Understanding Git's Architecture

- Ingram Monk
- Local Web developer, for small family business advertising properties for sale in France
- Have been using Git for 3-4 years
- Have yet to start using it well!
- Have realised that to use it well, I need to understand how it works, not just how to use it
- Please interject with your questions – no matter how silly you think they may be – I can guarantee I have already asked myself them!

2015-06-01-rev1

Contents

Why use Git?

- Have you ever edited a file and then afterwards wished you had not made the changes, only to be too late to undo them all?

- Would you like to be able to share your work with others and keep track of who did what?

- Would you like to be able to make different versions of the same file to test different ideas?

- Would you like to be able to back up your work without needing to remember to use a separate backup system?

What is Git?

- A Distributed Version Control System (DVCS)

What is Version Control?

- A logical way to organise and control revisions to documents
- A safe mechanism for users to collaborate on the same work without overwriting each other's changes
- A simple way to revert to previous versions of documents, in cases where you have messed up!

What does Distributed mean?

- Everybody has a working copy of the codebase on their local machine, as opposed to a reference copy of the codebase
- No requirement to share a common network
- No central server / repository
- I.e. decentralized and peer-to-peer

Why is Distributed a good approach?

- No need for network access to central server – good for remote working (conversely; don't lose your laptop!)

- Each user has a complete repository on their machine. Allows them to do private work without treading on each other's toes and without the need to seek permission to make changes – a faster way to work

- No single point of failure, as with a central server. Likewise each copy of the repository functions as a remote backup of codebase

- Quicker operations, as no network latency

- Allows simple forking of projects for the inevitable design disagreements inherent in FOSS software!

Git's architecture

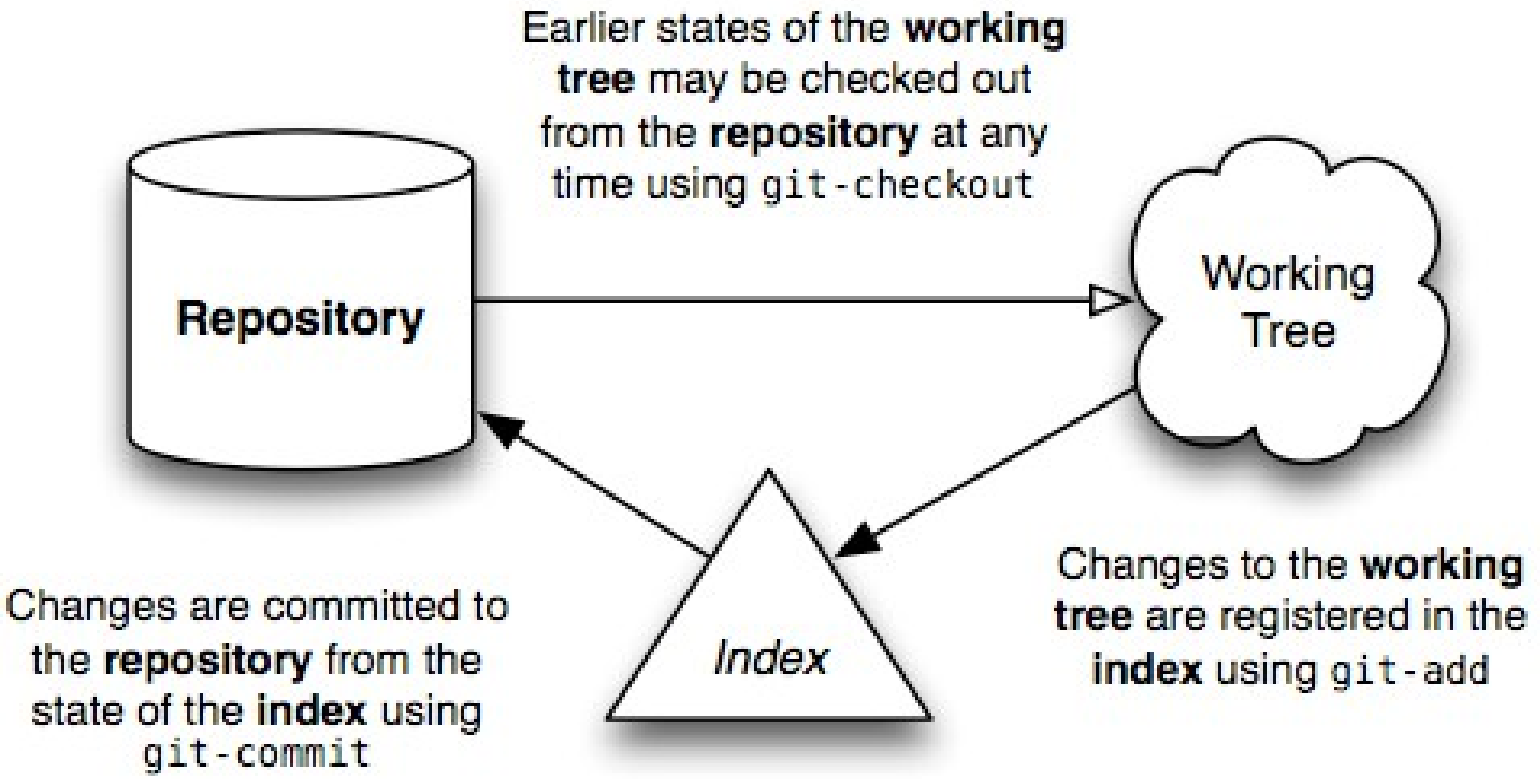
- This is often the last thing learned about, but in fact should be the first thing to be learnt
- Understanding how Git organises its files takes the voodoo out using Git
- It also helps you understand what is going on, so that you don't panic if you think you have lost your work, or believe you have overwritten it *

