How to Do Version Control with Git in your CSE Software Project

The IDEAS Scientific Software Productivity Project
ideas-productivity.org/resources/howtos/

Overview: The distributed version control system Git can be used to establish effective development and integration processes. Achieving this objective requires a basic understanding of Git usage and development workflows.

Target Audience: CSE software project leaders and developers who would like to adopt an appropriate and efficient version control workflow using Git for their software-intensive projects.

Purpose: Describe the basic setup and usage of Git, and outline the different basic building blocks for constructing effective workflows for single software source Git repositories.

Prerequisites: First read the document *What Is Version Control*.

Basic Git Setup: Before using Git on a new machine, perform the following minimal setup:

- Set up minimal Git settings for your account, including "user.name," "user.email," "color.ui," "push.default," and "rerere.enabled" [*1*].
- Install scripts locally for the Git shell prompt (*git-prompt.sh*) and Git tab completion (*git-completion.bash*), and add them to your shell (see documentation in the scripts).

Learn to Use Git: Understanding Git from an algorithms and data-structure perspective, rather than just learning commands, can increase software quality and developer productivity.

- If you are a self-learner, review the ***Git Tutorial and Reference Collection*** [*2*].
- For a more structured approach, take the course ***How to Use Git and GitHub*** [*3*].
- Search Google for specific issues. StackOverflow often has an exact solution.

Basic Tips for Using Git: The following basic guidelines and tips apply to all Git workflows [*1*].

- Format commit messages using a 50-char (or so) summary line, followed by a blank newline, then (optionally) longer explanatory text in paragraphs up to 72 chars wide [*6*].
- Create logical commits (see "SEPARATE CHANGES" in "gitworkflows(7)" [*5*] and "One Commit per Logical Change Solution" in the Udacity Git course [*3*]).
- Create local commits to local branch(es) before using commands that might pollute or destroy uncommitted changes (e.g., "git pull," "git checkout," "git reset," "git rebase").
- Back up your local branch after every few hours of work to some remote Git repo.
- Use "git reflog," "git checkout," "git reset --hard," or a similar command to recover an earlier state of your local repository. Previous states can almost always be restored.
- Don't commit large (generated) binary files to a Git repository. *Git LFS* may help.
- Never force push to a remote shared branch using "git push -f"' unless you and everyone else sharing the branch know what this means. Know how to *protect branches* in your git hosting system of choice.
- Create local "checkpoint" commits and then cleanup commits with "git rebase -i @{u}" before pushing to a remote shared branch (be careful not to rebase public commits).

This material is based upon work supported by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs.

Version 0.2, April 25, 2016

Git Workflow Building Blocks: When choosing or constructing a Git-based workflow, start with the simplest workflow that meets the project's needs and is appropriate to the level of current Git knowledge and skill of the developers. Then, as the project is presented with more challenges, consider augmenting the workflow using the following workflow building blocks (steps 2-5 can be added in any order) [*4*]:

1. **Start: The Simple Centralized Continuous Integration (CI) Workflow** has all developers pull from and push to the shared "master" branch in the one shared repository 'origin' (i.e., the basic SVN workflow). This is a simple but effective agile-consistent workflow and is a good choice for many simpler projects.
2. **Add a "develop" branch** in order to provide a more stable "master" branch that is updated on a regular, frequent basis.
3. **Add shorter-lived topic branches** for sets of related commits (e.g., refactors, bug fixes, work on features) to facilitate easy collaborating, code reviews, and back-outs of many commits (which all improve stability of the main development branch).
4. **Add release branches** for named, supported (multiple) releases of the software and release tags to provide patch releases and full traceability of software versions.
5. **Add longer-lived feature branches** when appropriate to keep the development of some new features isolated on topic branches until they are ready to be released to customers or to avoid cluttering the Git history in case they never make the cut and are therefore never merged into the main development branch. However, long-lived feature branches (as opposed to implementing a feature in several shorter-lived topic branches) can lead to later risky and expensive merges into the main development branch.
6. **Add one or more throwaway integration test branches** to test the integration of the various topic and feature branches that are not yet merged into the main development branch. This procedure helps detect integration problems early and makes more effective usage of computer testing resources.
7. **End: The git.git workflow (i.e., "gitworkflows(7)")** is a combination of the above workflow building blocks and is used for developing many projects, including the Git source code itself (i.e., "git.git" [*5*]) and the Linux kernel. However, because the git.git workflow is complex and labor-intensive, its use is justified only for projects where all the developers are Git savvy and the project's challenges justify its usage.

References

[1] Roscoe Bartlett. *Critical Beginner Git Usage Tips.* IDEAS Scientific Software Productivity Project. *https://ideas-productivity.org/resources/howtos/git-tutorial-and-reference-collection/beginner-tips*

[2] Roscoe Bartlett. *Git Tutorial and Reference Collection.* IDEAS Scientific Software Productivity Project. *https://ideas-productivity.org/resources/howtos/git-tutorial-and-reference-collection*

[3] Udacity. *How to Use Git and GitHub. https://www.udacity.com/course/how-to-use-git-and-github--ud775*

[4] Roscoe Bartlett. *Design Patterns for Incrementally Expanding Git Workflows for Research-Based Projects.* IDEAS Scientific Software Productivity Project. To be published. *https://docs.google.com/document/d/1uVQYI2cmNx09fDkHDA136yqDTqayhxqfvjFiuUue7wo*

[5] *gitworkflows(7) - An overview of recommended workflows with Git*,

This material is based upon work supported by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs.

https://www.kernel.org/pub/software/scm/git/docs/gitworkflows.html

[6] Chris Beams, *How to Write a Git Commit Message*, *http://chris.beams.io/posts/git-commit/*

This document was prepared by Roscoe A. Bartlett with key contributions from James M. Willenbring, Michael A. Heroux and Todd Gamblin.

This material is based upon work supported by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs.