git and other Version Control Systems

Why, how, and more!

Josh Gordon <joshgordon@umbc.edu>

2014-05-08

Rationale

Why are we doing this?

- Allow collaboration on projects
- Sync config files, etc.
- Keep track of changing files
  - Figure out who messes up a file
  - Reverse catastrophic changes
- Keep track of versions (more useful for large projects)
- share code with others (github, bitbucket, etc.)
- branch off and try implementing new features.

A look at Version Control Systems

Distributed? Centralized? What does it mean?

- Distributed
  - Working branches of repository copied in whole as working copy
  - Advantages:
    - Multiple distributed copies (backups aren't as necessary...)
    - Easy to fork off a new repository.
    - Changes offline can be committed.
    - No central server needed.
  - Disadvantages:
    - Larger space needed for checkout
    - Can't checkout a sub-part of a repository.
- Distributed Examples
  - git
  - mercurial (hg)

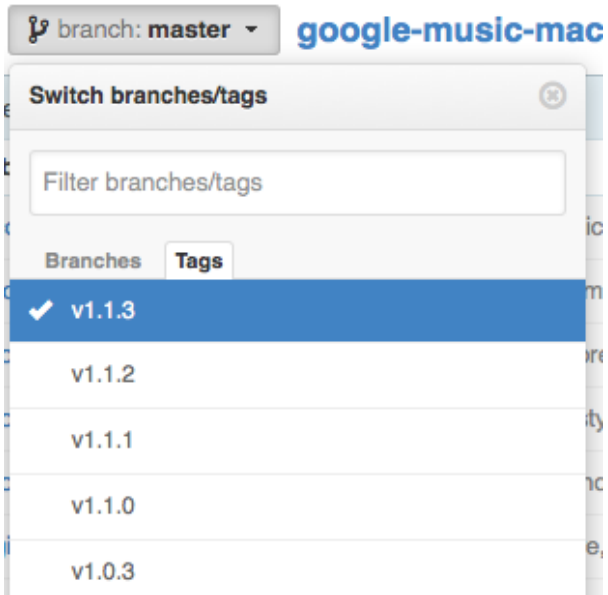
A look at Version Control Systems

Distributed? Centralized? What does it mean?

- Centralized
  - Only head of repository is copied.
  - Advantages:
    - All code is stored in the same place.
    - Code gets an integer version number.
    - Less space required to check out code.
    - Can generally check out a subsection of the repo.
  - Disadvantages:
    - Have to have a central server
    - Have to be online to commit code
- Examples
  - Subversion (SVN)
  - CVS (EWWWW)

Branches and Tags

- Branches
  - Used to do an extended amount of work without affecting the main branch.
  - Development frequently happens on branches and gets merged into the main branch
- Tags
  - Used to mark milestones in the codebase, such as version numbers.



git

History

Who started git development, and why?

History

History

- Initially for Linux Kernel Development
- Developed by Linus Torvalds
- Designed to scale up well. (Kernel is huge!)

A Quick Start

Really easy to start:

- `git init` – creates a new empty repository in the CWD.
- `git add $files` – Stages $files to be added to the repository.
- `git commit -m "comment"` – commits files to the local repository with comment "comment".
- `git push` – if you have a remote set up, pushes to the default remote and default branch.
- `git pull` – if you have a remote set up, pulls from the default remote and default branch.

note that for `git push` and `git pull` you can specify a remote/branch.

Remotes

- Remotes are a way to push and pull from a remote repository. Useful for sharing code with others, or with general population (github/bitbucket).
- Git supports multiple remotes, even within the same repository.
- Cloning a repo automatically adds source repo as a remote named "origin".

```
klingon-3:git josh$ git remote -v
bitbucket https://bitbucket.org/joshgordon/cmsc433.git (fetch)
bitbucket https://bitbucket.org/joshgordon/cmsc433.git (push)
origin josh@river:/git/college/cmsc433 (fetch)
origin josh@river:/git/college/cmsc433 (push)
```

Git supports different types of remotes:

- git
- https
- ssh
- local file-system

More in-depth comparison in online here.

Adding an origin is easy, assuming that the remote repository already exists. (empty or not...)

```
git remote add origin https://github.com/joshgordon/foobar.git
```

Arguments:

- origin - name of remote. (can be anything)
- https://... URL to remote git repository.

Pushing to this new remote is as simple as:
`git push origin master` (or whatever branch you want to push....)

Advanced Topics

Some more advanced uses/topics in git are:

- Keep configuration in git (dotfiles)
- Hooks
- Pull Requests

- Version control can be used to store dotfiles in your home directory.
- GNU's `stow` can make this even easier.

Thank you to Brandon Invergo for providing me with details on how to do this.

```
+-- bash
|-- +-- .bash_profile
|-- +-- .inputrc
|-- +-- .profile
-- emacs
|-- +-- .emacs.d
+-- emacs-mac
|-- +-- .emacs
+-- git
|-- +-- .gitconfig
|-- +-- .gitignore_global
...
```

1. Create a directory in your home directory named `dotfiles`.
2. Create a subdirectory for each config group that you want.
3. Move all your config files from your home directory for that group into it's corresponding directory.
4. Type `stow $group` to symlink everything in $group into your home directory.

Now you can keep your `dotfiles` directory in version control - and have the same config for everything everywhere you go.

```
klingon-3:dotfiles josh$ ssh raspi
josh@raspi's password:
[josh@raspi ~]$ git clone josh@joshgordon.net:/git/dotfiles
Cloning into 'dotfiles'...
[josh@raspi ~]$ ls -a
.  ..  .ssh  dotfiles
[josh@raspi ~]$ cd dotfiles/
[josh@raspi dotfiles]$ ls
bash  emacs  emacs-linux  emacs-mac  git  ssh  tmux  vim
[josh@raspi dotfiles]$ stow git ssh bash
[josh@raspi dotfiles]$ cd ~
[josh@raspi ~]$ ls -a
.   .bash_profile  .gitignore_global  .profile dotfiles
..  .gitconfig     .inputrc           .ssh
[josh@raspi ~]$
```

Hooks

Hooks are used to perform actions at various times:

- Client Side
  - `pre-commit`
  - `prepare-commit-msg`
  - `commit-msg`
  - `post-commit`
  - `post-checkout`
- Server Side
  - `pre-receive`
  - `post-receive`

More details about hooks here.

Hooks are created by adding files to `.git/hooks` with the appropriate names. Examples are there to get you started.

Pull Requests

- Pull requests are used to submit commits to someone else's repository.
- Usually used in conjunction with social coding sites (github, etc)
- Request to "pull" your commits into their repository.

References

- Git Documentation. Also includes the Git Tutorial.
- Github and Bitbucket are sites that host git repos for you.
- Mercurial is another VCS similar to git.
- Subversion (A.K.A SVN) is a centralized VCS.
- A website that pokes fun at the un-readability of Git documentation sometimes.
- A successful git branching model