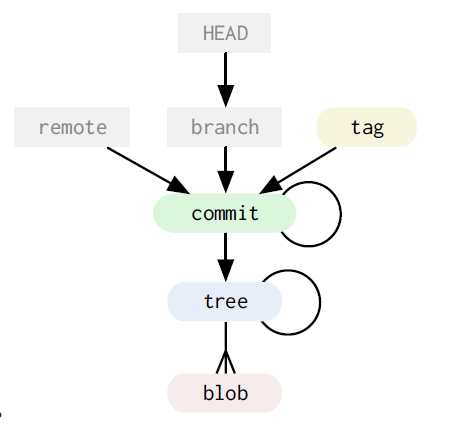
# Git Internals

## Git Object Types

Git objects are the actual data of Git, the main thing that the reposi - tory is made up of

All of these types of objects are stored in the Git Object Database, which is kept in the Git Directory. Each object is compressed (with 14 Zlib) and referenced by the SHA-1 value of its contents plus a small header (SHA stands for Secure Hash Algorithm)

In Git, the contents of files are stored as blobs. It is important to note that it is the contents that are stored, not the files. The names and modes of the files are not stored with the blob, just the contents.



### The Commit

So, now that we can store arbitrary trees of content in Git, where does the ‘history’ part of ‘tree history storage system’ come in? The answer is the **commit** object.

The Tag

The final type of object you will find in a Git database is the **tag**. This is an object that provides a permanent shorthand name for a par­ticular commit.

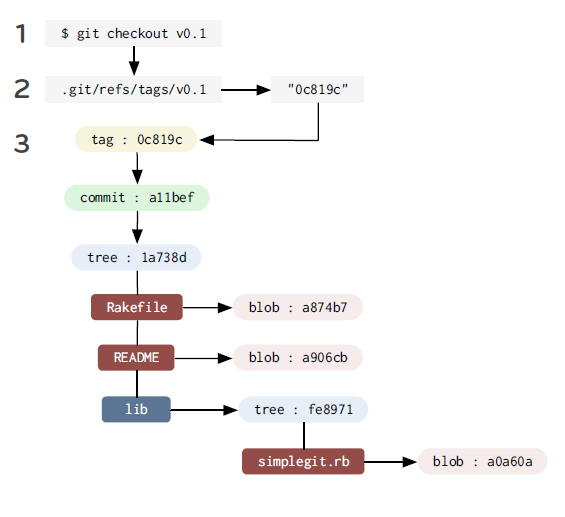
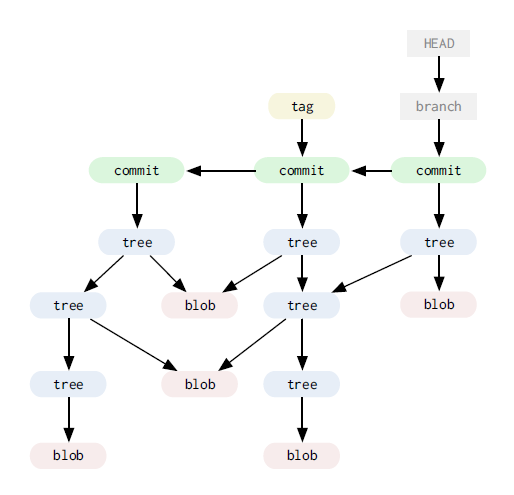
The Git Data Model

the Git object data is a *directed acyclic graph*

References

In addition to the Git objects, which are immutable – that is, they cannot ever be changed, there are references also stored in Git. Unlike the objects, references can constantly change. They are simple pointers to a particular commit, something like a tag, but eas­ily moveable.

A branch in Git is nothing more than a file in the .git/refs/heads/ directory that con­tains the SHA-1 of the most recent commit of that branch



In fact, in Git the act of creating a new branch is simply writing a file in the .git/refs/heads directory that has the SHA-1 of the last commit for that branch.

