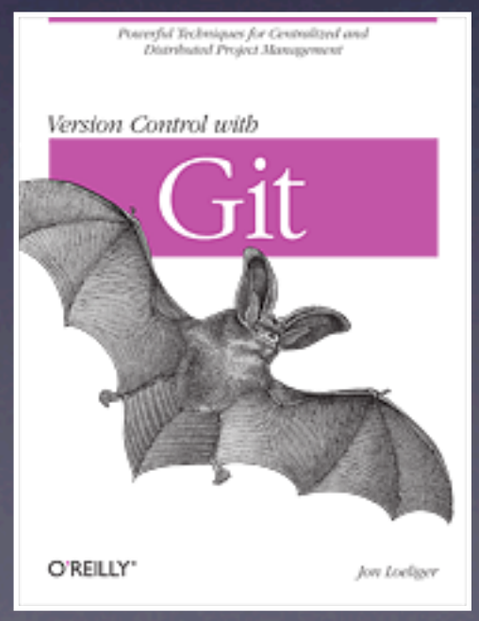
Git notes

NOLA Hackathon - October 2011

Warnings

- Git was made by über-nerds
- CLI syntax, terminology have evolved like Darwin's finches in a radiation leak
- "git help" is your friend
- Highly recommend the oreilly book.



Git data model

- (almost) everything is an object
- Object ID is SHA-1 hash of serialized object, including it's properties
- Type, size, <stream of bytes>

