An introduction to (version control with) git

Tom Wiesing

CS Club

September 5, 2015

Overview

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

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)

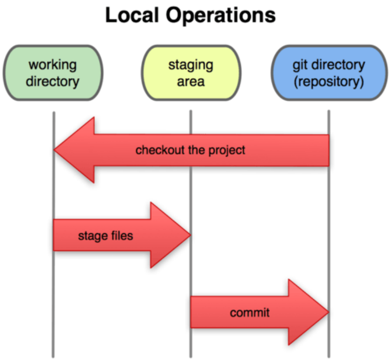
- Keep in mind: This talk is only an introduction - there is more
- basic interaction with git happens via the command line
  - GUIs exist, but it is best to learn git from the command line
  - Web-Frontends are widely used (we will talk about GitHub later)
- git stores information for a single project 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



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 .` exi 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

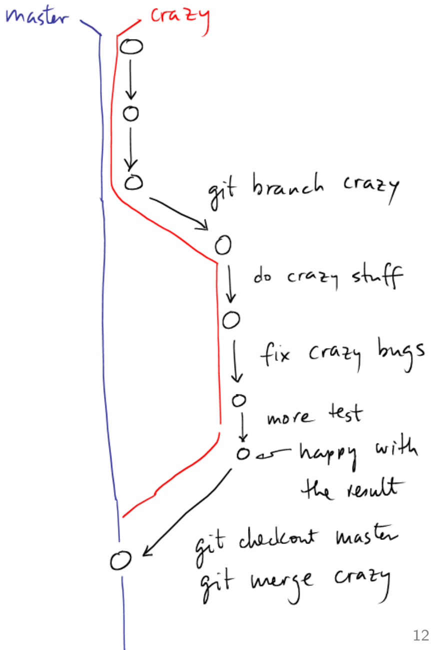
- Creating commits is great, but how to share them?
- git uses so called "remotes"
  - a remote is just a different clone of the same repository somewhere else
- commits can be pushed to it – using `git push`
- commits can be pulled from it – using `git pull`
- internally it is a bit more complicated 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 commits on branches like you would normally
  - but you need to switch to them first
  - `git branch name` - create a branch
  - `git checkout name` - switch to it
  - `git merge name` - merge a branch back into the current one
- Time for another short demo



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)

Going further (4): Working with remotes

- a remote is a clone of the same repository in another location
  - usually on a remote server
- you can add and remove remotes dynamically
  - e.g. `git remote add name url`
- when you `git clone` a repository a remote “origin” will be added automatically
- remotes have branches and you can push and pull your local branches
  - `git push remote branch` - pushes the *branch* to the remote *remote*
    - if you use `git push -u` you can just you can omit the names afterwards
  - `git fetch remote/branch` - fetches new commits from the remote branch only.
    - It can happen that only a rebase takes place
  - `git pull` - fetches new commits from the tracked branch and merges them into the local branch

Going further (5): Common Practices for Pushing & Pulling

- if you clone a repository, you usually only need `git push` and `git pull`
- before pushing new commits you need to pull first
  - there is `git push --force` but you **never** want to do this because this can lead to loss of data.
- when pulling new commits, merge conflicts might occur
  - use `git fetch remote/branch` and then `git merge remote/branch` to resolve conflicts

Going further (5): Common Practices for Pushing & Pulling

- if you clone a repository, you usually only need `git push` and `git pull`
- before pushing new commits you need to pull first
  - there is `git push --force` but you **never** want to do this because this can lead to loss of data.
- when pulling new commits, merge conflicts might occur
  - use `git fetch remote/branch` and then `git merge remote/branch` to resolve conflicts

Going further (6): Other useful commands

- `git diff` - to see unstaged changes inside files
- `git log` - show a log of recent commits on the current branch
  - exit by pressing *q*
- `git commit --amend` - Edit the previous commit instead of creating a new one.
- `git reset HEAD files` - removes changes from the staging area
- `git stash` - Stash your changes for a rainy day
- `git tag` - give names to certain commits
- ...

A brief introduction to **GitHub** (1)

- **GitHub** , `https://github.com` is a website that allows people to share and collaborate on git repositories online
  - open source alternatives also exist, for example GitLab.
- offers users an unlimited number of public repositories to collaborate on
- provides a Web Interface & Online editor for most of gits features
- has a few additional features in addition to repositories

A brief introduction to GitHub (2): Issue Tracking & Milestones

- Issues can be used to track bugs, todos and ideas for your project
- on github, anyone can comment on them
- people that have access to your repository can mark them as done
  - This can even be done from within commit messages
- you can use milestones to track your overall progress

A brief introduction to GitHub (3): Forking & Pull Requests

- sometimes you want to suggest specific changes in the code to a repository
- if you have access to the repository, you can just commit directly
- in other cases, you can “fork” the repository
    - This makes a clone of the repository to your github account
- you can then make changes in your fork and issue a pull request
- the owner of the original repository can then merge the changes back in
- time for a final demo

The end

Thank you for your attention!
Any Questions, Comments, etc?

- Image Sources:
  - `https://git-scm.com/images/logos/downloads/Git-Logo-2Color.png`
  - `https://git-scm.com/figures/18333fig0106-tn.png`
  - `https://assets-cdn.github.com/images/modules/logos_page/GitHub-Logo.png`
  - `http://www.cs.toronto.edu/~kenpu/articles/cs/git-intro/ex6.png`
  - `https://en.wikipedia.org/wiki/Version_control`
  - `http://cdn.meme.am/instances/500x/55168121.jpg`
  - `https://assets-cdn.github.com/images/modules/logos_page/Octocat.png`