# git and version control

## ymli

**CSCC System Admin 2014**

Where programmers learn from the masters

## About me

- Your TA

## Introduction

- source code management (SCM)
    - why do you need that?
- history
    - Linus Torvalds wanted his own SCM system for the Linux kernel
    - `git` as "*the stupid content tracker*"
        - British English slang roughly equivalent to "*unpleasant person*"

http://git-scm.com/book/en/Getting-Started-A-Short-History-of-Git

## Write my own BitKeeper

- BitKeeper is a proprietary SCM system, *non-free!*
- The new SCM should support a distributed, BitKeeper-like workflow
- Performance is No. 1

## Write my own BitKeeper

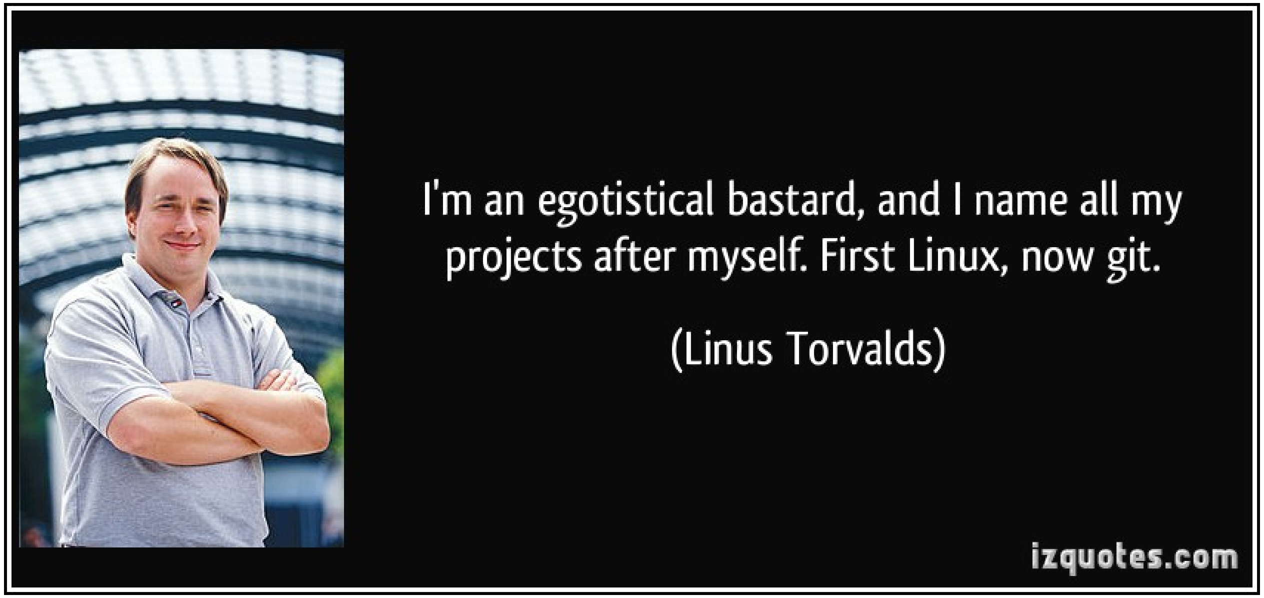

I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git.
(Linus Torvalds)
izquotes.com

image src: [http://akifrases.com/frase/132446](http://akifrases.com/frase/132446)