Object types

Blob

File contents

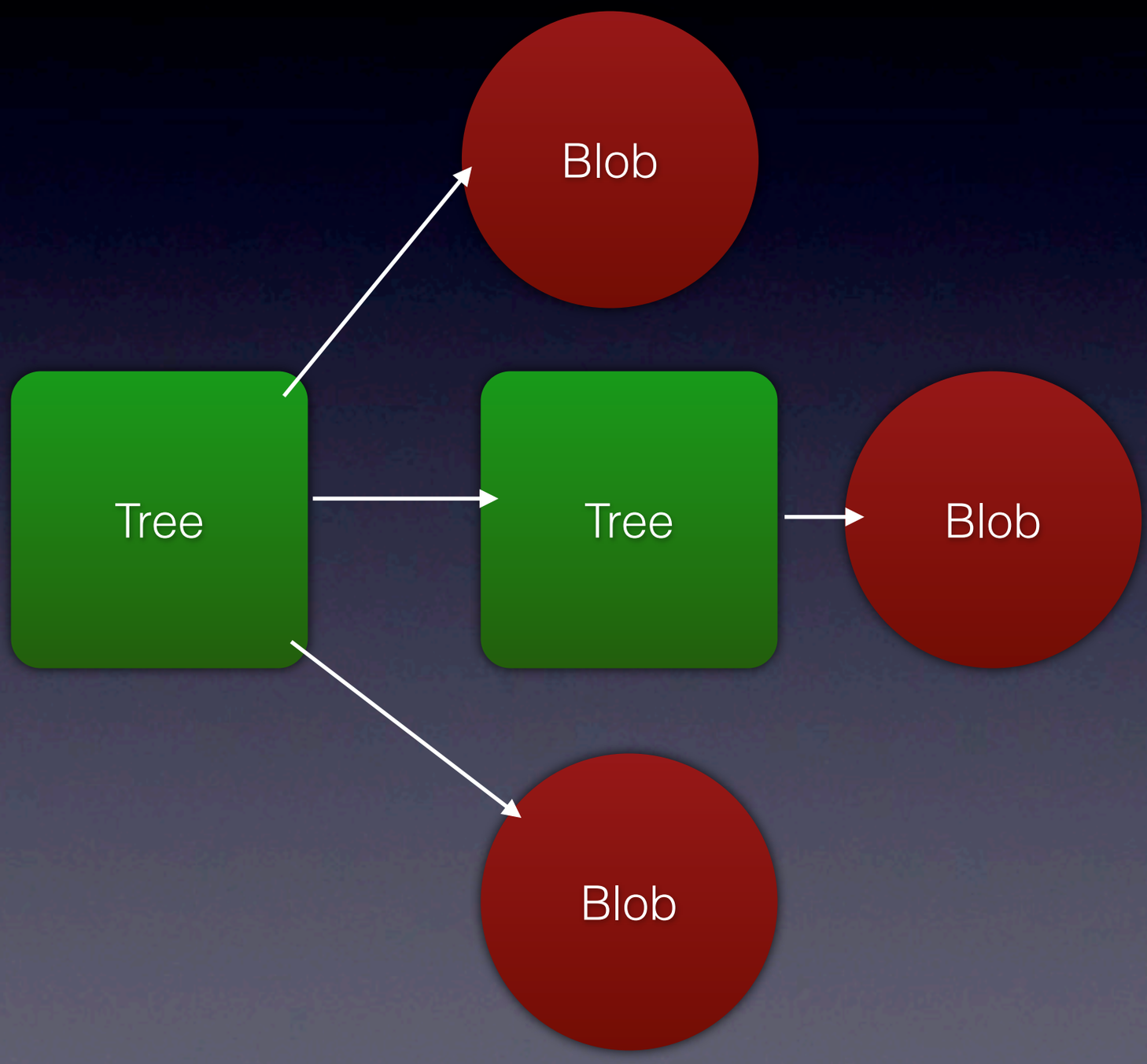
Tree

