Using Git: An Overview

Hashem Nasarat

February 15, 2013

What is git?

- Git: Version Control System (VCS)
- Distributed: Lives on more than one computer
- Keeps track of snapshots of a directory: [VCS] repository = files + history
- Free software
- Really complex (300,000 lines of code, written in C, Bash, and Perl)
- Really useful (helps keep track of what you do, so you make fewer mistakes)
- Started by the Linus Torvals, who also started the Linux kernel
- Git $\neq$ GitHub, though Git's code is available there.

Using Git

- Read the manual
- `git help` *<command>*
  - Warning: the documentation is pretty gross
- The following are the most useful bits of git knowledge I've picked up over the years, starting from the basics.
- If you're experienced with the basics of git (*add, commit, push, pull*) and you don't want a refresher, you could try digging a little deeper with Git From The Bottom Up for a few minutes.

Git: Local Repository Only

Basic Usage (local repository)

- `git init` – Create a repository

```
repo   files
 ---
```

```
[   ]   *
[___]
```

- `git add` – I want you to record the state of the following files

```
 ___
[ * ]
[___]
```

- `git commit` – Actually do it. (And attach a message describing changes)

```
 _________________
[ * (HEAD) message ]
[_________________]
```

- `git commit` (with no arguments) will open *vim*. Press `i`, write the commit message, press Esc then save and quit, by typing `:wq`
- **HEAD**: the current commit

Basic Usage (local repository)

- *rinse, repeat*

```
 __________________
[ * (HEAD) message3 ]
[ * message2        ]
[ * message         ]
[__________________]
```

Other Useful Basic Commands

- `git rm` *<filename>*
  - Remove a file from the repo
- `git mv` *<from-filename> <to-filename>*
  - Rename a file in the repo

Git: Local and Remote Repositories

GitHub Basics

- Git repositories can be synchronized between multiple local and multiple remote computers (e.g. your laptop, halligan, and GitHub).
- GitHub offers public remote repositories
- Remote repositories allow distributed development
- Get set up here
    1. Create an account
    2. Create a GitHub repo
    3. Follow the directions (git remote add origin *blah*, and git push -u origin master)

Basic Usage (local + remote repository)

- A repository's `.git/config` file has details
- Default remote name is `origin`
- *git remote add origin https://github.com/Hnasar/test.git*
- Local and remote repositories has benefits:
    - Work on stuff without an Internet connection
    - Work on a project from different computers
- Added complexity:
    - Manually keep changes synchronized.
    - Combining some changes requires intervention (a conflict)

Remote: Initial State

- Empty remote, new repository

```
 local         remote
(laptop)      (GitHub)
 ____          ____
[ *  ]        [    ]
[ *  ]        [    ]
[ *  ]        [    ]
[____]        [____]
```

Remote: Updating the remote

- Update the remote with local changes with `git push`

```
 local           remote
(laptop)        (GitHub)
 ____    push    ____
[ *  ]  ----->  [ *  ]
[ *  ]          [ *  ]
[ *  ]          [ *  ]
[____]          [____]
```

- (DANGEROUS if the remote repo gets completely messed up, try `git push -f` It's usually better to resolve conflicts rather than do this. This can lead to lost data)

Remote: Creating a new local

- Download an entire remote repository to a new local copy with `git clone`

```
 local           remote         local
 (laptop)       (GitHub)        (halligan)
 ____            ____    clone    ____
[ *  ]          [ *  ]  ------>  [ *  ]
[ *  ]          [ *  ]           [ *  ]
[ *  ]          [ *  ]           [ *  ]
[____]          [____]           [____]
```