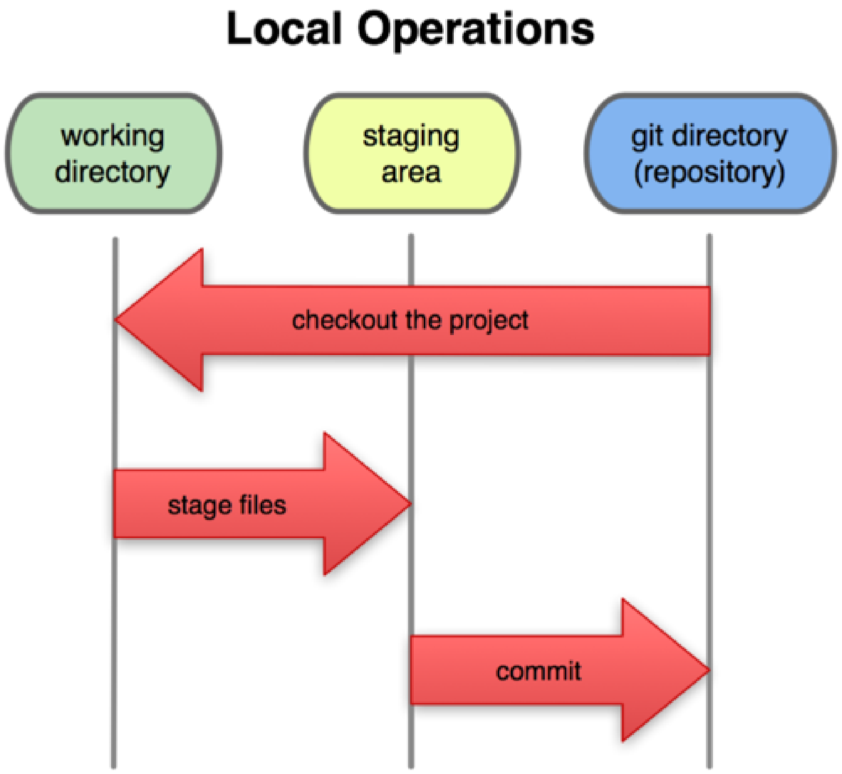
# Characteristics

## `git` is ...

- distributed
  - not dependent on network access or central servers
  - everyone can have a copy of the whole repository
  - everything can be done offline before `push`
- non-linear development
  - collaborative project
  - Linux kernel: averaging ~70 commits per day
- efficiency
  - check hash (SHA-1) and diff instead of comparing the whole contents
  - much like a *file system*
- very lightweight branch

## Getting started

## The three states



http://git-scm.com/book/en/Getting-Started-Git-Basics#The-Three-States

## First Steps

```
git init

git clone

git config
```

image src: http://www.healthtipsplace.com/baby.html

## First Steps

- create your own project

```
$ mkdir demo
$ cd demo
$ git init
```

- clone from other's repo

```
$ git clone git://git.kernel.org/pub/scm/git/git.git
# become a kernel hacker!
```

- configure your user info.

```
$ git config --global user.name "xatierlike Lee"
$ git config --global user.email xatierlike@gmail.com
```

http://git-scm.com/book/en/Getting-Started-First-Time-Git-Setup

## git init

```
$ mkdir demo

$ cd demo/

$ git init
Initialized empty Git repository in /Users/xatier/tmp/demo/.git/
```

- create needed files under the `.git` directory
- you can take a look inside `.git/` :)

http://git-scm.com/book/en/Git-Basics-Getting-a-Git-Repository

## git clone

- clone the whole repository from a url
- get a *local* copy of a Git repository

```
$ git clone git://git.kernel.org/pub/scm/git/git.git
Cloning into 'git'...
remote: Counting objects: 161444, done.
remote: Compressing objects: 100% (40617/40617), done.
remote: Total 161444 (delta 118857), reused 161444 (delta 118857)
Receiving objects: 100% (161444/161444), 38.58 MiB | 8.20 MiB/s, done.
Resolving deltas: 100% (118857/118857), done.
```

- different protocols

```
# local file
$ git clone file:///opt/git/project.git

# ssh
$ git clone ssh://user@server/project.git
$ git clone user@server:project.git

# http/s
$ git clone http://example.com/gitproject.git
```

http://git-scm.com/book/en/Git-on-the-Server-The-Protocols

# git config

- who you are & how to contact you
- you are responsible for your code!

```
$ git config --global user.name "xatierlike Lee"
$ git config --global user.email xatierlike@gmail.com
```

- `--global` configurations will be stored here

```
$ cat ~/.gitconfig
[user]
        name = xatierlike Lee
        email = xatierlike@gmail.com
```

http://git-scm.com/book/en/Customizing-Git-Git-Configuration

## Three useful commands

- show the working tree status

```
$ git status
```

- show changes between commits, commit and working tree, etc

```
$ git diff
```

- show commit logs

```
$ git log
```

### Use them often whenever you're not really sure what's going on!

## git status

- show the working tree status
- use this often!!!!
- after `git init`

```
$ git status
# On branch master
#
# Initial commit
#
nothing to commit (create/copy files and use "git add" to track)
```

- you have an empty *git repository* now, try to create some interesting stuffs!

[http://git-scm.com/book/en/Git-Basics-Recording-Changes-to-the-Repository](http://git-scm.com/book/en/Git-Basics-Recording-Changes-to-the-Repository)

## git status

- create an empty file

```
$ touch foo

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

- okay, now you have a file in your *repo*, but `git` doesn't know anything about that file yet
- tell `git` to *track* the content of the file

## git add

- add some changes to the *staging area*
- tell `git` to track the new added files

```
$ git add foo

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

- now you have a new *tracked* file in the *staging area*

# git diff

- Show changes between commits, commit and working tree, etc
- now, write some data to `foo`

```
$ echo "foo" > foo

$ git diff

diff --git a/foo b/foo
index e69de29..257cc56 100644
--- a/foo
+++ b/foo
@@ -0,0 +1 @@
+foo
(END)
```

Note: Is there any changes to the `.git` directory?

