Intro to Git

Netsoc

Goals

- Introduce Git without assuming prior knowledge
- Show you the 10% of Git that can do 90% of the work
- Slides & practical demonstration
- Questions & Answers (whenever you want)

Git

- Version Control System (VCS)
  - Keep track of changes in files over time
  - Track who changed what, when
  - Help multiple people work on the same files at the same time
  - Detect and sometimes resolve conflicts automatically
- Created by Linus Torvalds in 2005 for Linux Kernel development
- Command-line program, available on Linux, Mac and Windows https://git-scm.com/
- Free and Open-Source
- Once installed, just type "git" to get a list of useful commands

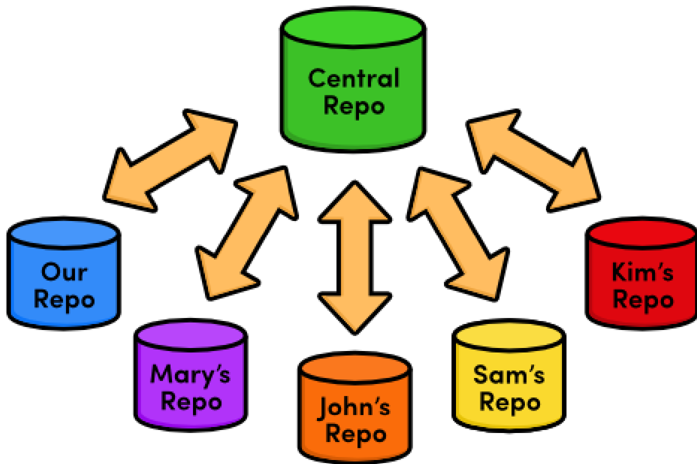


Figure: http://rypress.com/tutorials/git/media/8-7.png

Github

- A website for hosting git repositories
- It's free
- The user interface is great
- Bug tracker, search function, can "star" and follow repositories

Overview | Repositories 8 | Stars 62 | Followers 14 | Following 14

Sam Caulfield
samcaulfield

Add a bio

Dublin, Ireland
sam@samcaulfield.com

Organizations

Popular repositories — Customize your pinned repositories

GameEngine
C ★ 2

OpenGL
Sample OpenGL programs.
C ★ 1

libsc
A library of data structures and algorithms in C.
C ★ 1

Rival
A turn-based strategy game.
C ★ 1

memcheat
A tool for messing with the memory of other processes
C ★ 1

git-tutorial
Slides for a Netsoc Git Tutorial
TeX

687 contributions in the last year — Contribution settings

Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Jan
Mon
Wed
Fri

How Git works

- Git lives inside the .git folder in the project folder
- This is where git keeps track of things for each project
- Project developers don't need to touch this folder, it's for git only
- Project developers only care about the human-readable files that we create

Adding things to your repository

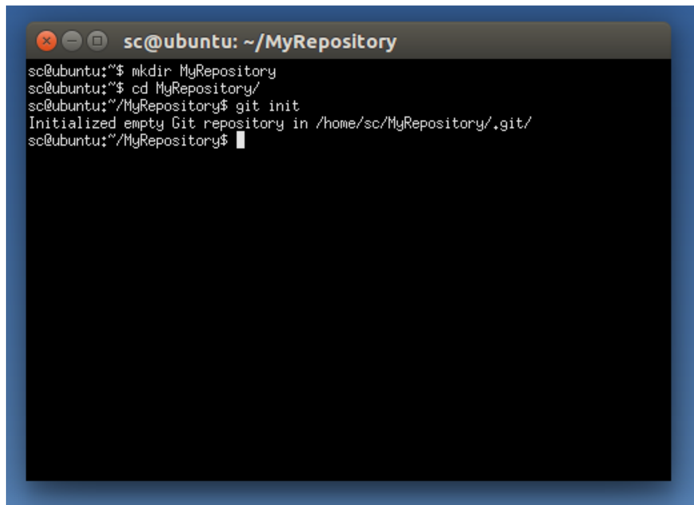
- When you create or delete files, git doesn't automatically add or delete them from the repository
- When you modify files, git doesn't automatically update the repository with the changes
- You can make all the changes you want, and then turn to git when you're ready to commit them
- This is good because it gives you time to think things through / clean up before uploading

Adding things to your repository

- With Git, modifying your respository is manual
- When you make changes, they only effect your copy of the repository
- Changes are called "commits"
- Then when you're ready, you upload the commits so your teammates can get them, and vice-versa

