An introduction to (version control with)

Tom Wiesing

CS Club

September 5, 2015

Overview

- Introduction
    - What is version control?
    - What is git? Why use it?
- Getting started with git
    - How does git store versions?
    - How do I use clone, init, add, commit?
    - How do I share content?
- Going further
    - Branching & Merging
    - Working with Remotes
    - Stashing & other useful commands
- A brief introduction to **GitHub**
    - Collaborating with other people
    - Making issues
    - Forking repositories and making pull requests

Introduction (1): What is version control?

- ▶ tracks any kind of content
  - ▶ e.g. websites, software, presentations
- ▶ knows about different versions
  - ▶ knows what was changed when
  - ▶ can revert changes if something goes wrong
- ▶ has a collaboration component
  - ▶ several people can work together on the same project
  - ▶ changes can be synced
  - ▶ easy to see who changed what

WIKIPEDIA
The Free Encyclopedia

Article Talk

Version control

From Wikipedia, the free encyclopedia

A component of software configuration management, **version control**, also known as **revision control** or **source control**,[1][2] is the management of changes to documents, , large web sites, and other collections of information.

Introduction (2): What is git and why use it?

- git – "the stupid content tracker"
  - open-source version control system
  - fast, scalable, distributable
- originally developed in 2005 for maintaining the linux kernel source code

Introduction (3): What is git and why use it?

- git is both for beginners and advanced users
  - provides high-level-commands
  - additonally gives full access to internals
- git is distributed - and it is easy to sync changes
  - no central server to share content required
  - changes can be synced in many ways

   http(s), ssh, git protocol, diffs via email, . . .

Getting started (1)

- Keep in mind: This talk is only an introduction - there is more
- basic interaction with ◆ **git** happens via the command line
  - GUIs exist, but it is best to learn git from the command line
  - Web-Frontends are widely used (we will talk about **GitHub** later)
- git stores information for a single project in a *git repository*
  - commonly found on your hard disk in form of a folder
  - *clones* of the repository can be made in order to share it

Getting started (2): Creating a repository

- ▶ you can create a new repository in a folder by using `git init`
- ▶ alternatively you can clone an existing repository with `git clone repository-url`
- ▶ creates a *working directory* where the current version is *checked out*
- ▶ different versions are tracked with so-called *commit*s
    - ▶ has a title and some information when and by whom it was made
    - ▶ stores a reference to the previous commit (version)
        - ▶ not for the initial commit of course
    - ▶ stores the changes were made since that version

Getting started (3): Working Directory, Staging Area & History

- commits are local to your clone - they are not automatically shared
- Making a commit
  - first make changes in your working directory (also called the index)
  - then add the files you want to commit to the staging area
  - finally you commit the changes in the staging area

Local Operations

working directory — staging area — git directory (repository)

checkout the project

stage files
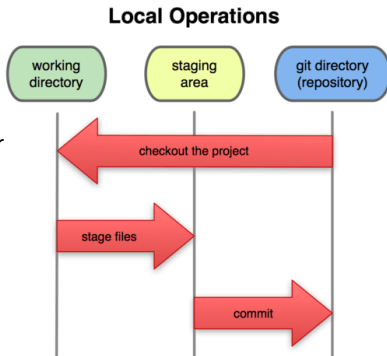
commit

Getting started (4): Commands for creating commits

- Commands for creating commits
  - `git status` - to see what is changed and what is in the staging area
  - `git add FILES` - to add files to the staging area
    - use `git add -A .` to add everything
  - `git rm FILES` - to delete a file and add that change to the staging area
  - `git commit -m MESSAGE` - to create a commit with the given message from the staging area
  - `git checkout FILE` - to reset FILE to the last commit
- Time for a short demo
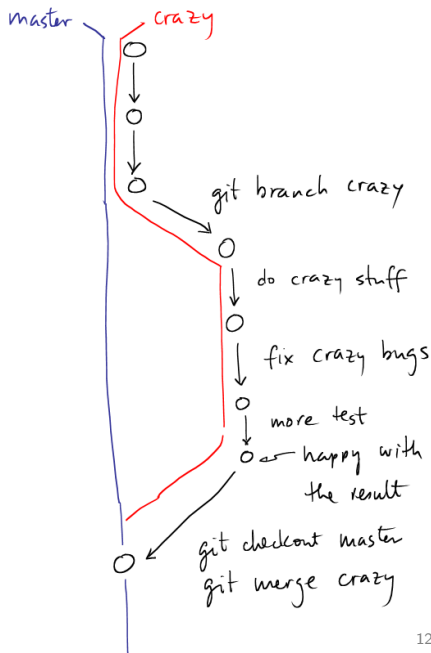
Getting started (5): Sharing commits

- ▶ Creating commits is great, but how to share them?
- ▶ git uses so called "remotes"
  - ▶ a remote is just a different clone of the same repository somewhere else
- ▶ commits can be pushed to it – using `git push`
- ▶ commits can be pulled from it – using `git pull`
- ▶ internally it is a bit more complicated than that - we need to talk about branching first

Going further (1): Branches

- several people can work at the same project at the same time
- they might make incompatible changes (that can be united later)
- git allows the versions of a repository to diverge using branches
    - they can also be brought back together using merging later
- The default branch is usually called "master"
- Each branch has a so-called HEAD that points to its latest commit
- There is also a HEAD of the repository which points to the current branch

Going further (2): Branching & Merging

- you can make commands on branches like you would normally
  - but you need to switch to the branch first
  - `git branch name` - create a branch
  - `git checkout name` - switch to it
  - `git merge name` - merge a branch back into the current one

master · crazy

git branch crazy

do crazy stuff

fix crazy bugs

more test

happy with the result

git checkout master
git merge crazy

Going further (3): Resolving Merge conflicts

- ▶ merging can cause conflicts
  - ▶ when files were modified on both branches and git can not merge them automatically
- ▶ git will tell you when you run `git merge` if there are conflicts
  - ▶ you can edit the affected files manually, then stage the files (`git add`) and commit them (`git commit`)
    - ▶ `git status` is always helpful when doing this
  - ▶ `git merge --abort` - cancel the merge and go back to what was there before
  - ▶ "fake" a merge by forcing git to use one of the two versions
    - ▶ `git merge -X ours branch` - use the version of the current branch
    - ▶ `git merge -X theirs branch` - use the other branches version
    - ▶ (you need to do this before starting to merge)