# git commit

- do the `commit`

```
$ git commit
```

- this command will open your `$EDITOR`

```
create the file 'foo'
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   foo
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   foo
#
```

# git commit

- a brief summary of your commit
    - file changed
    - insertions
    - deletions

```
$ git commit
[master (root-commit) 4c26688] create the file 'foo'
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 foo
```

- dark magic

```
git commit --amend
```

# git log

- take a look at your *commit log*

```
$ git log
```

- open a pager like the `less` command

```
commit 4c26688fc47fc120b262e7155356cb52796bc4bf
Author: xatierlike <xatierlike@gmail.com>
Date:   Sun Aug 18 17:35:46 2013 -0500

    create the file 'foo'
(END)
```

[http://git-scm.com/book/en/Git-Tools-Revision-Selection](http://git-scm.com/book/en/Git-Tools-Revision-Selection)

# git add

- add our changes to the *staging area*

```
$ git add foo
```

- `git diff` shows the differences between your *working directory* and the *staging area*

```
$ git diff
```

- you can check the changes you *staged*

```
$ git diff --staged
```

- now commit the staging area to the repo
- this will take a *snapshot* to the repository

```
$ git commit
[master eabdcbe] modified foo
 1 file changed, 1 insertion(+)
```

## git log

- look at the `git log` again

```
$ git log
```

- will show you

```
commit eabdcbe9999db6d1a58539e84cf4f32f6d841c7b
Author: xatierlike <xatierlike@gmail.com>
Date:   Sun Aug 18 17:51:02 2013 -0500

    modified foo

commit 4c26688fc47fc120b262e7155356cb52796bc4bf
Author: xatierlike <xatierlike@gmail.com>
Date:   Sun Aug 18 17:35:46 2013 -0500

    create the file 'foo'
```

## do `diff` often

```
$ git diff              (1)
$ git diff --staged     (2)
$ git diff --check      (3)
$ git diff HEAD         (4)
```

1. Changes in the working tree not yet staged for the next commit.
2. Changes between the index and your last commit.
3. Warn if changes introduce whitespace errors.
4. Changes in the working tree since your last commit.