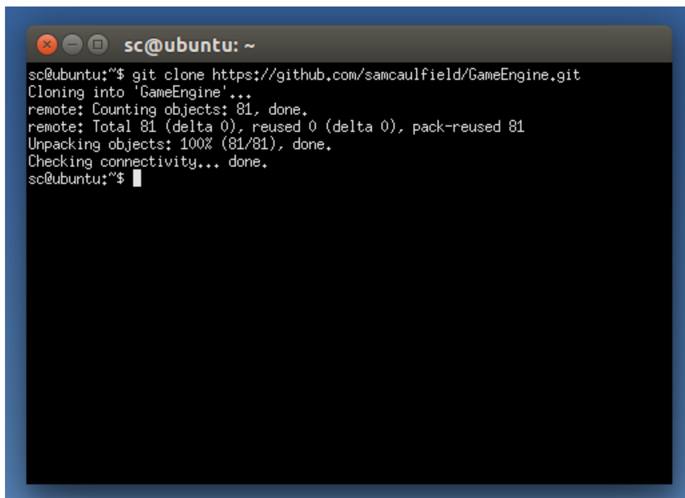
Getting a repository

- Need to create or get a copy of the repository to begin working
- to create: git init



Getting a repository

- If you're not starting from scratch, need to get it from somewhere
- to copy from somewhere else: git clone



Adding things to your repository

- Git has the concept of a "staging area"
- The purpose of the staging area is to gather up a set of changes to make a commit out of
- By default changes to files are unstaged
- You manually add files to the staging area
- Then you can confirm the changes (commit)

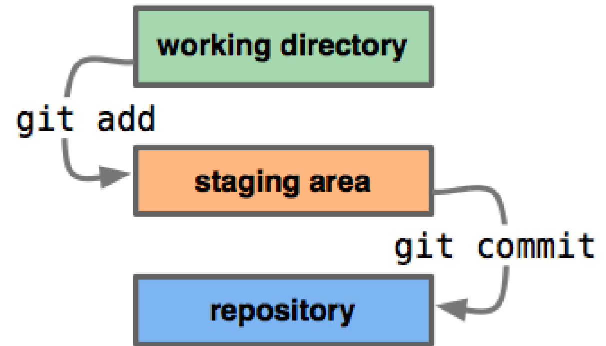


Figure:
http://codingdomain.com/git/partial-commits/git-staging-area.png

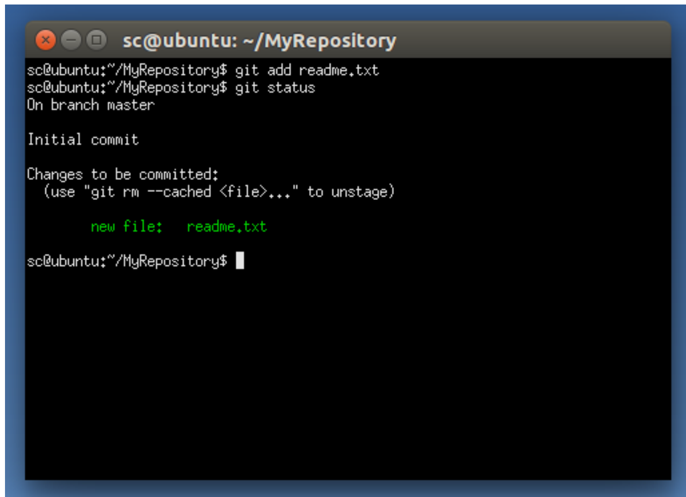
sc@ubuntu: ~/MyRepository

```
sc@ubuntu:~/MyRepository$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
sc@ubuntu:~/MyRepository$
```

sc@ubuntu: ~/MyRepository

```
sc@ubuntu:~/MyRepository$ touch readme.txt
sc@ubuntu:~/MyRepository$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        readme.txt

nothing added to commit but untracked files present (use "git add" to track)
sc@ubuntu:~/MyRepository$
```

- Use the "git add" command (same as "git stage")
- You can use "git add –patch" to add pieces of files



Staging files

- You can add multiple files with wildcard (git add *)
- This often goes wrong
- .exe .bin .o .class .so etc.