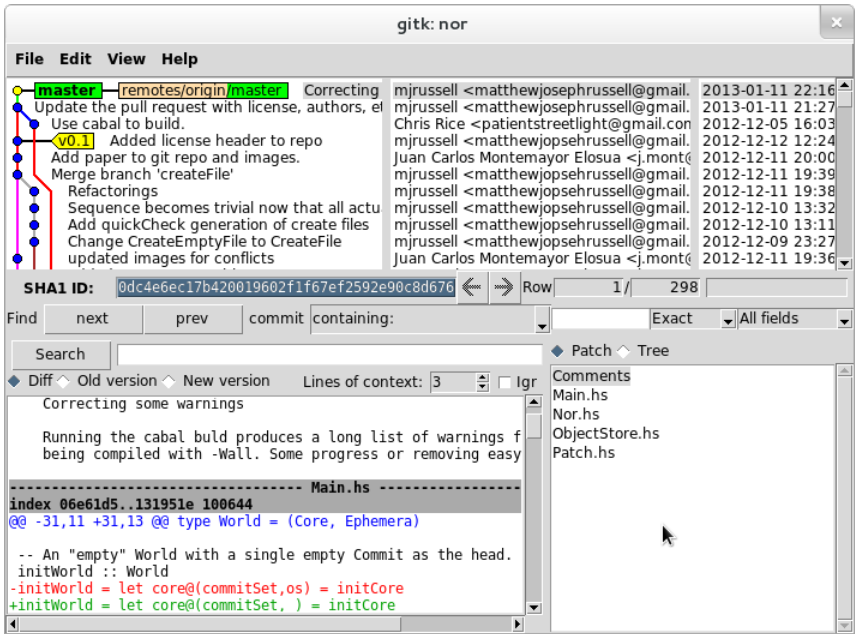
Remote: Updating the local

- Update local repositories with remote changes with `git pull`

```
 local           remote         local
 (laptop)       (GitHub)        (halligan)
 ____            ____             ____
[ *  ]  (push)  [ *  ]   pull    [ *  ]
[ *  ]  ----->  [ *  ]  ------>  [ *  ]
[ *  ]          [ *  ]           [ *  ]
[ *  ]          [ *  ]           [ *  ]
[____]          [____]           [____]
```

- *`git pull` is usually bad form. Use `git pull --rebase`*
- Read this article for more info.

Viewing a Repository 1

- `gitk --all`
- 

```
master  remotes/origin/master  Correcting              mjrussell <matthewjosephrussell@gmail.     2013-01-11 22:16
Update the pull request with license, authors, et      mjrussell <matthewjosephrussell@gmail.     2013-01-11 21:27
  Use cabal to build.                                  Chris Rice <patientstreetlight@gmail.con   2012-12-05 16:03
v0.1  Added license header to repo                     mjrussell <matthewjopsehrussell@gmail.     2012-12-12 12:24
  Add paper to git repo and images.                    Juan Carlos Montemayor Elosua <j.mont(     2012-12-11 20:00
  Merge branch 'createFile'                            mjrussell <matthewjopsehrussell@gmail.     2012-12-11 19:39
    Refactorings                                       mjrussell <matthewjopsehrussell@gmail.     2012-12-11 19:38
    Sequence becomes trivial now that all actu         mjrussell <matthewjopsehrussell@gmail.     2012-12-10 13:32
    Add quickCheck generation of create files          mjrussell <matthewjopsehrussell@gmail.     2012-12-10 13:11
    Change CreateEmptyFile to CreateFile               mjrussell <matthewjopsehrussell@gmail.     2012-12-09 23:27
    updated images for conflicts                       Juan Carlos Montemayor Elosua <j.mont(     2012-12-11 19:36
```

SHA1 ID: 0dc4e6ec17b420019602f1f67ef2592e90c8d676   Row 1 / 298

Find next prev commit containing: Exact   All fields

Search   ◆ Patch ◇ Tree

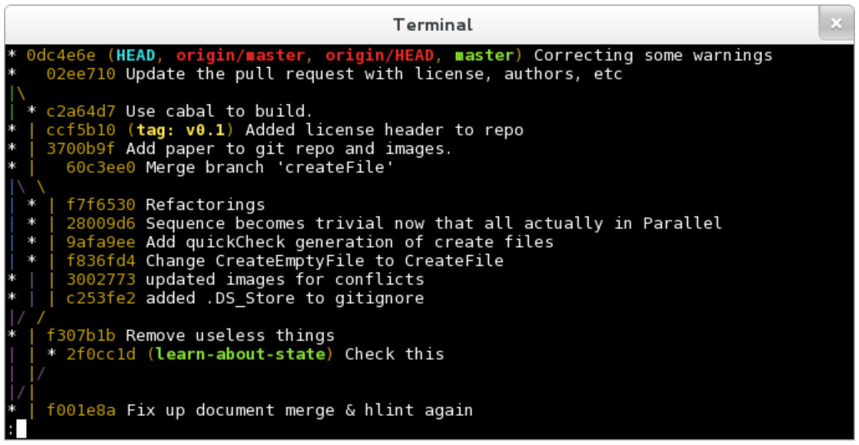
◆ Diff ◇ Old version ◇ New version   Lines of context: 3   Igr

```
    Correcting some warnings

    Running the cabal buld produces a long list of warnings f
    being compiled with -Wall. Some progress or removing easy

---------------------------------- Main.hs -----------------
index 06e61d5..131951e 100644
@@ -31,11 +31,13 @@ type World = (Core, Ephemera)

 -- An "empty" World with a single empty Commit as the head.
 initWorld :: World
-initWorld = let core@(commitSet,os) = initCore
+initWorld = let core@(commitSet, ) = initCore
```