### make sure to check everything before you do a `git commit` !!!

## file status lifecycle

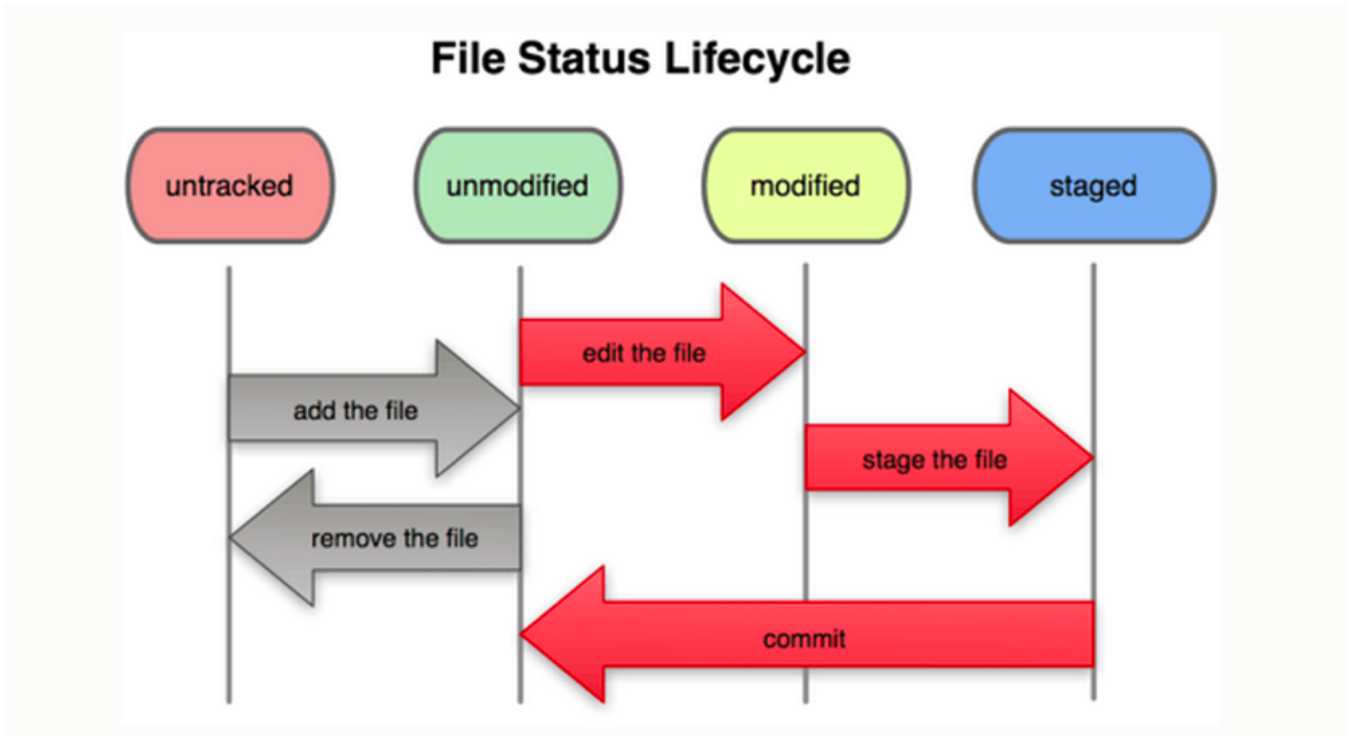


image src: [http://git-scm.com/book/en/Git-Basics-Recording-Changes-to-the-Repository](http://git-scm.com/book/en/Git-Basics-Recording-Changes-to-the-Repository)

## Summary

- `commit` only keeps your changes in the *staging area*
    - that is, the changes in the working directory will be recorded only after you `add`ed them
- use `git diff` to double check the *staging area* before you `add` & `commit`
    - don't act like a dummy in your commit messages :P
- keep modifications small and clear between commits
- rules to write good commit messages
    - a brief summary at the first line
    - describe the implementation details or your algorithm after that
    - write commit messages in English

## Okay, this is the very basic usage of git :)

**Time to take a break!**

## git add/rm/mv

- `git add`
  - before your commit
  - add files to your *staging area*
  - add new modifications to your *staging area*
- `git rm` (be careful!!!!)
  - remove entries from the *staging area*
    - By default, a `git rm file` will remove the file from the *staging area* entirely and also off your disk
  - `git rm --cached` leave the file in the working directory (i.e. untrack the file)
- `git mv`
  - git does not track file by name (git is content-based, remember that?)
  - `git mv` = `git rm --cached` + `git add`

# git commit

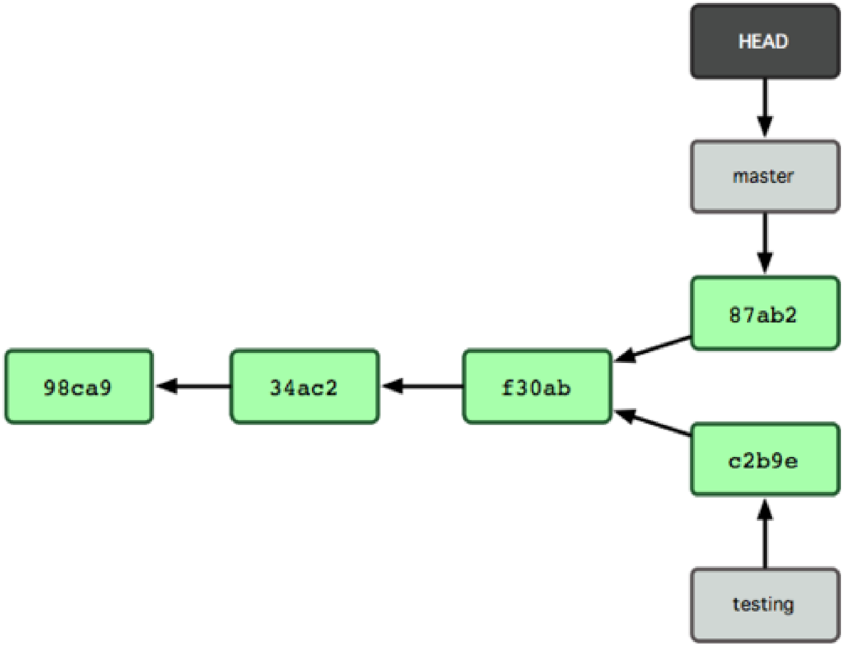
- make snapshots of your staging area
  - after you committed, the working directory will be *clean*
  - that is, we have not made any changes since our last commit
- record your name and email
  - `git config` and `~/.gitconfig`
- commit messages
  - it's very important to write a good commit message
  - useful for code review and trace the history
  - *what*, *how* and *why* in this commit

