EED Git Standards and Workflow (Draft)

Patrick Pascual 8/03/2017

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Quick Intro to Git

SLAC

- Git is a distributed version control system
- No absolute “upstream” or “downstream” repo (user-defined)
- Git tracks changesets, not individual files
- One feature/bug fix can involve changes across multiple files
- Git and related tools (GitHub, GitLab) make collaborative development (especially remote) easier
- Widespread adoption => many APIs and plugins available for a wide range of tools (Trello, Jira, etc.)

Common CVS operations and Git equivalents

SLAC

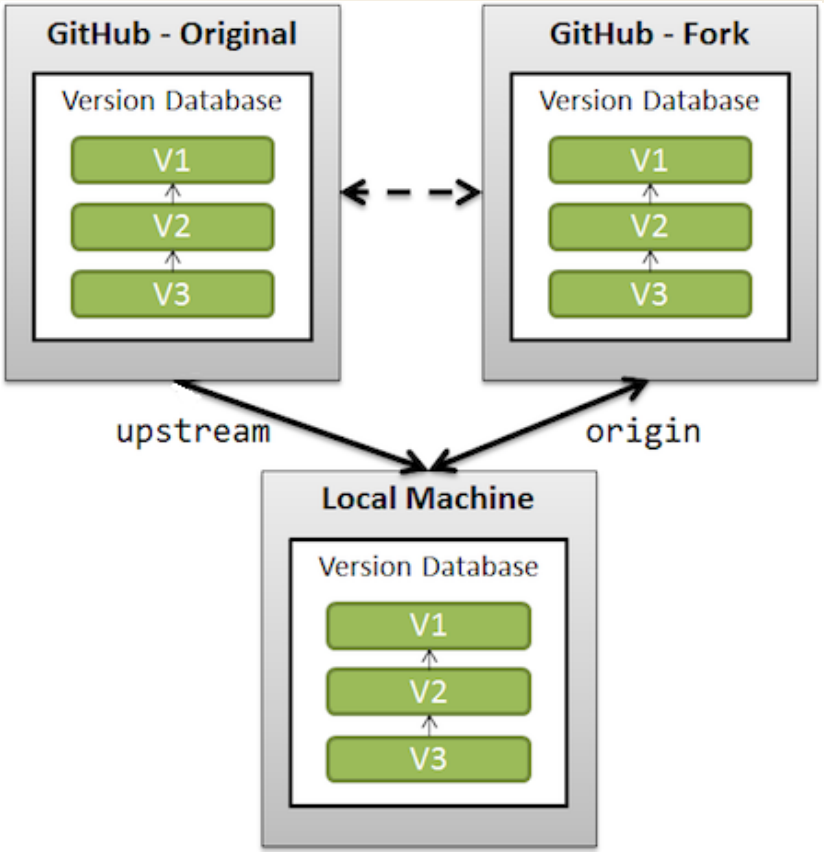
Operation	CVS	Git
Checking out a repo	cvs checkout <module>	git clone <path_to_module>
Adding a file	cvs add <filename>	git add <filename>
Committing changes	cvs commit <filename>	git commit # local repo
Merging revisions	cvs update	git pull [--rebase] # same as git fetch; git merge
*Upstream commits (Git-specific)		git push # may require resolving of conflicts and pull request accepted by repo maintainer
Tagging	cvs tag <tagname>	git tag <tagname> -m “<tagcomment>”

SLAC

Clone and pull

- Anyone can fork an existing repo and push changes without affecting the original source repo
- Changes are pulled *from the clone repo* into the original source repo by generating a pull request
- Maintainers who have push access to the source repo can make changes to the pull request
- This model is typically used for open source projects to help new contributors get to work

Use case: GitHub forks



Collaborative development models, cont.

SLAC

Shared repository model

- Collaborators all have push access to a single shared repo
- Topic branches are generated for changes
- Pull requests are generated when merging the branch into the main repo
- Pull requests initiate code review before merge

Use case: Operations E-Log

Project owner: Patrick Pascual

Developer: Matt Gibbs

Migrating to GitHub

1. Patrick creates a local copy of the latest source repo in AFS
2. Patrick creates a new empty repo at https://github.com/slaclab/mccelog
3. Patrick sets the remote url of his local copy to point at the GitHub repo (git@github.com:slaclab/mccelog.git)
4. Patrick pushes the latest source code from the local repo to the GitHub repo (from an internet accessible AFS machine, e.g. lcls-dev3)

Use case: Operations E-Log, cont.