* you may well have have done this of course!

Git's architecture – Terminology

- **Repository** – a collection of ***commits*** detailing the project and its history
- **Working Tree** – a directory (and all its sub-directories and files) which has a ***Repository*** associated with it (the ***.git*** folder under the root of the project)
- **Commit** – a snapshot of your ***Working Tree*** at a particular point in time
- The **Index** – a staging area into which you register the code changes that you wish to ***commit***.

Basic work flow



Directory content tracking 1
What it does *not* do

- Git does not store a copy of all of the ***Working Tree*** files with each ***commit***

- Nor does it store a list of difference patches for an incremental replaying of project history

- It also does not store the files in a directory / file structure that is representative of your ***Working Tree***

Directory content tracking 2
What it does do

- Git stores all files it tracks in leaf nodes called ***blobs*** (equivalent of filesystem inodes / files)

- Git records the structure of the nodes within the ***Repository*** in a ***tree*** (equivalent of directory entries)

- A ***tree*** can contain both ***blobs*** and ***trees***. This is how it can build up the directory structure of your ***Working Tree***

- The final object type is the ***commit*** itself

Representation of filesystem in Git

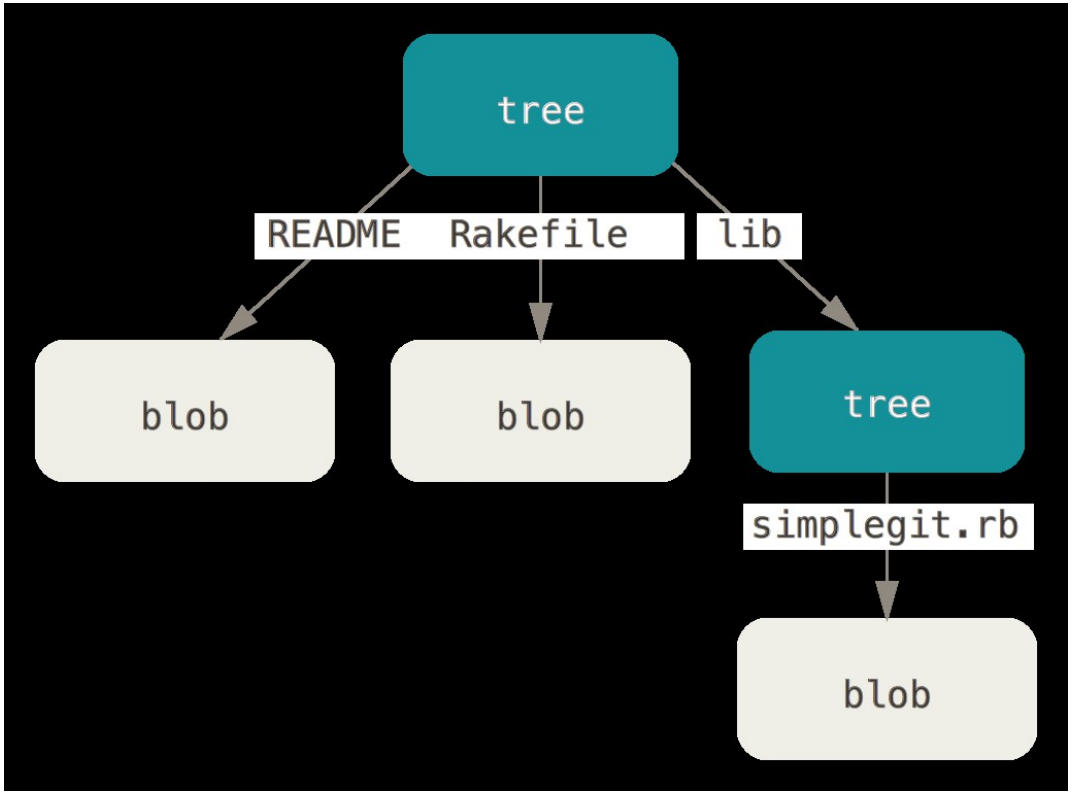
1 sub-directory

3 files

/lib/simplegit.rb

/Rakefile

/README



Object naming & metadata

- ***Blobs, trees*** & ***commits*** are named using the SHA-1 algorithm, giving a 40 character name:

  a5bce3fd2565d8f458555a0c6f42d0504a848bd5

- Thankfully we can refer to these by the first 6-7 characters, e.g. a5bce3

- ***Blob*** names are composed of a hash of the file size and contents, but not the filename

- This ensures that the ***blob*** will represent the file content regardless of any other file metadata

Blob objects

- Unlike a filesystem, data in Git is immutable (i.e. it cannot change)
- If the file being tracked is changed, then a new ***blob*** is written when the file is re-added to the ***Index***, unless...
- there is a ***blob*** that already contains that exact file contents, in which case the ***Index*** will reference this instead
- This provides a very compact way of storage, as only files which have been changed are stored afresh as a new ***blob***

Tree objects

- Are written recursively from the **Index** on ***commit***
- Each ***tree*** contains references to other ***sub-trees*** as well as the ***blobs*** at this directory level
- A ***tree*** also contains the metadata about the ***blobs*** and ***trees*** contained within:
  - Mode (file permissions – 0644, 0700 etc.)
  - Type (***blob*** / ***tree***)
  - Filename

Commit objects