.gitignore

- .gitignore file can mitigate this
- .gitignore is a file you put in the repository
- prevents people adding files that shouldn't go in the repository
- one file type per line
- git status won't show the files and they won't be used in wildcards

Tracking files

- Git tracks exactly what has changed in each file since the last "commit"
- You can view unstaged changes with "git diff"
- You can view staged changes with "git diff –staged"
- You can view all changes with "git diff HEAD"

Unstaging changes

- You can unstage changes
- Do this if you aren't ready to commit a change after all
- This just takes the change out of the staging area, it doesn't delete it
- Remember staging/unstaging != creating/deleting
- The staging area is just showing Git your intent for the files
- "git reset HEAD file"

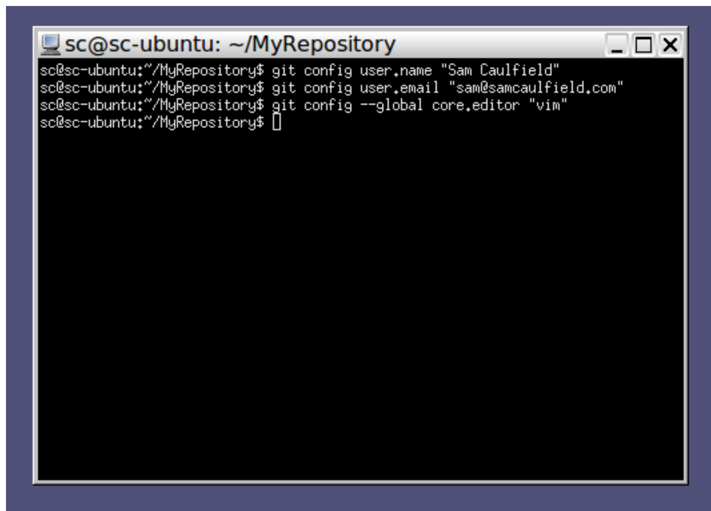
Committing changes

- Staging/unstaging doesn't make permanent changes to the repository
- That's what committing is for
- Commits "lock in" staged changes
- When you commit, the staged changes are converted to a commit
- Git stores what each commit changed in it's .git folder
- Before you make your first commit, you'll need to give Git some information

Git Config

- Commits are summarised in a log
- The log is a good record of the project's development history
- All commits can be traced back to an author (name, date and email)
- To tell Git how to "sign" your commits, use "git config"

Git Config



Committing Changes

- Now that the config is set up, you can commit
- Use the "git commit" command
- This gathers up the staged changes into a commit
- You have to enter a commit message (for the log)

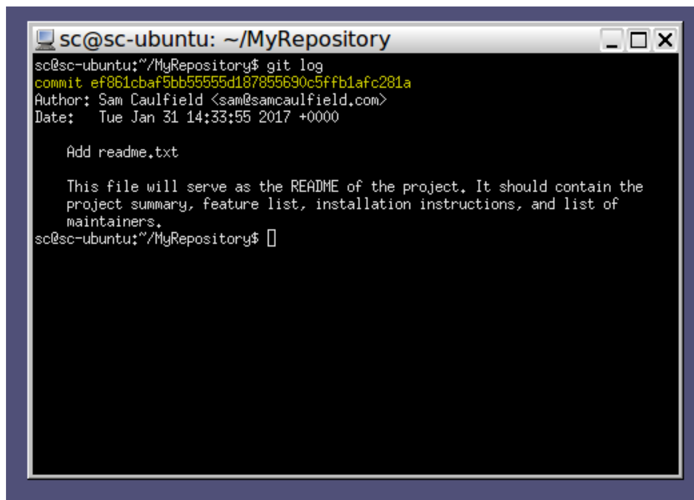
Committing

sc@sc-ubuntu: ~/MyRepository

```
 1 Add readme.txt
 2
 3 This file will serve as the README of the project. It should contain the
 4 project summary, feature list, installation instructions, and list of
 5 maintainers.
 6 # Please enter the commit message for your changes. Lines starting
 7 # with '#' will be ignored, and an empty message aborts the commit.
 8 # On branch master
 9 #
10 # Initial commit
11 #
12 # Changes to be committed:
13 #       new file:   readme.txt
14 #
~
~
~
~
~
~
~
~
~
                                                        5,12          All
```

Git Log

- The "git log" tool is useful for showing a summary of commits
- Remember the commits are the history
- So a history of the commits is the history of the project



Committing Tips

- Small and simple changes with good descriptions
- Do one thing well
- Should bring the repository from one consistent state to another
- Be polished, i.e. code, tests and documentation where reasonable
- Solving a problem often comes down to thinking what series of commits will it take to solve

