Software Revision Control for MASS

Git Basics, Best Practices

Matthew Sell, CSSE Student
MASS Research Participant, February 2014

What is revision control?

The obligatory Wikipedia definition:

“…revision control is any kind of practice that tracks and provides control over changes to source code”

Why bother?

- Provides recovery of, or reference to, previous modifications to source files
- Allows for experimental diversions (branches) with possibility of merge back to mainline (“sandbox”)
- Streamlines collaboration between developers
- All the cool developers use it
  - Maybe not, but if you work in a professional software development environment, you WILL use revision control.

Become proficient with a VCS and you’ll never look back…

Misperceptions…

- "I already have a backup tool"
  - Revision control is NOT a backup tool. Although it *can* provide backups of your code, don't treat it that way.
- "It's an added complexity that I don't need"
  - The benefits outweigh the added effort; much like using an IDE improves productivity over a text editor.
- "I'm the only one working on this project – I remember my changes"
  - Ever wonder why something doesn't work after making a series of changes? Everything seemed to work a couple of weeks ago? Stop wondering and look at the revision history!
- "My project is too small / simple"
  - Small projects grow into bigger ones frequently. Keep your project under revision control from the beginning.
- "I don't make that many mistakes"
  - Nobody is *that* good. Eventually you will make a terrible mistake and revision control **will** save the day.

How it works, basics

1. Project starts life in a “repository” on a server
2. Existing code imported into the repository
3. Project “cloned” to a local “working directory” on a development client
4. Files/directories updated in working copy
5. Changes “committed”, “pushed” to remote repository
6. Working copy “updated” to get peer commits
7. Repository “tagged” upon release or when reaching major milestones

Best Practices

- Repository “mainline” code should ***ALWAYS*** compile
- ***NEVER*** commit code that won’t compile
- Don’t commit “work in progress”; commit only “completed” tasks
  - Work on a “private” branch first
- ***ALWAYS*** provide useful comments when committing updates
- ***ALWAYS*** run unit tests before committing to repository
  - *You DID create unit tests for your modifications, right?*
  - Don’t forget to commit your unit tests, too!
- ***ALWAYS*** tag revisions provided to “customers”

Using Git

- “Pro Git”, Scott Chacon, Apress

  (http://git-scm.com/book)
- “Git – The Simple Guide”, Roger Dudler

  (http://rogerdudler.github.io/git-guide)

  (http://rogerdudler.github.io/git-guide/files/git_cheat_sheet.pdf)
- “Git Cheat Sheet”, Atlassian

  (https://www.atlassian.com/dms/wac/images/landing/git/atlassian_git_cheatsheet.pdf)

What is “Git”?

- Distributed Version Control System
  - Allows “offline” work
  - Duplicates entire repository
- Born from Linux kernel development
  - Unique requirements of large distributed team
  - Frequent branching
  - Need for speed
- Initially designed / developed by Linus Torvalds
  - Need I say more?

The absolute basics…

- Create repository <`git init`>
- Checkout
  - <`git clone drive:/repo/path`>
  - <`git clone /repo/path`>
  - <`git clone username@host.domain:/repo/path`>
- Add files to the “index”
  - <`git add filename`>
  - <`git add *`>
- Update source files
- Commit changes locally <`git commit –m “Commit message”`>
- Push updates to remote repository <`git push origin master`>
- Get changes from remote repository <`git pull`>

Branching, Git's "killer feature"

- Inexpensive operation! Branch early, branch often
- Create a local branch when starting to make changes, merge those changes when satisfied
- Can't finish work on a branch before having to switch tasks? Make another branch!
- Use descriptive branch names:
  - initials_purpose-issuenumber
  - release-1.2.0
  - hotfix-1.2.1
- More info: http://nvie.com/posts/a-successful-git-branching-model

Git Software

- Command-line (Windows, OS X, Linux)
  - http://git-scm.com/downloads
- GUI (Windows, OS X)
  - http://www.sourcetreeapp.com
- Many others available!

Questions ???

Credits:

- Cherie Wasous: Slide Templates
- Jeff Meyer, Fluke Corporation: Git books and practical use information
- Curt Mills, Fluke Corporation: Practical Git use
- Atlassian: SourceTree software, Git presentations, cheat sheet
- Roger Dudler: Simple Git examples, cheat sheet
- Scott Chacon, Apress: Pro Git book