Git - Single Repository

Kurt Schmidt

Dept. of Computer Science, Drexel University

August 25, 2016

(Stolen largely from Geoff Mainland)

Intro

What is version control

- Track the history of a collection of files (source code)
- Allows us to:
  - See what files changed, and when
  - Compare (`diff`) two ore more versions
  - Recover (check out) older versions of files
  - Experiment with new ideas, features, without the risk of losing existing work (branching) – (not this set)
- Greatly facilitates collaboration (subsequent set)

Version Control with **Git**

- Many version control systems: Bazaar (BZR), CVS, Subversion (SVN), Darcs, Mercurial, Perforce, Visual SourceSafe, etc.
- But **git** has (largely) won
    - Developed to manage Linux kernel source code
    - Popularised by GitHub
    - Widely used

First Repository

Setting up your First Repository

- You can have multiple repositories, in various locations
- The entire subtree is included
- Repositories can contain repositories (let's not do that now)
- Let's set some global settings (`~/.gitconfig`)

```
$ git config --global user.name "Your name"
$ git config --global user.email "Your email"
```

The Actual Repository

Create a new directory, with no files in it, and create an empty repository:

```
$ mkdir lab-git
$ cd lab-git
$ git init
```

```
Inialized empty Git repository in /.../lab-git/.git
```

Adding a New File

Let's create a new file

```
$ echo 'Hello, Wrld!' > hello.txt
$ git status
```

```
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
# hello.txt
nothing added to commit but untracked files present (use "git add" to tr
```

git add

- Any changes need to be staged, or added to the *index*
- `commit` will add staged changes to the repository, clear the index

```
$ git add hello.txt
$ git status
```

```
# On branch master
#
# Initial commit
#
# Changes to be committed:
#             (use "git rm --cached <file>..." to unstage)
#
# new file:  hello.txt
#
```

git commit

- We can now commit

  `git commit [-m` *msg* `]`
- Note, the `-m`. It'll ask you for a message anyway, so might as well do it now
  - Just a quick msg, help you distinguish between commits

```
$ git commit -m "Initial commit"
```

```
[master (root-commit) 59b8633] Initial commit
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 hello.txt
```

Committing Changes

Oops...

Let's fix that error:

```
$ sed -i s/Wrld/World/ hello.txt
$ git status
```

```
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working direc
#
# modified:  hello.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Commit New Changes

- We can add individual files, as before

```
$ git add hello.txt
```

- Or, the `-u` flag will have git search all the files it's previously seen (been added) for updates

```
$ git add -u
```

  - Note, update won't pull in new files

- Now, we can commit

```
$ git commit -m"Fixed typo"
```

Examining the Repository – `log`

```
$ git log
```

```
commit a8cd966467c62aa93efe1069f6a7e0a301eb468b
Author: Kurt Schmidt <kschmidt@cs.drexel.edu>
Date:   Wed Jul 29 19:25:38 2015 -0400

    Fixed typo

commit 59b86338e073bef31e8026761f4c809fcfe29001
Author: Kurt Schmidt <kschmidt@cs.drexel.edu>
Date:   Tue Jul 28 21:58:01 2015 -0400

    Initial commit
```

Examining the Repository – `diff`

- To see the differences between 2 different commits:

```
$ git diff 59b863..a8cd96
```

```
diff --git a/hello.txt b/hello.txt
index 26899e5..8ab686e 100644
--- a/hello.txt
+++ b/hello.txt
@@ -1 +1 @@
-Hello, wrld!
+Hello, World!
```

- Can, optionally, indicate an individual file:

```
$ git diff 59b863..a8cd96 -- hello.txt
```

- To compare working file(s) against last commit:

```
$ git diff HEAD -- hello.txt
```

help, Summary

Quick Summary, So Far

`git init`	Initialise a new repo
`git status`	Check status of working tree
`git add` *files*...	Add files (changes) to index
`git commit -m`*msg*	Commit changes in index to repo
`git log`	See commit history
`git diff`	Compare versions

git help

- Get a list of `git` commands, and a brief description:
  `git help`
- Get more detailed help on a topic:
  `git help` *`topic`*

First Look at Internals

Index vs. Working Tree

- A *working tree* is just a copy of the files checked out of the repository. It's where you work on your files
- The *index* is the set of changes to be committed.
- The index may be different from the working directory
- Changes to the working directory must be added to the index

Branches, `master`, `HEAD`

- A brand-new repository has a single branch
    - You can branch to, e.g., add a feature
    - Merge back in when it's working happily
    - We'll discuss branching in another set
- This default branch is called `master`
- `HEAD` is a reference, points to the current (checked out) branch

SHA-1, commit identifiers

- Each revision (commit) gets a unique identifier

```
commit a8cd966467c62aa93efe1069f6a7e0a301eb468b
Author: Kurt Schmidt <kschmidt@cs.drexel.edu>
Date:   Wed Jul 29 19:25:38 2015 -0400

    Fixed typo

commit 59b86338e073bef31e8026761f4c809fcfe29001
Author: Kurt Schmidt <kschmidt@cs.drexel.edu>
Date:   Tue Jul 28 21:58:01 2015 -0400

    Initial commit
```

SHA-1, commit identifiers (cont.)

- A hash value, created from the committed content, plus a header
  - Note this serves as a padlock. The committed content can't be modified. Well, it's easy to see that is has been
- Each serves as a way to identify individual commits
- You can simply use the first 4 character (providing that they make a prefix unique to the repository)

Recovering

Removing Uncommitted Changes – git reset

- To remove all pending changes from the index
  `git reset`
- **Note**, `--mixed` (the default) does **nothing** to the working tree, to the files in the directory