Commit Messages

- The commits are the project history
- So the commit messages are the more human-friendly project history
- Explain why a change was made
- Developer-to-developer communication

Commit Messages

- Title & body
- Personal preference & project convention
- The commit message "header" is generally kept to 50 chars or less
- The header generally uses the imperative tense
- "Add tests to maths library", "Fix bug in save dialog" etc.
- The commit message body can be as long as you want
- But is generally kept to 73 chars or less per line
- http://chris.beams.io/posts/git-commit/

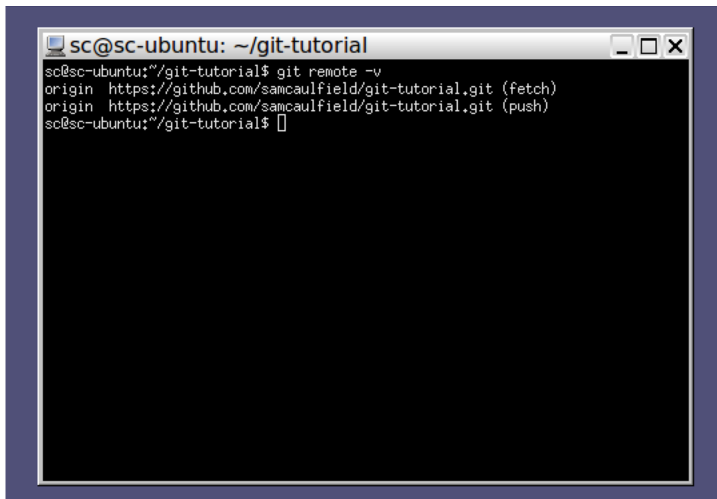
Syncing your changes

- Up to now we have only been modifying our own copy of the repository
- To share the changes with your team you need to synchronise
- This is done by downloading their changes "pulling"
- and uploading your changes "pushing"
- The "git pull" and "git push" commands do this

Syncing your changes - remotes

- To sync, you have to know where to pull/push changes from/to
- In Git, servers that host repositories are called "remotes"
- You can easily list the servers for your project with "git remote -v"
- If you cloned a repository from Github, the remote is automatically set up for you

Git remotes



Pulling changes

- You want to start the sync by pulling the new version of the repository
- This is done with "git pull"
- Generally speaking, you want to use "git pull –rebase"
- git pull –rebase keeps the log history cleaner because it often doesn't produce a "merge commit"
- Sometimes when you pull, conflicts can occur

Pushing changes

- Once you've got up to date with the remote repository you'll want to push your changes
- This is so your team members can access them
- This is as simple as "git push"

Merge Conflicts

- Sometimes two people have modified the same part of the same file
- This is a problem for the developers to solve, not Git
- Git just highlights the conflict lines in the file and lets you solve it
- This involves opening the file with the conflict in an editor and fixing it yourself
- You then commit the fixed file
- https://help.github.com/articles/resolving-a-merge-conflict-using-the-command-line/

Merge Conflicts

- Merge conflicts can often be avoided by better project management
- Two people shouldn't really be modifying the same bits of the same file too often
- Can have the concept of "file owners"
- One person or team responsible for changing particular files

Branches

- A branch is like a private copy of the repository
- Each person can have multiple branches, and can share the branches with others
- Every project has at least one branch, just one is fine though
- One branch is usually considered the One True Branch (called "master")
- The master branch usually has the latest stable version of the project

Branches

- Not really worth worrying about just starting out
- Can be quite useful for large projects
- Can create a branch to do a feature, and it hides the messy commits away
- General use case is: create branch, implement feature, merge branch into master and delete branch

Branches

- "git branch branchname" - create a branch
- "git branch -d branchname" - delete a branch
- "git checkout branchname" - switch to that branch
- Before we were doing everything on the master branch
- Same commands apply to all branches

Forks

- Different to branches
- Not a concept that the git program deals with
- A fork is a term for when a whole project is copied and worked on by someone else
- e.g. Ubuntu started as a fork of Debian
- It's more of a people term than a technical term

Pull request

- A git term where someone makes a contribution to a project that they don't have push access on
- A project maintainer has to manually approve the change and pull it
- A good way to accept contributions from the public
- Github has a nice user interface for this

Tags

- In git you can "tag" certain commits
- This can be to tag release versions
- You can tag the current commit with ‘git tag -a v1.0 -m "beta release"‘
- This can make it easier to keep track of stable versions of the project