Note: commit logs from last night

## .gitignore

- in your repo directory
    - `./.gitignore`
- things you don't want to track
    - log files, tmp files, build files (change frequently)
    - auto generated files
    - database passwords
- `git ls-files` and `git status`
- of course, you need to add this file to the repo
- some good examples here

## branch



## branch

- one of the most powerful features of git
- keep multiple snapshots of the repo
- topic branch
  - hot-fix
  - bugfix
  - feature

```
git branch
git checkout
git merge
```

http://git-scm.com/book/en/Git-Branching-What-a-Branch-Is

## git branch

- list (local) branches

```
$ git branch
 * master
```

- create a new branch

```
$ git branch testing

$ git branch
 * master
   testing
```

- delete a branch

```
$ git branch -d testing
```

## git checkout

- タイムマシン
- restore from any snapshots
- switch between your branches

image src: Steins; Gate

## git checkout

- switch between your branches

```
$ git branch
 * master
   testing


$ git checkout testing
Switched to branch 'testing'


$ git branch
   master
 * testing


$ git checkout master
Switched to branch 'master'
```

- shortcut: create & checkout to it

```
$ git checkout -b blah
Switched to a new branch 'blah'
```

## git merge

- merge a branch into your current branch (`HEAD`)

```
$ git merge testing
blah blah...
x files changed, y insertions(+), z deletions(-)
```

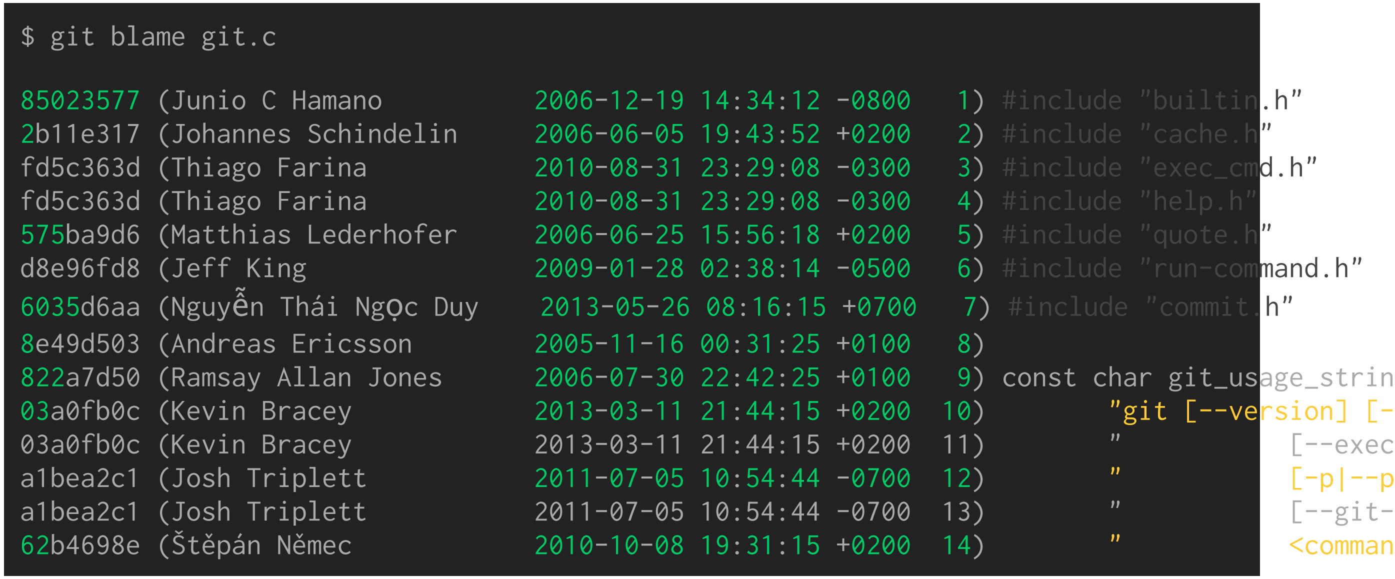
- conflict!
  - different commits modifying on the same parts of code
  - fix it, then `add` and `commit`

```
$ cat README
<<<<<<< HEAD
Many Hello World Examples
=======
Hello World Lang Examples
>>>>>>> fix_readme
```