Directory tree

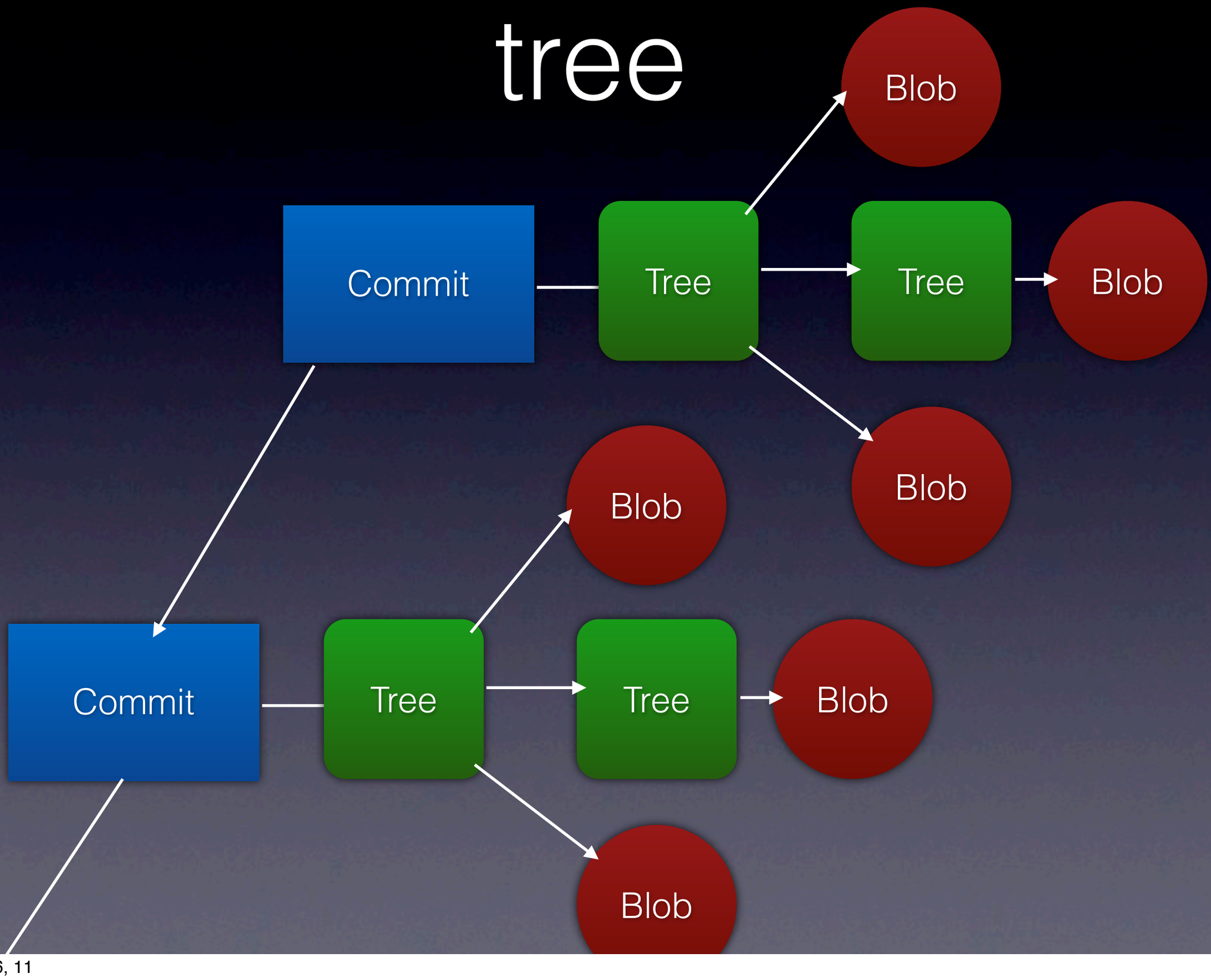
Commit

Version snapshot

