Git 101: An Introduction to Git

Derek Smith
@clok, @derek-smith

Oversee.net

It is easy to shoot your foot off with git,
but also easy to revert to a previous foot
and merge it with your current leg.

– Jack William Bell

Basic Configuration Files

Config File (.gitconfig)

This is the basic config file that is globally set from the users home directory. (ex. /home/user/.gitconfig)

You can add attributes to your .gitconfig by either editing the file directly or through CLI inputs.

It is recommended that you set your user name and email address at a global level.

```
$:~/> git config --global user.name "Derek Smith"

$:~/> git config --global user.name
Derek Smith

$:~/> cat .gitconfig
[user]
    name = Derek Smith
    email = dsmith@oversee.net

$:~/>
```

My typical .gitconfig

```
[user]
   name = Derek Smith
   email = dsmith@oversee.net
[core]
   excludesFile = /home/dsmith/.gitignore_global
   editor = emacs -nw
```

Git Ignore (.gitignore)

Include file extensions, names and negations.

You can set a global ignore if you like or you can add custom ignore files
per repo.

```
$:~/dev/malarkey/> cat .gitognore
# Ignore compiled shared objects
*.so

# Ignore png image files
*.png

# Ignore a singular file
a_very_SpeciFic_file.name

# Ignore an entire directory and all sub dies and files
etc/*

# Negate the ignore above for a portion of files within etc/
!etc/malarkey
!etc/malarkey/*

$:~/dev/malarkey/>
```

The Git Repo Basics

Git manages a repo from the root directory down.

Unlike SVN, the entire repo is stored within one .git directory.

Git Init

To initialize a new simply use 'git init'

This will initialize an empty repo in the current working directory.

Any global configurations you have set will be applied.

```
$:~/my_repo/> git init
Initialized empty Git repository in /home/dsmith/my_repo/.git/

$:~/my_repo/>
```

Git Status

Produces report of all modified, deleted and untracked files within a repo.

Use this to determine what needs to be staged for commit.

If a file meets the criteria in .gitignore then it will not be displayed in
the status message.

```
$:~/my_repo/> git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#  hello_git.txt
nothing added to commit but untracked files present (use "git add" to
track)

$:~/my_repo/>
```

Git Add

Stage all modifications to previously committed files and to any new files that you want to add to the repo.

Performing this command on a directory will recursively add all files.

```
$:~/my_repo/> ls
hello_git.txt   new-file.pl  new-file.rb

$:~/my_repo/> git add new-file.*
$:~/my_repo/> git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   new file:   new-file.pl
#   new file:   new-file.rb
#

$:~/my_repo/>
```

Git Commit

Commit locally staged changes to the the local repo.

Unlike 'svn commit' which will push to the shared repo.

'git commit' will open the editor set in either the .gitconfig or by the 'EDITOR' environment variable.

After save & quiting the editor the commit will complete.

Quiting the editor, w/o saving, will cancel the commit.

'-m "<commit message>"'
'-a' will automatically stage all tracked changes and commit them.

```
$:~/my_repo/> git commit -m "NEATO commit message"
[master (root-commit) 0bde0dd] Initial commit
1 file changed, 1 insertion(+)
create mode 100644 hello_git.txt

$:~/my_repo/> git log
commit 0bde0dddcc1107344e3bdbabc0cefb5a183c3a47
Author: Derek Smith <dsmith@oversee.net>
Date:   Wed Jan 9 12:18:28 2013 -0800

NEATO commit message

$:~/my_repo/>
```

Git Rm

Remove a tracked file from version control and the file system.

If a file is not tracked within the repo, 'git rm' will ignore the file.

```
$:~/my_repo/> ls
hello_git.txt  file_to_delete

$:~/my_repo/> git rm file_to_delete
rm 'file_to_delete'

$:~/my_repo/> git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#  deleted:    file_to_delete
#

$:~/my_repo/> git commit -m "Removed file_to_delete from repo"
[master 40f4fd4] Removed file_to_delete from repo
 1 file changed, 1 deletion(-)
 delete mode 100644 file_to_delete

$:~/my_repo/> ls
hello_git.txt

$:~/my_repo/>
```

Git Branch

Branching is fast and simple within git.