- three way merge
  - find the common history

# git blame

- last modification of each line
- very useful for history tracing or when something is broken
- ~~抓戰犯~~



```
$ git blame git.c

85023577 (Junio C Hamano          2006-12-19 14:34:12 -0800   1) #include "builtin.h"
2b11e317 (Johannes Schindelin     2006-06-05 19:43:52 +0200   2) #include "cache.h"
fd5c363d (Thiago Farina           2010-08-31 23:29:08 -0300   3) #include "exec_cmd.h"
fd5c363d (Thiago Farina           2010-08-31 23:29:08 -0300   4) #include "help.h"
575ba9d6 (Matthias Lederhofer     2006-06-25 15:56:18 +0200   5) #include "quote.h"
d8e96fd8 (Jeff King               2009-01-28 02:38:14 -0500   6) #include "run-command.h"
6035d6aa (Nguyễn Thái Ngọc Duy    2013-05-26 08:16:15 +0700   7) #include "commit.h"
8e49d503 (Andreas Ericsson        2005-11-16 00:31:25 +0100   8)
822a7d50 (Ramsay Allan Jones      2006-07-30 22:42:25 +0100   9) const char git_usage_strin
03a0fb0c (Kevin Bracey            2013-03-11 21:44:15 +0200  10)       "git [--version] [-
03a0fb0c (Kevin Bracey            2013-03-11 21:44:15 +0200  11)       "             [--exec
a1bea2c1 (Josh Triplett           2011-07-05 10:54:44 -0700  12)       "             [-p|--p
a1bea2c1 (Josh Triplett           2011-07-05 10:54:44 -0700  13)       "             [--git-
62b4698e (Štěpán Němec            2010-10-08 19:31:15 +0200  14)       "             <comman
```

## git shortlog

- who is the most active committer?

```
$ git shortlog -nse

12975  Junio C Hamano <gitster@pobox.com>
 1398  Shawn O. Pearce <spearce@spearce.org>
 1208  Jeff King <peff@peff.net>
 1106  Linus Torvalds <torvalds@linux-foundation.org>
  731  Johannes Schindelin <johannes.schindelin@gmx.de>
  679  Jonathan Nieder <jrnieder@gmail.com>
  529  Nguyễn Thái Ngọc Duy <pclouds@gmail.com>
  511  Jakub Narębski <jnareb@gmail.com>
  470  Eric Wong <normalperson@yhbt.net>
  400  René Scharfe <l.s.r@web.de>
  369  Johannes Sixt <j6t@kdbg.org>
  344  Nicolas Pitre <nico@fluxnic.net>
  339  Michael Haggerty <mhagger@alum.mit.edu>
  336  Felipe Contreras <felipe.contreras@gmail.com>
```

# git show

Show various types of objects

```
git show <commit>
git show <tag>
```

## git reflog

Show the "reference" of each commit

```
git reflog
```

## git whatchanged

Show logs with differences that each commit introduces

```
git whatchanged
```

## git stash

Stash away the changes in a dirty working directory

```
git stash
git stash save "message"
git stash pop
git stash list
git stash show -p <reversion>
git stash apply
git stash drop
```

## git tag

Give a name to a commit

```
git tag -a v1.0
```

# git reset

- Reset current HEAD to the specified state
- DANGEROUS !!!
- two modes
  - `--mixed` Resets the index but not the working tree (default)
  - `--hard` Resets the index and working tree. Any changes to tracked files in the working tree since are discarded.

```
git reset <reversion>
git reset --hard HEAD~3
git reset <file>
git reset --hard <file>
```

