Introduction To GIT

Rob Di Marco

Philly Linux Users Group

July 14, 2008

A Brief History of Git

- Linus uses BitKeeper to manage Linux code
- Ran into BitKeeper licensing issue
  - Liked functionality
  - Looked at CVS as how not to do things
- April 5, 2005 - Linus sends out email showing first version
- June 15, 2005 - Git used for Linux version control

Git is Not an SCM

Never mind merging. It's not an SCM, it's a distribution and archival mechanism. I bet you could make a reasonable SCM on top of it, though. Another way of looking at it is to say that it's really a content-addressable filesystem, used to track directory trees.

Linus Torvalds, 7 Apr 2005

http://lkml.org/lkml/2005/4/8/9

Centralized Version Control

- Traditional version control system
  - Server with database
  - Clients have a working version
- Examples
  - CVS
  - Subversion
  - Visual Source Safe
- Challenges
  - Multi-developer conflicts
  - Client/server communication

Distributed Version Control

- Authoritative server by convention only
- Every working checkout is a repository
- Get version control even when detached
- Backups are trivial
- Other distributed systems include
  - Mercurial
  - BitKeeper
  - Darcs
  - Bazaar

Git Advantages

- Resilience
  - No one repository has more data than any other
- Speed
  - Very fast operations compared to other VCS (I'm looking at you CVS and Subversion)
- Space
  - Compression can be done across repository not just per file
  - Minimizes local size as well as push/pull data transfers
- Simplicity
  - Object model is very simple
- Large userbase with robust tools

Some GIT Disadvantages

- Definite learning curve, especially for those used to centralized systems
  - Can sometimes seem overwhelming to learn
- Documentation mostly through man pages
- Windows support can be an issue
  - Can use through Cygwin
  - Also have the msysgit project

Git Architecture

- Index
  - Stores information about current working directory and changes made to it
- Object Database
  - Blobs (files)
    - Stored in .git/objects
    - Indexed by unique hash
    - All files are stored as blobs
  - Trees (directories)
  - Commits
    - One object for every commit
    - Contains hash of parent, name of author, time of commit, and hash of the current tree
  - Tags

Some Commands

- Getting a Repository
  - git init
  - git clone
- Commits
  - git add
  - git commit
- Getting information
  - git help
  - git status
  - git diff
  - git log
  - git show

Our First Git Repository

- `mkdir first-git-repo && cd first-git-repo`
- `git init`
  - Creates the basic artifacts in the .git directory
- `echo "Hello World" > hello.txt`
- `git add .`
  - Adds content to the index
  - Index reflects the working version
  - Must be run prior to a commit
- `git commit -a -m 'Check in number one'`

Key Git Files/Directories

- ~/.gitconfig
- .git
  - In top level of repository
  - Contains all objects, commits, configuration, for project
  - .git/config has project specific configurations
- .gitignore
  - Stored in directory for ignoring

Working With Git

- `echo "I love Git" >> hello.txt`
- `git diff`
  - Shows changes we have made
- `git status`
  - Shows list of modified files
- `git add hello.txt`
- `git diff`
  - No changes shown as diff compares to the index
- `git diff HEAD`
  - Now can see the changes in working version
- `git status`
- `git commit -m 'Second commit'`

Viewing What Has Changed

- `git log`
  - Note the hash code for each commit.
- `git show <OBJECT>`
  - Can use full or shortened hash
- `git reflog` to see all changes that have occurred

Git and Patch files

- *`git diff HEAD^^`*
  - Show what has changed in last two commits
- *git diff HEAD~10..HEAD~2*
  - Show what changed between 10 commits ago and two commits ago
- *`git format-patch HEAD^^..HEAD`*
  - Will create individual patch files per commit
- *`git apply`* to apply patches
  - *`git am`* to apply patches from an mbox
- Can also compare
  - Between specific objects
  - To branches/tags

Undoing What is Done

- git checkout
  - Used to checkout a specific version/branch of the tree
- git reset
  - Moves the tree back to a certain specified version
  - Use the --force to ignore working changes
- git revert
  - Reverts a commit
  - Does not delete the commit object, just applies a patch
  - Reverts can themselves be reverted!
- Git never deletes a commit object
  - It is very hard to shoot yourself in the foot!

Git and Tagging

- Tags are just human readable shortcuts for hashes
- Branches can be made from any commit
- *git tag <tag-name>*

Branching

- Git branching is lightweight
  - No massive copying a la CVS/Subversion
  - Tools for helping merge branches and changes easily
- You are ALWAYS on a branch
- Branches can be local or remote
- Key commands
  - git branch
  - git merge
  - git cherry-pick
    - Allows you to choose specific commits to apply
    - You can edit the commits while cherry picking

Using Branches

- `git checkout -b branch`
- `git checkout -b devel/branch`
- `git branch`
  - Lists all local branches available
- We can now make changes in one branch and propagate change using
  - git merge
  - git cherry-pick

Rebasing Example

- Simple branching

```
o--o--o <-- origin
         \
          a--b--c <-- mywork
```