SLAC

Deploying changes

1. Matt creates a local clone from https://github.com/slaclab/mccelog
2. Matt creates a topic/feature branch in the local clone
3. Upon completion, Matt creates a remote topic branch on the GitHub repo and pushes his local changes up
4. Matt creates a GitHub pull request to merge the new topic branch into the “master” branch
5. Patrick and Matt review the changes in the pull request and iterate as needed
6. Once completed, Patrick accepts the pull request and merges the changes to the GitHub master branch
7. Patrick pulls the changes into an AFS “release” copy of the mccelog repo
8. Patrick “deploys” the changes into Operations E-Log production (to be replaced by directly deploying application as a Git branch/clone)

GitHub forks, cont.

The GitHub Flow (aka Integration Manager workflow)

(Adapted from https://git-scm.com/book/en/v2/GitHub-Contributing-to-a-Project)

1. Fork the project
2. Create a topic branch
3. Commit changes to topic branch
4. Push branch to GitHub project (from step #1)
5. Open a Pull Request on GitHub
6. Discuss/iterate (additional commits)
7. Project owner merges or declines the Pull Request

Transitioning to GitHub

SLAC

Importing a project from CVS

1. Use eco's cvs2git option (from the --help menu):
   'epics-checkout also supports a command called cvs2git "eco cvs2git" that imports a module from CVS into a git bare repo.
   "eco cvs2git" prompts you for a module name and type and repo location.
   It then creates a bare git repo in the location specified; imports the history from CVS and adds a default .gitignore.
   It comments out the module location in the CVSROOT/modules file; however, it does NOT do a cvs remove of the software from CVS.'

Transitioning to GitHub, cont.

SLAC

Importing existing code into GitHub

1. Join the slaclab organization on GitHub by sending your **GitHub** username (which may or may not be the same as your SLAC username) and info requesting access to bvan@slac.stanford.edu
   Alternatively, SLAC employees also have access to Stanford's GitLab Enterprise site by logging into https://code.stanford.edu with their **Stanford** credentials.
2. Create a new private repo on https://github.com/slaclab
3. In your working copy, set the remote-url to point at GitHub:

```
git remote set-url origin git@github.com:slaclab/<reponame>.git
git remote -v
origin  git@github.com:slaclab/<reponame>.git (fetch)
origin  git@github.com:slaclab/<reponame>.git (push)
```

4. Do an initial push to the repo on GitHub (must be done from internet-accessible machine, e.g. lcls-dev3):

```
git push origin master
```

Deploying code

SLAC

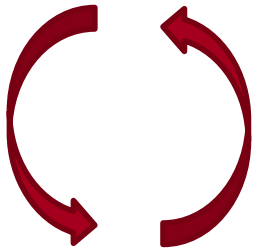
Goals

- EED should have a consistent, uniform deployment workflow *regardless* of role (IOC engineer, collaborator, operator, accelerator physicist, etc.)
- All external collaboration must conform to the EED software deployment workflow as part of EED coding standards and practices
- Corner use cases (e.g., emergency hot fixes) must be accounted for in the EED software deployment workflow
- To encourage modern software practices that help increase productivity and code quality (code review, etc.)

Deploying code, cont.

- Continue to use cram and eco utilities for most applications (EPICS IOCs, etc.)
- (Re-)create staged software deployment (similar to PEPII era):

Development (GitHub, working directory repos)



Test (centralized AFS repos)

Release (“Gold” AFS/production repos)

Deploying code, cont.

SLAC

Future Goals

- Implement continuous integration for builds (Jenkins, etc.)
- Formally integrate EED software deployment with project management/issue tracking (CATER, Jira, etc.)
- Completely test-driven development cycle, with CI, unit-testing, and *DOCUMENTATION* for all applications (documentation as code)
- Containerized application deployment (Docker, etc.), where applicable