Comments
Main.hs
Nor.hs
ObjectStore.hs
Patch.hs

- available on halligan, Ubuntu/Debian, homebrew

Viewing a Repository 2

- `git log --graph --oneline --all --decorate`
- mnemonic: (git log g.o.a.d.), goad, meaning it's annoying to type all that
- (Shortcut: `Ctrl + r`, then start typing `git log --graph` …)
- 

```
* 0dc4e6e (HEAD, origin/master, origin/HEAD, master) Correcting some warnings
*   02ee710 Update the pull request with license, authors, etc
|\
| * c2a64d7 Use cabal to build.
* | ccf5b10 (tag: v0.1) Added license header to repo
* | 3700b9f Add paper to git repo and images.
* |   60c3ee0 Merge branch 'createFile'
|\ \
| * | f7f6530 Refactorings
| * | 28009d6 Sequence becomes trivial now that all actually in Parallel
| * | 9afa9ee Add quickCheck generation of create files
| * | f836fd4 Change CreateEmptyFile to CreateFile
* | | 3002773 updated images for conflicts
* | | c253fe2 added .DS_Store to gitignore
|/ /
* | f307b1b Remove useless things
| | * 2f0cc1d (learn-about-state) Check this
| |/
|/|
* | f001e8a Fix up document merge & hlint again
:
```

Committing and Good Commit Style

Adding and Committing

- Commits are the basic unit of a repository
- Mark a new state of files at a point in time
- Commits are referenced by their hash (e.g. 0dc4e6e – see *Viewing a Repository 2*, above)
- Commit message message indicates to viewers what the changes in the commit did.
- (Use `git diff` to see what was changed from the last commit)
- 2-part command

    1. `git add` <*path[s]*> – record these changes in the next commit
    2. `git commit` – make the commit, and add a message

- (try `git add -p` to select exactly which changes within files are added)
- Before a commit is made, `git reset` (without any arguments!) will undo `git add`

Commit Style

- A good commit will contain only the changes necessary to some new feature of a repository.

- E.g. If the feature is: "ensure all `img` tags have an `alt` attribute", a good commit will add `alt` tags for every `img` in one go, and NOT create a new commit for every changed `img` tag, or every file that I change things in.

- Good commit message form:

    - Feature in present tense
    - One blank line
    - Explanation/reasoning of changes

```
Add alt attribute to every img

As per Section 508 Amendment to the Rehabilitation Act of 1973
and the HTML 5 specification, every img should have an alt
attribute which "provides equivalent content for those who
cannot process images or who have image loading disabled".
```

Undoing Commits and Fixing Things

Git reset

- `git reset --hard` *<commit>*
  - DANGEROUS – you will lose any **uncommitted** changes
  - used to undo commits
  - Moves branch label, and HEAD to commit specified

Git reset example

```
* 31a3f57 (HEAD, master) Third commit
* 20ea82d Second commit
* 9ef5cfb First commit
```

- `git reset --hard 20ea82d`

Git reset example

```
* 20ea82d (HEAD, master) Second commit
* 9ef5cfb First commit
```

Un-undoing Commits

- Commits are only truly deleted after a given time passes (several days)

- `git reflog`
  - displays most recent commits which have been HEAD

```
20ea82d HEAD@{0}: reset: moving to HEAD~1
31a3f57 HEAD@{1}: checkout: moving from 20ea82d to master
20ea82d HEAD@{2}: checkout: moving from master to HEAD~1
31a3f57 HEAD@{3}: commit: Third commit
20ea82d HEAD@{4}: commit: Second commit
9ef5cfb HEAD@{5}: commit (initial): First commit
```

- `git reset --hard 31a3f57`

Un-undoing Commits

- *Back to the start!*

```
* 31a3f57 (HEAD, master) Third commit
* 20ea82d Second commit
* 9ef5cfb First commit
```

Working with Branches

Branches

- Branches allow multiple lines of commits, which may be dealing with differing features, to not overlap (which might cause confusion).