```
$ touch tmp_file
$ git add *
$ git status
```

```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# modified:  hello.txt
# new file:  tmp_file
#
```

git reset (cont.)

- Didn't mean to include that temp file
- Let's back out all staged changes (clear the index):

```
$ git reset
```

```
Unstaged changes after reset:
M hello.txt
```

```
$ git add hello.txt
$ git status
```

```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# modified:  hello.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
# tmp_file
```

git reset (cont.)

- Add just the file we want, commit

```
$ git add hello.txt
$ git commit -m"Added another line to hello"
```

```
[master 599c722] Added another line to hello
 1 files changed, 1 insertions(+), 0 deletions(-)
```

- Use

  `git reset HEAD` *`file...`*

  to unstage individual files

Hard reset

`git reset --hard`

You're cooking along, and you've gone down a bad path. You just want to go back to the last commit, start again. A hard reset:

- Clears the index
- Returns the working directory to the last committed state

```
$ echo "A bunch of changes that aren't working for me" >> hello.txt
$ git reset --hard
```

```
HEAD is now at 599c722 Added another line to hello
```

```
$ cat hello.txt
```

```
Hello, World!
How's things?
```

Check Out a File

Let's say you don't want to trash all of your changes, you're just unhappy with one file, want to start fresh on that. `git checkout` *`files..`*.

- Recovers the most recently committed version of the file
- (Modifies the file(s) in your working tree)

```
$ echo "a dark path I can't find my way back from" >> trouble_file
$ git checkout -- trouble_file
$ cat trouble_file
```

```
Everything's fine on this line.
$
```

(the `--` is Posix for "no more command options follow")

Removing a File from Future Commits

All right, you've been using a file, and you're done with it. Maybe replacing some library w/a newer one. So, you delete it:

```
$ \rm trouble_file
$ git commit -m "removed trouble_file"
```

```
# On branch master
# Changed but not updated:
#   (use "git add/rm <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working direc
#
# deleted:    trouble_file
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Removing a File from Future Commits - (cont)

That didn't work so well. Not a problem:

```
$ git add -u
$ git commit -m "removed trouble_file"
```

```
[master 99fd33e] removed trouble_file
 1 files changed, 0 insertions(+), 1 deletions(-)
 delete mode 100644 trouble_file
```

Removing a File – git rm

Do it all at once:

`git rm` *`files...`*

```
$ git rm old.lib
```

```
rm old.lib
```

```
$ git commit -m"removed the old lib"
```

```
[master 1fb317b] removed the old lib
 0 files changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 old.lib
```

Undoing Last Commit

Consider the recent history:

```
commit b26a94946c21248aad149557dd3d825848a06e76
Author: Kurt Schmidt <kschmidt@cs.drexel.edu>
Date:   Mon Aug 3 02:13:06 2015 -0400

    Adding curtains, bad path

commit 67fecd740797b3b532d4a34100d4d7654ba3881e
Date:   Mon Aug 3 01:45:59 2015 -0400

    Building on a bad commit

commit ddfb7e2e9580d0303b9e8fa047717a4770e345ce
Date:   Sun Aug 2 23:19:26 2015 -0400

    This is a sad commit, gonna trash it

commit 501eb01dd938dd9a59f8037972007ee0f85dd15a
Date:   Sun Aug 2 23:15:18 2015 -0400

    Last good commit
```

Undoing Last Commit (cont.)

Let's roll back the most recent commit:

```
$ git revert b26a
```

Or, if you've not detached `HEAD`:

```
$ git revert HEAD
```

- Note, this doesn't actually remove the last commit
- Start this on a clean working directory (no changes)
- The revert will take the working tree back to the previous commit state, then commit that
- Safe, no loss of history
- Best way to go, if the repository has been published (shared)

Undoing Last Commit – (cont.)

This is what the commit history looks like now:

```
commit f515f31ada2b7337fac1b5ad4de8b87efb5881fc
Author: Kurt Schmidt <kschmidt@cs.drexel.edu>
Date:   Tue Aug 4 15:20:38 2015 -0400

    Revert "Adding curtains, bad path"

    This reverts commit b26a94946c21248aad149557dd3d825848a06e76.

commit b26a94946c21248aad149557dd3d825848a06e76
Author: Kurt Schmidt <kschmidt@cs.drexel.edu>
Date:   Mon Aug 3 02:13:06 2015 -0400

    Adding curtains, bad path

commit 67fecd740797b3b532d4a34100d4d7654ba3881e
Author: Kurt Schmidt <kschmidt@cs.drexel.edu>
Date:   Mon Aug 3 01:45:59 2015 -0400

    Building on a bad commit
...
```

Undoing Multiple Commits

You can roll back a sequence of commits. Consider the following history:

```
$ git log --oneline
```

```
c3317e7 Continuing bad path
cf10f50 Started a bad path
2e65010 2nd commit, good
11ac734 Init
```

We'd like to get back to the 2nd commit.

Undoing Multiple Commits

```
$ git revert --no-commit 2e65..c3317
```

Or, if you've not detached `HEAD`:

```
$ git revert --no-commit 2e65..HEAD
```

- This apparently, rolls back *to* commit 2e65010
- The `--no-commit` saves us from committing each rollback
- So we must commit (just once)

```
$ git commit -m "Roll back to good state"
$ git log --oneline
```

```
92e61db Roll back to good state
c3317e7 Continuing bad path
cf10f50 Started a bad path
2e65010 2nd commit, good
11ac734 Init
```

Summary of Simple Commands, Recovery

`git rm` *file*	Remove a file
`git add -u`	Add changes to the index†
`git reset`	Reset index
`git checkout` *file*	Discard changes to *file*
`git reset --hard`	Discard *all* changes
`git revert` *key*	Revert repository to state of previous co

†Doesn't see newly created files