- Store pointers to the snapshot of the ***Working Tree*** you staged in the ***Index*** to ***commit*** (i.e. the root ***tree***)
- Also store pointers to the parent ***commit***(s) (from second ***commit*** onwards)
- Also stores metadata about the ***commit***:
  - Author name + email
  - Committer name + email
  - Timestamp of commit
  - Commit message written when making ***commit***

Putting it all together

- Now we have a ***commit***, we can introspect into it, to find the SHA-1 id of the root ***tree***
- We can then look at this root ***tree*** to find the ***blobs*** and ***trees*** it contains
- We can then iterate through all the sub-***trees*** to see the ***blobs*** and ***trees*** they contain
- This gives us a complete snapshot of the ***Working Tree*** at the time the ***commit*** was made

That's a lot of work to do!

- Thankfully Git will do all of this for us
- There are many GUI tools on the market as well to help with this, although I find they confuse me more than they help
- Using GitHub will allow us to view the ***commits*** individually to see the changes that have been made
- It will also let us view the full snapshot of the repo at the point in time of the ***commit*** we are looking at

But that's not all by a long way

- There are numerous further concepts to learn to get a full understanding of Git's architecture, but for the moment the most important are ***branches*** and ***HEAD***

- ***Branches*** allow us to make parallel ***Working Trees***, which do not interfere with each other, and do not require a change of working environment

- This means we can easily diverge from the main line of development without messing up the code in the main line

Branches

- ***Branches*** are user-namable, movable pointers to ***commits***

- You can choose to name them, rather than have to use the 40 character SHA-1 id

- The ***branch*** pointer will move on automatically with each ***commit*** made on the ***branch***, so that it points to the last ***commit*** made – this tip is more correctly known as the ***branch head***

- By default the first ***branch*** is called **master**, but you are free to change this if you wish

HEAD (capitalised)

- At first this is one of the most mystifying things about Git
- It is actually so simple, it doesn't really make sense!
- It is just a pointer to the ***branch head*** (lower case) that you are currently working on
- Detached ***HEAD*** rightfully sounds bad, and can lead to danger, but it is just a case of ***HEAD*** pointing to a specific ***commit*** and not a ***branch head*** – we will cover this in a later session

To be continued...

Further resources:

- https://jwiegley.github.io/git-from-the-bottom-up/
- http://git-scm.com/book/en/v2/Git-Internals-Plumbing-and-Porcelain

- If you wish to ask me any questions whatsoever, please email:
- ingram.monk@gmail.com