")

Node sitting at the end of a branch

*e.g. "master(HEAD:#ABCD)" means the last commit on the master branch, with a SHA starting with 'ABCD' (for example, '#ABCD123456789...0')

SHA-ID ("SHA")

An ID number in a repository

*'SHA' is the algorithm, which generates a 40 character stream given an input blob of text (e.g. your commit's contents)

tag

An sticker or name applied to a node in a repository

*e.g. "working_new_vers", "friday_handoff", "r1"

master

Primary branch of a repository, from which most content is generated. The trunk, and if no branches all content in the repository lives here!

rebase

Revising Repo Contents

*edit, remove, squash, etc.

Example Use *Cheat Sheet*

Create a New Repository

git init

Grab a Local Copy of GitHub Repo

git clone repo-url (e.g. 'https://github.com/ergsense/DTECTS_hw.git')

Check Repo Status (check for changes)

git status

Commit new content

*git add ** ('*' for all new content, specific names otherwise)

git commit

Reset Your Repo (i.e. reset back to HEAD)

git reset –hard

Switch to a Previous Commit

git checkout *sha-id*

Core Components

Use the following ideas to properly achieve source control.

1. Your project is everything housed within the root directory. This includes:
   source code
   all settings (deployment & development, all!)
   description documents
   pictures
   ... quite literally everything that encompasses and contributes to the definition of your project/product!

2. Only one version of a project (or any file of the project!) is ever left present in the repository at a given time.
   If you want to retrieve a previous version though, it is simple. 'git checkout *sha-id*'!

3. Keep dev & experimentation local, on your PC. Only push to the server (GitHub) when ready, and complete!

4. Rebase or re-work your repository as softly and rarely as possible
   Only commit when complete. Branches otherwise!

Important Commands

Memorize these, to heart. This is 100% of what is needed to successfully implement & maintain source code control.

<u>Creation</u>

- "git init"
- "git status"

<u>Generation</u>

- "git add <*file*>"
- "git add *"
- "git rm <*file*>"

<u>Commit</u>

- "git commit"
- "git reset --hard" (reset to HEAD)
- "git reset --hard <*commit-id*>" (reset to a specific commit)

<u>Review & Correct</u>

- "gitk --all &" (see Figure 1 for example. My primary repo viewer, quick & easy)
- "git commit --amend" (update an existing commit)
- "git rebase -i HEAD~1" (where '1' is how far back you'd like to rebase)

Useful Commands

These are used often, and they promote clean & organized repository development. Establish the habit early, and often!

<u>Tagging & Tracking</u>

- "git tag *tag_name*" (tag a commit with a tag, a name for later use & reference)
- "git checkout -b *branch_name*" (checkout a new branch)

That's it!