Entire branch structure is contained within the repo.

```
# Create a new branch named 'features'
$:~/my_repo/> git branch features

# List all local branches
$:~/my_repo/> git branch
  feature
* master

# List all remote branches
$:~/dev/malarkey> git branch -r
  github.corp/master
  github.corp/v4
  origin/master
  origin/v4
```

Git Checkout

Switch to another working branch.

```
# Checkout a previously created branch.
$:~/my_repo/> git checkout <branch name>

# Create and checkout a newly named branch.
$:~/my_repo/> git checkout -B <branch name>

# Discard any uncommitted modifications and move the file back to the
HEAD.
$:~/my_repo/> git checkout -- <file name>
```

Now to bring the fun ...

Git Clone

The essential git command.

Similar to 'svn checkout' in that it will locally clone a remote repo.

This major difference is that once a repo is cloned, all commits remain local until explicitly pushed to the remote origin.

```
git clone <path to repo>
```

```
$:~/dev/> git clone dsmith.dev:/home/dsmith/repos/git-demo.git
Initialized empty Git repository in /home/dsmith/dev/git-demo/.git
Checking out files: 100% (113/113), done.
```

Git Remote

The next essential git command.

Remote allows you to add and remove remote repo paths.

Essential for synchronizing multiple machines and repos.

```
# Add a new remote repo
$:~/github.corp/git-demo/> git remote add github git@github.com:clok/
git-tuts.git

# List all remote linked repos
$:~/github.corp/git-demo/> git remote -v
github git@github.com:clok/git-tuts.git (fetch)
github git@github.com:clok/git-tuts.git (push)
origin dsmith@dsmith.dev.corp.oversee.net:~/repos/git-demo.git (fetch)
origin dsmith@dsmith.dev.corp.oversee.net:~/repos/git-demo.git (push)

# Remove a remote repo link
$:~/github.corp/git-demo/> git remote rm github
```

Git Fetch

Check and retrieve an update from a remote repo and branch.

The update will be stored in a branch called '<remote>/<branch>'

```
# Fetch all remote/branches
git fetch

# Fetch only a specific remote/branch
git fetch <remote name> <branch name>
```

Git Merge

Straight forward, merge branch into current working branch.

Git merges are VERY fast.

By default, git will attempt to "fast-forward" the merge.

A fast-forward merge is a merge where the pointers are adjusted, but no merge commit is created.

Use '--no-ff' to have a merge commit created.

```
# Perform the fetch
$:~/github/redis/> git fetch origin unstable
From git://github.com/antirez/redis
 * branch            unstable   -> FETCH_HEAD

# Then merge the fetched code
$:~/github/redis/> git merge origin/unstable
Updating be6cbd3..d7740fc
Fast-forward
 CONTRIBUTING                                       |    7 ++
 COPYING                                            |    2 +-
...
 utils/install_server.sh                            |    2 +
132 files changed, 4403 insertions(+), 1007 deletions(-)
```

Git Pull

Basically a 'fetch + merge'

Similar in function to 'svn update'

```
# Perform the pull
$:~/github/redis/> git pull origin unstable
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 7 (delta 4), reused 5 (delta 2)
Unpacking objects: 100% (7/7), done.
From git://github.com/antirez/redis
   776d987..a785986  master     -> origin/unstable
Updating 776d987..a785986
Fast-forward
 README.markdown |   28 ++++++++++++++++++++++++++++
 1 file changed, 28 insertions(+)
```

Git Push

Attempt to push all commits in local repo to remote repo.

Similar in function to 'svn commit'

```
# Perform the push
$:~/github/git-demo/> git push github master
Counting objects: 8, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 4.00 KiB, done.
Total 5 (delta 3), reused 0 (delta 0)
To git@github.com:clok/git-tuts.git
   20f47b7..9b404fb  master -> master
```

... More Advanced Fun ...

