Git for GlueX?

Offline Software Meeting

Mark M. Ito

Jefferson Lab

June 10, 2015

Not Everything Needs to be in Git

- spectrum of projects, only some would migrate to Git
  - resumes
  - GlueX notes
  - talks
  - analysis plugins
  - computing support scripts
  - reconstruction code
  - papers
- concentrate on sim-recon/hdds for now

Subversion vs. Git: Repository Multiplicity

- Subversion: one and only one
- Git: many
- svn: network access necessary for repo-dependent operations, git: repo is local
- git: need to impose administrative controls, i. e., structure, workflow, rules
- git: you clone a repo to get a local copy of entire repo
- git repo "remembers" its parent, every commit has fingerprint, commits arranged in branches, tree of fingerprints enables comparison of different repos
- Public vs. Private: svn only has a public repo, git: public repos visible to all, private repos on local disks

Git vs. Subversion: Commits

- Scope of Commits
  - Subversion: commits all go to The Repository, affects everyone unless on a branch
  - Git: commits are local, thus "branching" is automatic
  - Advantage of development on a branch: save work as you go, but before it is ready for others
- Nature of Commits
  - Subversion: one step
  - Git: two-steps
  - git: first, files are staged for commit (git add), then committed (git commit)
  - set of staged files independent of set of local files independent of committed versions files
  - enables grouping of changes meant to implement fix or feature

Using Git

- Making a new repository: git clone <parent_URL>
  - local directory tree
  - .git directory at top-level only, repo information
- Checking out a branch: git checkout <branch_name>
  - local directory tree put into context of branch
- Moving changes from branch to branch: git merge <source_branch_name>
- Moving changes back to the parent repo: git push <target_URL>
- Bringing local repository up-to-date with parent repo: git fetch, git pull

Workflow Models

1. One Public Repo, Master Branch
   - clone the repo
   - work on master branch
   - push changes to master branch
   - equivalent to traditional "trunk" system
2. One Public Repo, Topic Branch, Self Merge
   - clone the repo
   - work on master branch or local topic branch
   - push changes to topic branch
   - merge into the master branch
3. One Public Repo, Topic Branch, Integrator Merge
   - Same as previous except do not merge yourself, issue pull request to Integrator: please bring in my changes
4. Multiple Public Repos, Integrator Merge
   - each developer has public repo
   - developers push to their own public repo, topic branch
   - issue pull request to Integrator
   - Integrator pulls and merges

Nice Git Features

- fast
- complete command line help, just like svn
- helpful error messages; alternates offered; missing actions suggested
- GitHub added-value items

Extra Steps When Using Git

- Need to invent and enforce workflow structure/policy
- Another system to learn
- Extra steps in workflow

How to Manage Conversion?

- Conversion from Subversion to Git to GitHub a solved problem.
- Ideally: move instantaneously from svn as authority to git as authority
  - otherwise changes in one or other get lost
- ...but climbing learning curve not instantaneous...
  - sudden switch could halt progress
- Non-ideal: bi-directional maintenance of two systems? (yuck)
- More research to do...ideas welcome

Why Change?

- Better management of changes
- Better communication of changes
- Better documentation of changes
- Less down-time with broken code on trunk