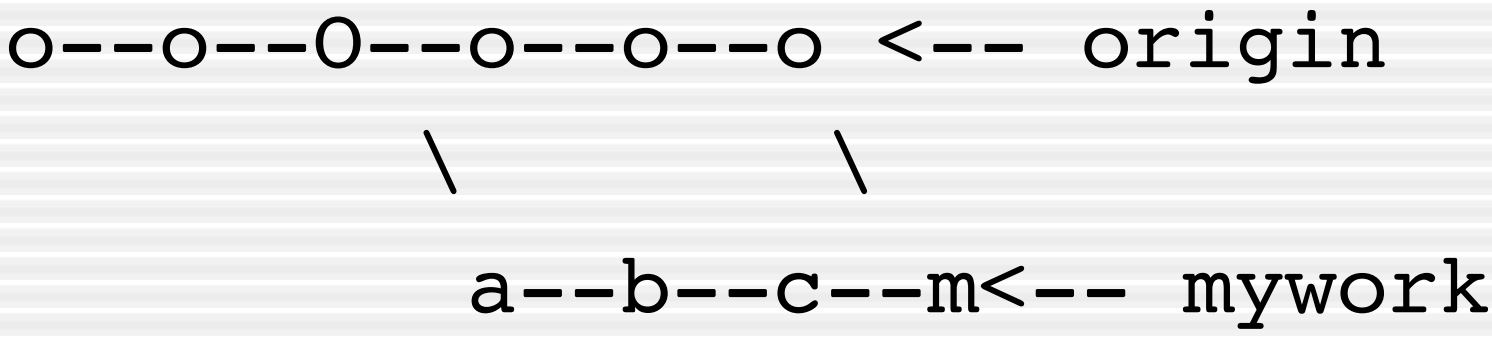
Rebasing Example

- Work done on origin branch

```
o--o--O--o--o--o <-- origin
          \
           a--b--c <-- mywork
```

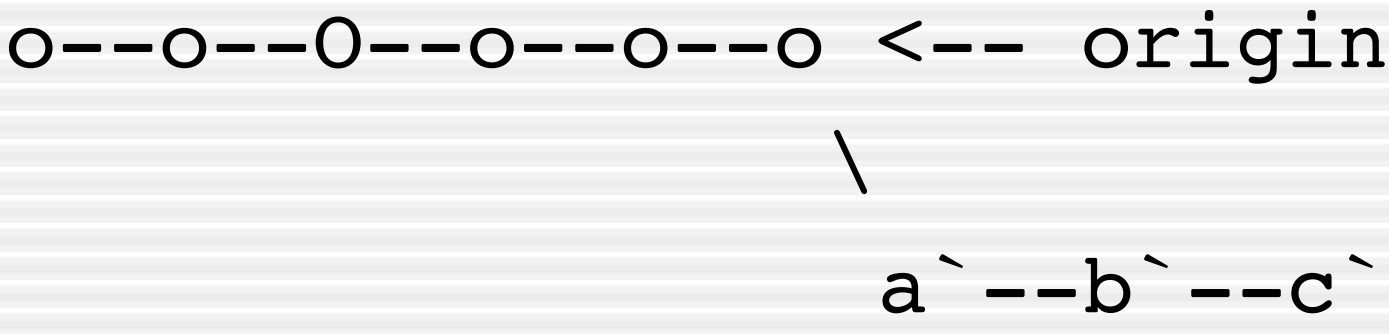
Rebasing Example

- Could merge changes into branch
- *git merge origin*



Rebasing Example

- Rebasing moves branch point
- *git rebase origin*



Cleaning Up

- git fsck
  - Checks object database to make sure all is sane
  - Can show information about dangling objects
- git gc
  - Cleans up repository and compress files
  - When used with --prune, cleans out dangling blobs
  - Can really speed up larger repositories

Using Remote

- Use git clone to replicate repository
- Get changes with
  - git fetch (fetches and merges)
  - git pull
- Propagate changes with
  - git push
- Protocols
  - Local filesystem
  - SSH
  - Rsync
  - HTTP
  - Git protocol

Cloning our Repository

- *git clone first-git-repo*
  - Now have a full git repository to work with
- Changes are pushed back with *git push*
  - Pushing changes WILL NOT change working copy on the repository being worked on
- Branches can be based off of remote branches
  - *git branch --track new-branch remote/branch*
- Remote configuration information stored in *.git/config*
  - Can have multiple remote backends!

Git for Software Versioning

- Create convention to define default server
- Developers clone from central server
- Lots of tools for transmitting patches between developers
- Being used for
  - Linux (obviously)
  - Ruby On Rails
  - Check out http://github.com for a variety of hosted projects

Git for Backups

- Example: Directory needs regular backups
  - Could use rsync but unwieldy in size
- Create Git repository for appropriate directory
  - Regular local commits
  - Regular push to backup location
  - Get simple revision heistory

Git for Configuration Management

- Example: Apache configurations
  - Multiple environments (dev/test/production)
  - Minor differences between environments
    - IP Address
    - Log levels
  - Want to effectively move changes across environments

Git and Other VCS

- Integrations with
  - Subversion
  - CVS
  - Darcs
  - Many others
- Example of integration with Subversion
  - Use git-svn to fetch and commit push
    - Note initial fetch may take a long time as each commit is downloaded individually!
  - Use git commands for everything
  - Branches integrated with tags and branches

Some Git Coolness

- bash/zsh completion
- Gitk
  - GUI to review changes
- git instaweb
  - Used for starting HTTP process for browing project

Further Resources

- http://git.or.cz/
  - http://git.or.cz/course/cvs.html (For CVS users)
  - http://git.or.cz/course/svn.html (For SVN users)
- http://www.kernel.org/pub/software/scm/git/docs/user-manual.html
- http://jonas.iki.fi/git_guides/HTML/git_guide/