Bare Clone Repository

Makes a publishable form of a repo.

Preferable method for sharing a codebase among multiple developers.

```
git clone --bare <path to repo>
```

```
# Create a bare repository
$:~/> git clone --bare /home/my/codebase/ /remote/repos/codebase.git
```

Submodules

An external repo you want to use within another repo.

NOTE:
External code under git version control should NEVER be copied to a local repo.

Use 'git submodule' to link the desired external code to your repo.

If you need to modify the code in the submodule, then fork it and have a blast!

```
$:~/my_repo/> git submodule add git@github.com:clok/tt.git my-submodule
Cloning into 'my-submodule'...
remote: Counting objects: 35, done.
remote: Compressing objects: 100% (27/27), done.
remote: Total 35 (delta 10), reused 23 (delta 3)
Receiving objects: 100% (35/35), 5.30 KiB, done.
Resolving deltas: 100% (10/10), done.
```

```
$:~/my_repo/> cat .gitmodules
[submodule "my-submodule"]
path = my-submodule
url = git@github.com:clok/tiny-timer.git

$:~/my_repo/>
```

Cloning with Submodules

```
$:~/some/dir> git clone --recursive <path to repo/>

Or

# Clone the repo
$:~/some/dir> git clone <path to repo/>

# Initialize all submodules in the .gitmodules file
$:~/some/dir> git submodule init

# Fetch the code for the submodules
$:~/some/dir> git submodule update
```

The Stash Hash

The git stash command will move unstaged changes to stash hash within the local repo.

This will reset the repo back to HEAD.

A useful tool when merge conflicts arise or you just want to quickly discard changes.

The stash hash can be accessed at anytime within git and will always be there unless you drop the stash.

```
# Move uncommitted changes to stash hash
$:~/my_repo/> git stash
Saved working directory and index state WIP on master: 42b6efc Added
sanitize functions for table mode in Parser.pm
HEAD is now at 42b6efc Added sanitize functions for table mode in
Parser.pm

# List all available stash objects
$:~/my_repo/> git stash list
stash@{0}: WIP on master: 42b6efc Added sanitize functions for table
mode in Parser.pm
stash@{1}: WIP on v4: 4bfd79e Updated documentation for DeviceAtlas.pm
stash@{2}: WIP on master: 3fa9ee6 Added logging
stash@{3}: WIP on v2.0: f8f1f98 Testing out RedisDB.pm
```

Detached Checkout

Checkout a named branch in a detached state.

Will not have the ability to commit back to the repo unless given a name.

Useful for code inspection and experimentation.

```
git checkout <commit>
```

```
$:~/my_repo/> git checkout f14c2d8240e7861902d4d0d481b9a04f3e567b3c
Note: checking out 'f14c2d8240e7861902d4d0d481b9a04f3e567b3c'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at f14c2d8... Added analytics_table.pl build scripts
```

Rebase a.k.a. Squash

Note: Do not squash commits that you've already shared with others. You're changing history and it will cause trouble for others.

The most common use of this command is to take many commits and "squash" them into one mega commit.

This can be useful in keeping a clean commit history, but it does remove the meta-data associated with the commits within.

Thank you! Questions?

Sample Repos

- My .emacs.d on github that is an example of Submodules in use.
- This repo on github

Helper Script

- autoGit.pl

Tools

- github - Get on it! Be a social coder.
- gitk - Default management GUI for *NIX systems. Should be installed by default.
- SourceTree - A fancier gitk from Atlassian for Mac that will help manage svn and git repos.
- Git Extensions A git management GUI for Windows.
- git-cola - A GUI for Linux, Mac and Windows.
- A general list of GUIs

The Dox

- My go to for info on git commands: http://git-scm.com/documentation
- The man pages

Tutorials

- Good list of common examples: Everyday Git
- Git Magic

Tutorial Screen Cast

- PLAYterm: git demo