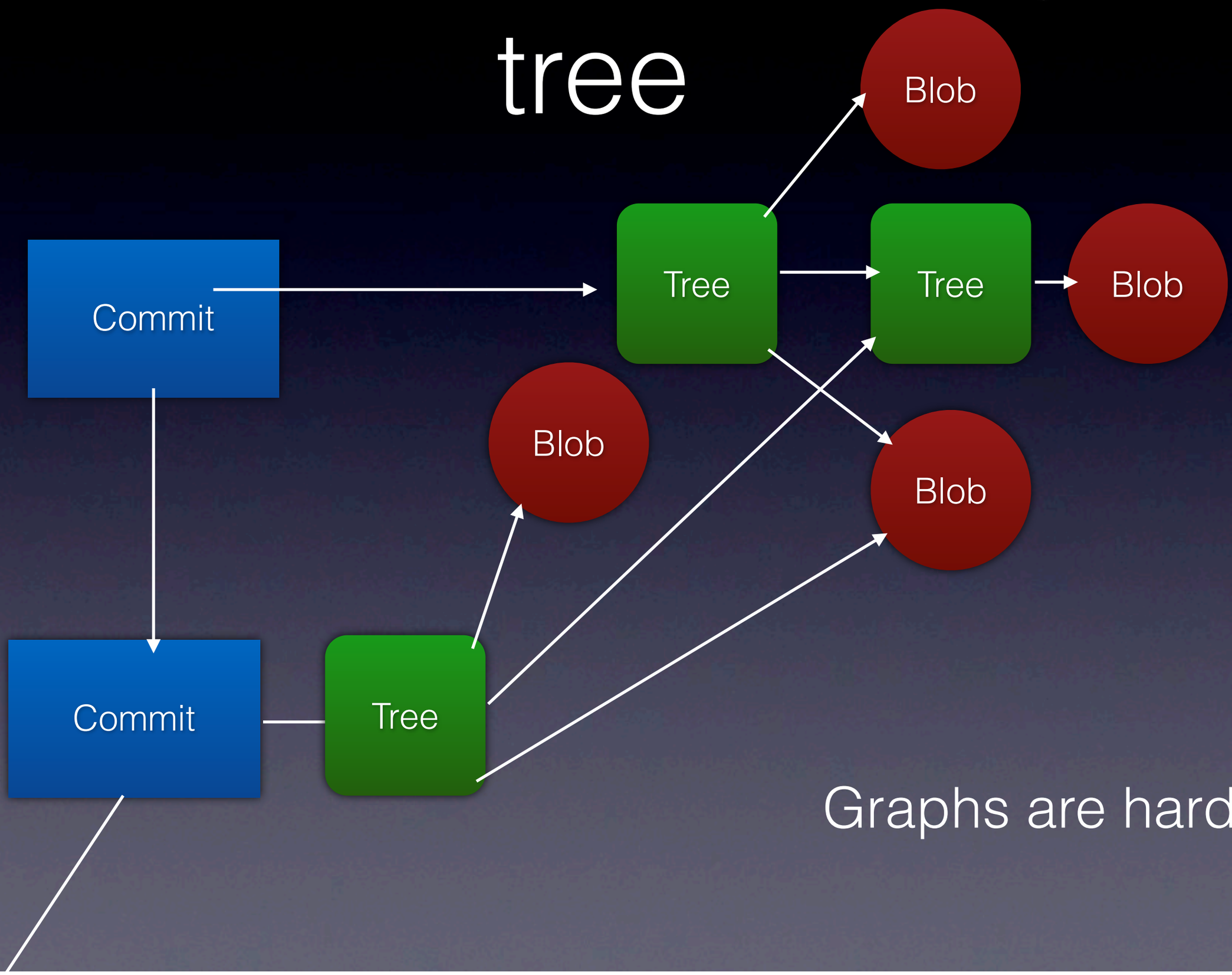
Directory tree



Versioned directory tree

