389COM: Version Control

Version Control

Dr Carey Pridgeon, DR Nazaraf Shah

Coventry University
http://www.coventry.ac.uk

2016-08-06 Tue

Centralised - 1

- SVN *Apache Subversion*
- All changes are stored in the repository. Your local copy is the current version only.
- Since you only store the current version locally, working on the same codebase from multiple machines means you introduce conflicts very easily.
- Conflicted files are downloaded and timestamped in the filename to allow you to manually merge to get the file you wanted.
- Dropbox's version history takes the same approach as SVN, but they have logic that prevents conflicts.

Centralised - 2

- Sharing a single repository between multiple developers can be done, but it is very easy to break things.
- Branching (creating a parallel copy of your code to test speculative changes) is messy, it involves a complete code tree copy, not just commit pointers (we will go into those).
- Centralised Version Control isn't really relevant for programming any more, the only real benefit is that you have a version history.

Decentralised

- Git and Mercurial
- With both of these there is a sort of central repository, as with SVN, only not as basic.
- Each developer has their own complete local copy, including a full repository history and all branches.
- Code changes are committed locally, then the changed portions in the local repository are pushed (uploaded) to the central repository.
- Commits slot into a tree of versions, unlike SVN it is trivially easy to maintain different versions within the same repository.

Revision History - 1

- Revision history for Git and Mercurial is COW *Copy On Write* based.
- When a change is made to a tracked file (a file that is being managed by version control), a changeset is stored.
- Changesets can be applied to turn one version of a file into the one the changeset represents.
- Successive file *versions* are chains of these Changesets.

Revision History - 2

- Occasionally instead of changesets a new whole file is stored, if there are enough differences to make this worthwhile.
- Every change has a unique id, so it can be referred to later.
- Instead of replacing entire files or groups of files, you can apply a patch that will only update the relevant files.
- while both Git and Mercuriak do basically the same thing, there is still a fairly lively 'Emacs vs Vi' style type debate over which does it better.

Revision History - 3

- Mozilla use Mercurial primarily, but mirror that repository to git for read only access or easy forking.
- Mercurial requires all the history be checked out, Git has the option not to.
- When you checkout the Mozilla codebase you will see why this is an issue for some people. If you didn't notice, it takes a long time.

Conflict Resolution - 1

- Since multiple developers can be editing the same file, conflicts can emerge.
- Git and Mercurial will let you commit any file locally.
- If however that file will cause a conflict on the main (shared) repository, it won't let you push it there.

Conflict Resolution - 2

- This way problems only exist locally, you won't break other developers code by making an unexpected change.
- The solution to a code conflict is usually to get (pull down) the committed file your local file conflicts with, edit in your changes, then commit and push the fixed file.

Conflict Resolution - 3

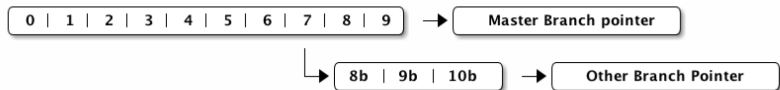
- The easiest way to minimise this is to always pull down the latest copy of the code before working on it. The more you keep your code in sync, the fewer problem you will have.
- If conflicts are very likely because you are working on a major change, the best thing is to create a new branch and work in that. (more on Branches in a bit).

Tagging

- A Tag marks a milestone in development, this can be a release, or a point when the design is changed.
- Tags mean the code state at that point can be downloaded later.
- Create a tag - `git tag v1 -m tag_message`

Branching - 1

- A branch in Git is simply a lightweight movable pointer to the head of a commit chain, of which you can have several.
- Branching splits the code into two streams for parallel development/testing. You commit to a specified stream without effecting the other. This is used for patching or trying a new idea that might break your code.



Branching - 2

- Create a new branch git branch branchname.
- Switch to that branch (or back to another) git checkout branchname
- Or you can combine these git checkout -b branchname
- Pushing the local branch upstream to your repository.
- git push origin branchname
- The disparate branches can be merged later if you want.

Forking

- Forking is just like cloning, except you don't commit to the same upstream repository.
- You can see the current upstream repository url by typing `git remote -v`
- Change the upstream repository with the command `git remote set-url origin new_url`
- Any subsequent commits are in your repository/fork, not the original.
- Gitlab et al make forking really easy.

Submodules

- Code in one git project can be made part of another via the submodule feature
- Submodules are always pointed at a specific commit in the other another codebase.
- This avoids subsequent changes in that codebase breaking your program.
- To update a submodule you have to specify a new pull point (head or some specific commit).

Patches

- Patches are diff files that let git/Mercurial write file changes to yur repository
- Patches should always be applied to a branch of the main project.
- You create the branch, apply the Patch, test the result, and..
- Either you merge the branch back in or discard it if the patch didn't work.
- So, you can test a change without risking your code.

Good Practice

- Commit often.
- Use the features available to track progress.
- Test on each commit using Continuous Integration.
- Use tags whenever you are about to make some major change or set a release version.

Continuous Integration - 1

- CI Builds you code at each commit, runs your tests and shows you the results.
- CI helps you catch errors as they are introduced into your codebase.
- It is is pointless without some form of embedded testing (compiles != works after all).

Continuous Integration - 2

- There are multiple CI platforms - Travis and Jenkins being examples.
- Gitlab is the only free Git hosting service to have it's own system, and it is really easy to use.
- The worksheet associated with this module includes a section where you set up Gitlab CI on a project.

Obligatory XKCD

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.