- must run `git diff HEAD` before you use `git reset`

## code with your friends!

You'll never code alone

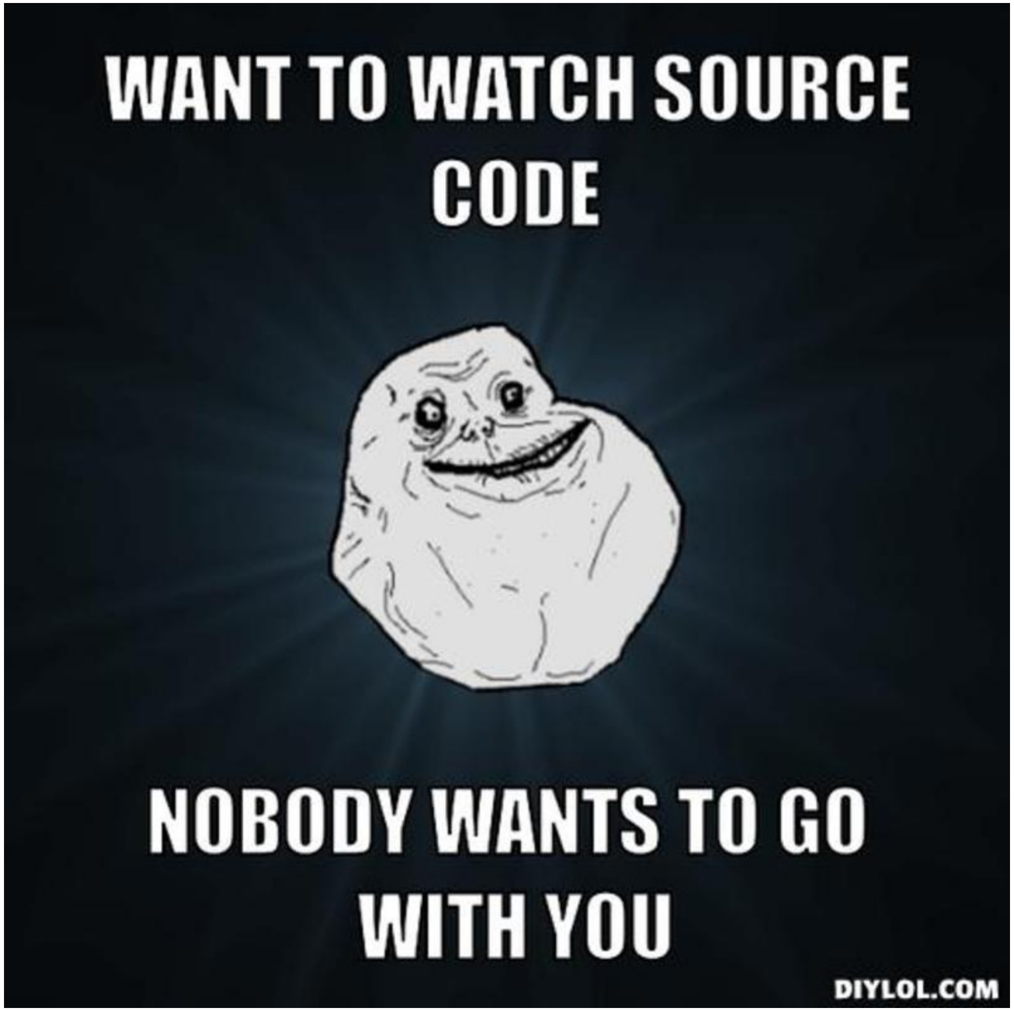


image src: http://diylol.com/meme-generator/40-year-old-college-student/memes/want-to-watch-source-code-nobody-wants-to-go-with-you

## distributed version control

- project development with 100+ of people?
- everyone has his/her own copy(s) of the remote repository
- do most stuffs off-line (write some codes on airplane!)
- you can do any experiments on your own local repository
- `remote` / `fetch` / `push`
  - manage your remote repositories with `git remote`
  - update your project with `git fetch`
  - share your changes with `git push`

http://git-scm.com/book/en/Distributed-Git-Distributed-Workflows

## git remote

- list remotes

```
$ git remote
origin

$ git remote -v
origin  git@github.com:xatier/git_intro.git (fetch)
origin  git@github.com:xatier/git_intro.git (push)
```

- add / remove / rename

```
$ git remote add linux-nfs git://linux-nfs.org/pub/linux/nfs-2.6.git
$ git remote rm linux-nfs
$ git remote rename linux-nfs linux-nfs-hack
```

- remote branches

```
$ git branch -a
   gh-pages
 * master
   remotes/origin/HEAD -> origin/master
   remotes/origin/gh-pages
   remotes/origin/master
```

# git remote

- secrets here

```
$ cat .git/config

(omitted...)

[remote "origin"]
    url = git@github.com:xatier/git_intro.git
    fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
    remote = origin
    merge = refs/heads/master
[branch "gh-pages"]
    remote = origin
    merge = refs/heads/gh-pages
```

- remote show

```
git remote show
git remote show origin
```

## fetch, pull and push

- `git fetch`
    - download new branches and data from a remote repository
    - synchronize with remote repo
- `git pull`
    - fetch from a remote repo and *try to merge* into the current branch
    - `git pull` = `git fetch` + `git merge`
- `git push`
    - push your (new) branches to a remote repository

http://git-scm.com/book/en/Git-Basics-Working-with-Remotes

## The *Merge* workflow

- `git push` copies your branches to a remote repository
- `git fetch` copies remote branches to your repository
- `git pull` does fetch and merge in one go