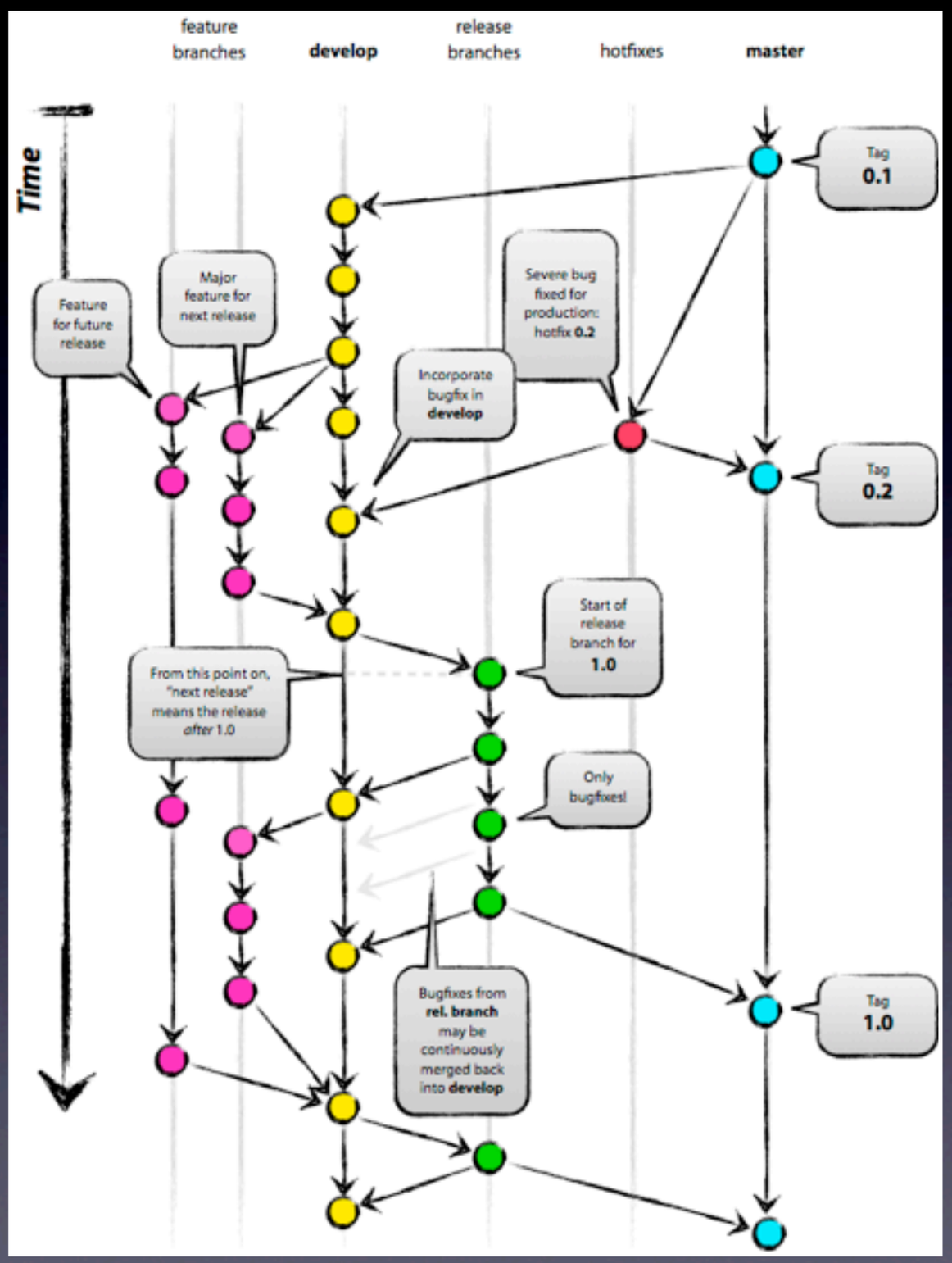


Versioned directory tree



Graphs are hard!