## The Treeish

* Full SHA-1
* PARTIAL SHA-1

dae86e

* Branch or tag name

Anything in .git/refs/heads

* Carrot parent

dae86e^N

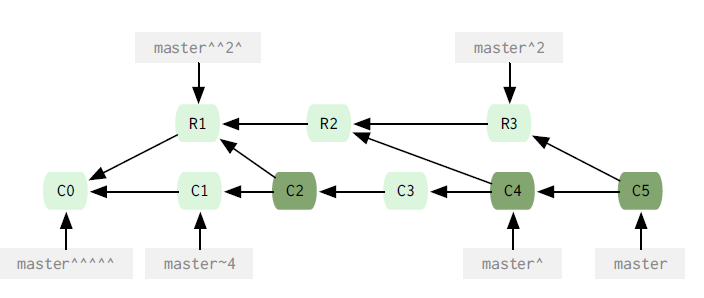
this refers to the Nth parent of that commit. Only really helpful for commits that merged two or more commits

* Tilde spec

dae86e~N

refers to the Nth generation grandparent of that commit

dae86e~5 ⬄ dae86e^^^^^



* Tree pointer

dae86e^{tree}

This points to th tree of that commit

#### Git repository

* .git/config
* .git/index

index file of the Git project

* .git/objects/

Main directory that holds the data of your Git objects ( Content of the files , commit, tree and tag obejcts)

This is the main directory that holds the data of your git objects that is all the contents you have ever checked in plus your commit, tree and tag objects. The files are stored by their SHA-1 values. The first two characters make up the subdirectory and the last 38 is the filename

a576fac355dd17e39fd2671b010e36299f713b4d ⬄ [GIT\_DIR]/objects/a5/76fac355dd17e39fd2671b010e36299f713b4d

* .git/refs/

This directory normally has 3 directories in it

Heads

Remote

Tags

* .git/HEAD

This file holds a reference to the branch you are currently on. This tells Git what to use as the parent of your next commit

* .git/hooks

Contains shell scripts that are invoked after the git command

### Working directory

Your working directory is temporary – everything is stored permanently in your git repository. Your working directory is a just a copy of a tree so you can edit it and commit changes

### Browsing Git

* git show master^

command to view the trees

* git ls-tree master^
* git ls-tree –r –t master^ (run it recursively)

Command to view blobs

* git cat-file –t ae850bd view type
* git cat-file –p ae850bd print blob

graphical interfcaces

* gitk
* git instaweb

### Searching Git

Git has an easy way for searching through treess in your repository whitout having to check them out into your working directory

git grep

### Git diff

If you simply run git diff with no arguments, it will show the differences between your working directory and the index

### Branching

Command to see the differences between the branches

*Git diff –stat master newfunc*

*git merge newfunc*

*git branch –d newfunc*

## Undoing a merge

*git reset*

*By default it will only reset your index, leaving the partially merged files in your directory.*

*git reset –hard*

*--hard makes sure both your index file and working directory are changed*

### Rebasing

git rebase master

if there is a conflict, yout have three things you can do here

* fix the file runs git add on it and run git rebase –continue
* run git rebase --abort will reset us to what our repo looks like before tried the rebase
* run git rebase –skip skips this patch , abandoning the change forever

When you clone a repository, it in essence copies all the git objects to a new directory, checks you out a single local branch named the same as the HEAD branch on the cloned repo (normally master) and stores all the other branches under a remote reference by default named ‘origin’

git checkout –track newfunction origin/newfunction

--track indicates that you may want to pull from or push to the origin of this branch later

Bare repository Is a repository without a working directory

# Statshing

git stash

take the changes from the last commit to the current state of your working directory and store it manually

I can see my stashes by running

git stash list

git stash show stash@{1}

git stash show stash@{0}

git stash show stash@{2}

git diff stash@{1}

And finally I can apply it

git stash apply stash@{1}

git stash apply without the actual stash reference it will just apply the last trash you saved on that branch

### Tagging

Creating a tag in git is much like making abranch. A tag is basically a signed branch that never moves

git tag –a v0.1 -m

### Multiple remotes

Decentralized part in git. You can add multiple repositories

git remote add mycap https://

git remote add official https://

useful information about a remote branch

git remote show origin

## Extra tools

* git gc: runs the garbage collector for your repository
* git fsck: does an integrity check of the git filesystem
* git prune: Removes objects that are no longer pointed to by any object in any reachable branch

git prune –n : to see what it will do