- A branch is a label attached to a commit.
- Default branch name is `master`
- View branches (including the current one) with `git branch -a`

```
  * 7a0fc15 Patch.hs: Fix incorrect editsToChangeHunks offsets
  * e564f63 Make the type of Edit more general.
* | 0bbe999 Implements applyPatch
* | b6d7003 Implements sequencePatches
|/
* 6f2a864 Paralell patch changes
```

Using Branches

- Create a branch with `git branch` <*branch-name*>
- Delete a branch with `git branch -d` <*branch-name*>
- Switch branches with `git checkout` <*branch-name*>
- When you commit, the new commit's parent is the tip of the current branch, and the branch will now point to the new commit.
- A successful Git branching model

Combining Branches

1. `git merge` <*branch to merge in*>
   - Produces a commit with multiple parents

```
*   ca5ac46 Merge branch 'master' of github.com:jmont/nor
|\
| * 7a0fc15 Patch.hs: Fix incorrect editsToChangeHunks offsets
| * e564f63 Make the type of Edit more general.
* | 0bbe999 Implements applyPatch
* | b6d7003 Implements sequencePatches
|/
* 6f2a864 Paralell patch changes
```

2. `git rebase` *<branch to rebase onto>*
   - Removes the branch by making the branch's commits stem from the end of the other.

```
* 7a0fc15 Patch.hs: Fix incorrect editsToChangeHunks offsets
* e564f63 Make the type of Edit more general.
* 0bbe999 Implements applyPatch
* b6d7003 Implements sequencePatches
* 6f2a864 Paralell patch changes
```

Conflicts

- Git is smart about what lines changed in which files in a commit

- Some commits indicate contradicting changes.
- If git can't figure it out, it writes both version the file, complains of a conflict and tells you to fix it.
- Make the file look how you want, then do `git add .` and `git commit`

```
these lines
are not
in conflict
<<<<<<
THESE ARE IN CONFLICT!
======
These are in conflict.
>>>>>> version 2
these lines
are good too
```

GitHub Forking and Pull Requests

GitHub: Forking

- GitHub “Forking” is something GitHub invented (not a part of git)

- GitHub “Forking” is a way to copy a remote git repo from one GitHub user to another.
- “fork” typically means taking an existing project, and developing it in a new direction. This is what happened when Ubuntu forked from Debian. We say Debian is the upstream.

```
 remote          remote
 tuftsdev/       hnasar/
 running-dogs    running-dogs
 ____            ____
[ *  ]   fork   [ *  ]
[ *  ]  ----->  [ *  ]
[ *  ]          [ *  ]
[____]          [____]
```

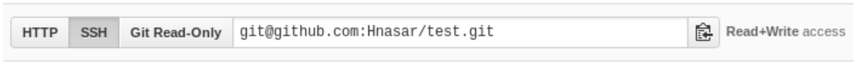
GitHub: Pull Request

- Typically, free software software developers share patches (modifications to code, try `git format-patch` <*commit*>) via email or posting on websites.

- GitHub created a notion of a “Pull Request” to easily allow GitHub “Forked” projects to collaborate in a similar fashion as sharing patches.
- Good explanation
- Pull Requests must be accepted by the recipient.

```
 remote          remote
 tuftsdev/       hnasar/
 running-dogs    running-dogs
 ____           ____
[_*__] <------ [ *  ]
[ *  ]   pull  [ *  ]
[ *  ] request [ *  ]
[ *  ]         [ *  ]
[____]         [____]
```

GitHub: No passwords

- Possible to use GitHub without typing in username & password each time
- Set up SSH keys
- Make sure that your remote URIs are set to `git@github.com/`...
- 

`git@github.com:Hnasar/test.git`

- Check a repository's `.git/config` file

Time Travel

Checkout & Blame

- checkout moves HEAD (the current commit, and the corresponding state of the files)
- (remember `git log --graph --oneline --all --decorate`)
- `git checkout` *<commit-hash>* (e.g. `git checkout 0dc4e6e`)
  - 'detached HEAD' state, which means HEAD isn't on a branch
  - `git checkout` a branch to "reattach" the HEAD
- `git blame` *<file>* to see when and who last made changes to a part of a file.
- `git show` *<commit>* displays the contents of a given commit.

Commit-ishs

- **commit-ish**: some way to reference commits
  - branch-name
  - commit-hash
  - relationships
- '~' - ancestor (defaults to ~1)
- Ex: HEAD~2 (parent of parent of current commit)

Rewriting History

- `git rebase -i` *<commit-ish>*
- reorder, delete, squash commits
- Used to make it look like you knew what you were doing
- Never rewrite commits which you have already pushed (messes up everyone else)

End

Questions/Comments

- More reference available here
- Still unclear?
- Did I miss something?

Quiz

1. Create a repository
2. Commit thrice
3. Create a branch from the first commit and add another commit or two.
4. Merge the new branch into master
5. Undo the merge in master
6. Rebase the new branch onto master instead
7. Squash all the commits on top of the initial commit