Branching models are variable

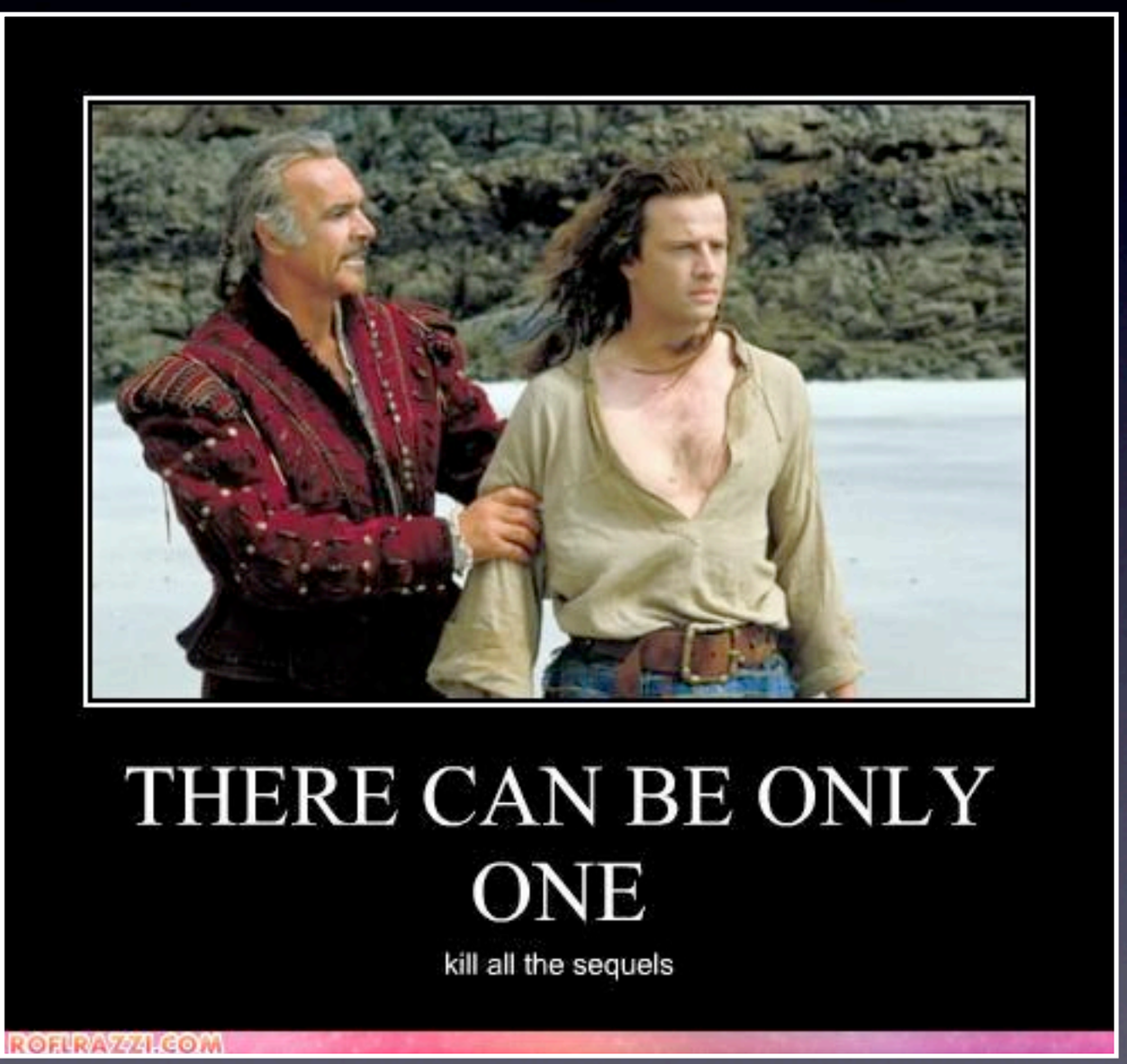


http://nvie.com/posts/a-successful-git-branching-model/

http://nvie.com/posts/a-successful-git-branching-model/

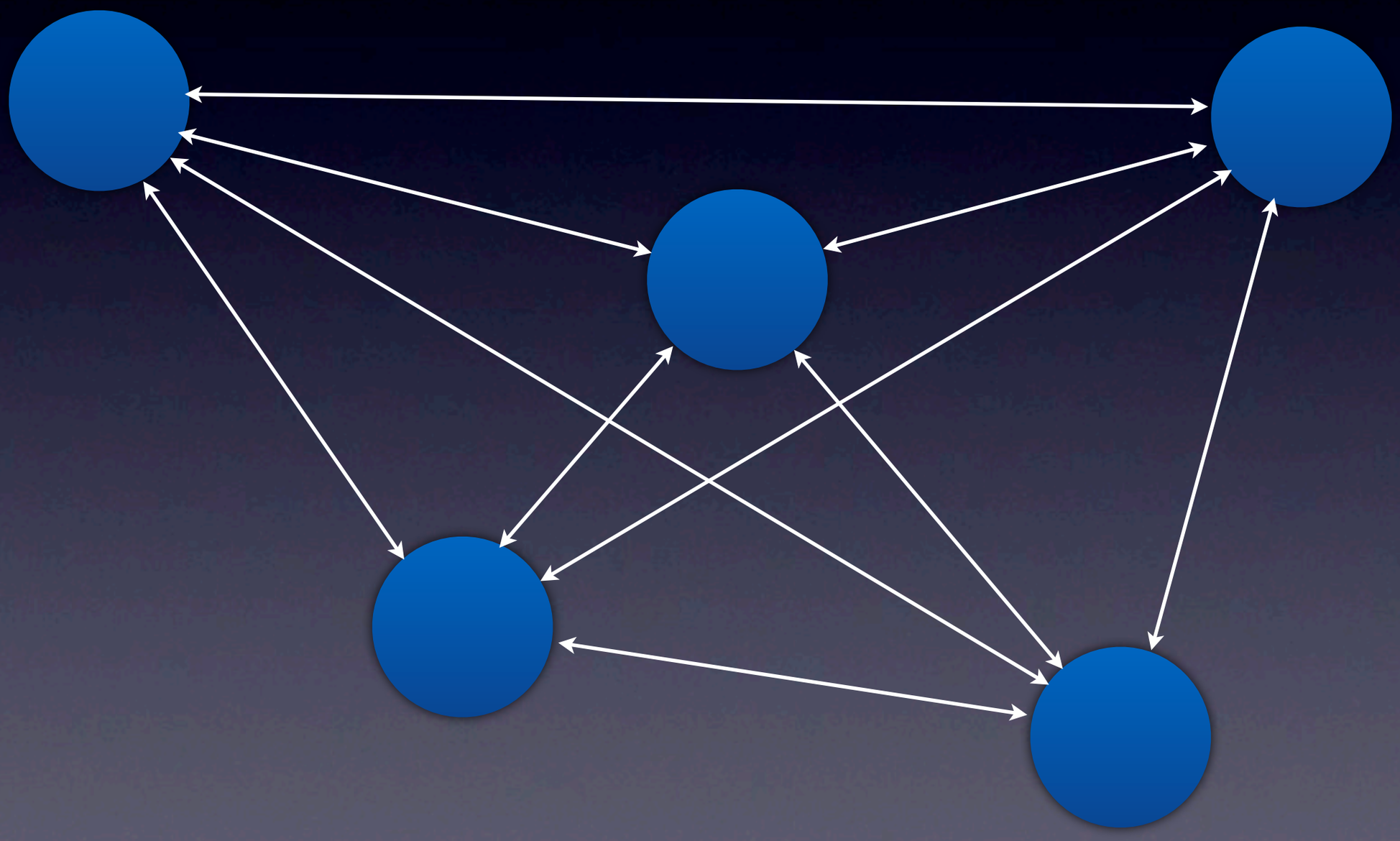
Repo sharing models

The Singleton



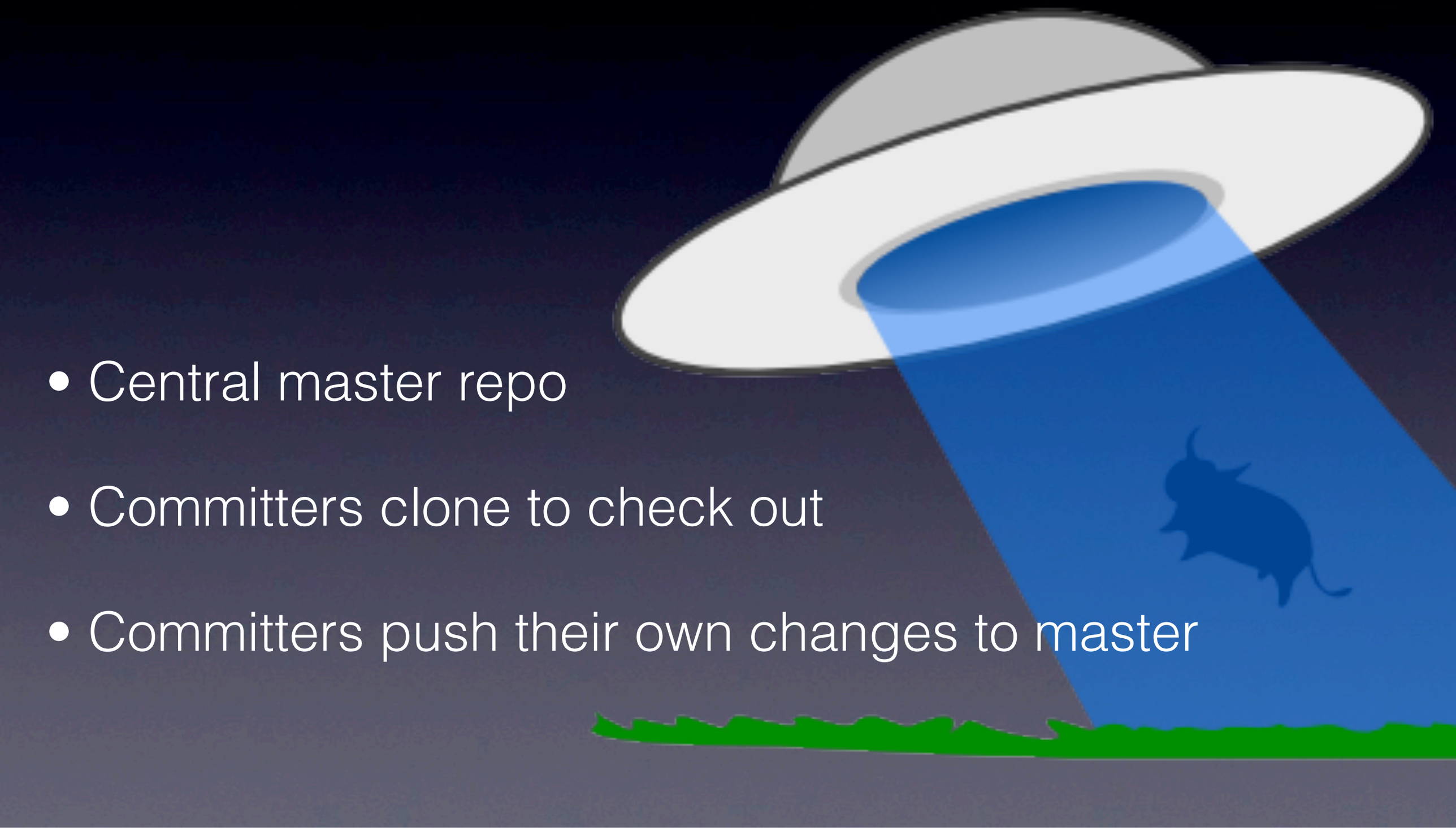
- "Git init" in any directory
- Local versioning, branching
- Can publish/share later with full history

The P2P Shuffle



In principle, each developer can maintain their own repository and push / pull things to each other, with no single canonical master. This, however, is a bit of a pain in the ass to manage if you have more than two developers!

Mothership



- Central master repo
- Committers clone to check out
- Committers push their own changes to master