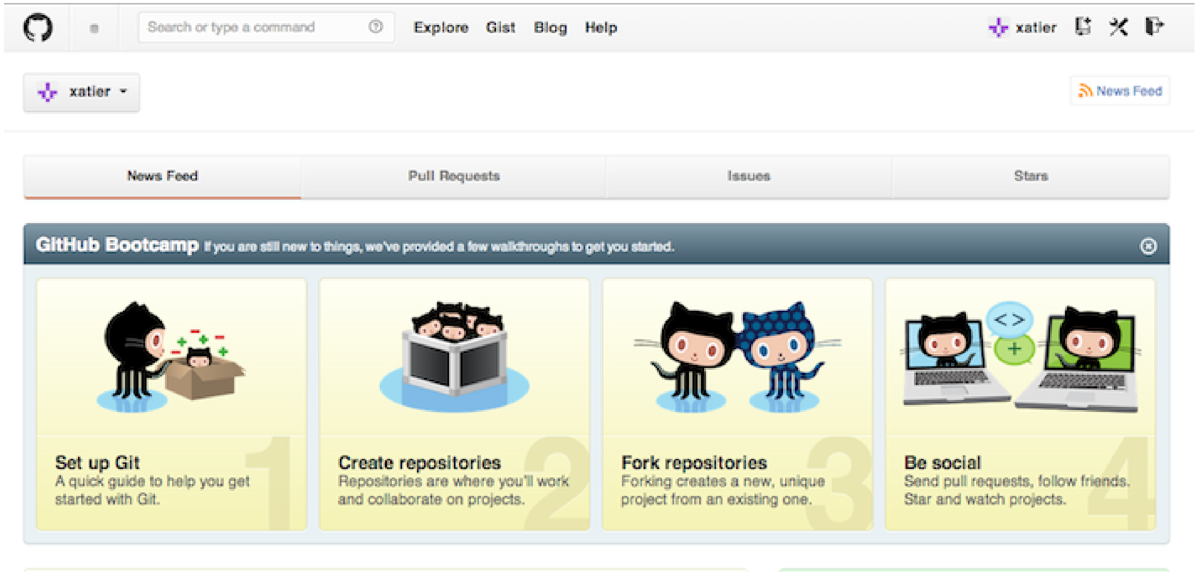
## Do not

- use `git push` unless you actually want to share the local branch
- use `git pull` unless you actually want to merge the remote branch

```
git fetch <remote>
git fetch <remote> <branch>
git pull <url> <branch>
git push <remote> <branch>
git push <remote> <local_branch_name>:<remote_branch_name>
```

man gitworkflows

## github.com



GitHub Bootcamp If you are still new to things, we've provided a few walkthroughs to get you started.

Set up Git
A quick guide to help you get started with Git.

Create repositories
Repositories are where you'll work and collaborate on projects.

Fork repositories
Forking creates a new, unique project from an existing one.

Be social
Send pull requests, follow friends. Star and watch projects.

# github.com

- social coding
- the most popular open source code repository hosting service
- contribute to open source projects
- get jobs from your commit logs!

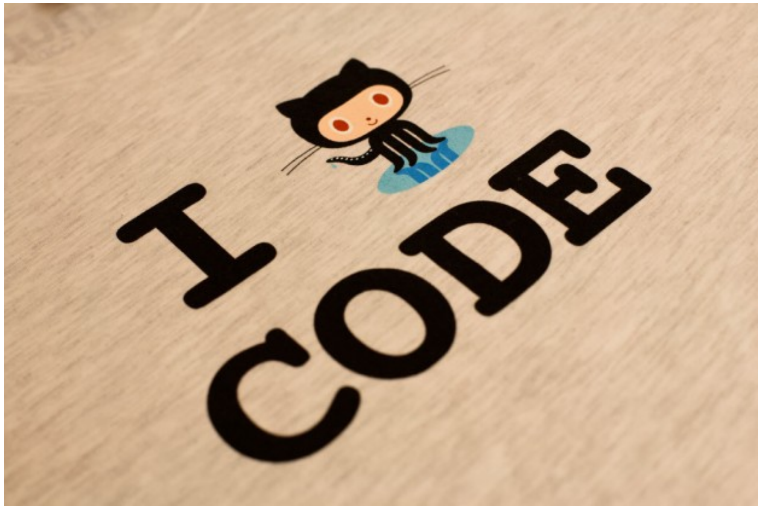


image src: https://svpply.com/item/425245/GitHub__I_octocat_Code

# README.md

- a introduction of this repository
- (github flavored) Markdown
- http://markdown.tw/
- https://github-markdown-preview.heroku.com

https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

## git help

```
$ git help tutorial
```

**Thanks**

## Reference

- http://gitscm.com/book
- http://gitref.org/
- https://www.kernel.org/pub/software/scm/git/docs/
- https://www.atlassian.com/git/tutorials/
- http://www-cs-students.stanford.edu/~blynn/gitmagic/
- http://jonas.nitro.dk/git/quick-reference.html