As a practical matter, having a central master is very useful to organize around... but you can always change where the master lives. This is much like using CVS or SVN, but still gives benefits such as full history, offline branching, etc.

http://www.openclipart.org/detail/149239/cow-abduction-by-hector-gomez

Pull requests

- Github style
- Mothership + shared personal clones
- Complete free reign over your personal clone
- Helper tools to request, perform merge from personal clone to master

This is a style you'll see on sites like Github and gitorious. These sites have infrastructure for giving anyone with an account their own hosted clone of the repo, where they can push their own changes to share.

Some UI sugar makes it relatively easy to send a "pull request" asking the master project's maintainers to review and merge the changes in a branch.

This can be used for everybody's work to implement a review process, with the same facilities for longtime and drive-by contributors.

http://commons.wikimedia.org/wiki/File:Simplex_Pull_Station.jpg

Push to review

- Gerritt style
- Similar, but triggered by pushing to a particular shared repo.
-

Extension example

- https://github.com/brion/OEmbedConsumer
- The master repo!

Clone to local

cd extensions

git clone https://github.com/brion/OEmbedConsumer.git

LocalSettings.php:

require "$IP/extensions/OEmbedConsumer/
OEmbedConsumer.php";

Run it

<oembed>http://www.youtube.com/watch?v=Sau6l9TOhgk</oembed>

Make a tweak

Flickr support?

Error handling?

Toolbar button?

Show tree state

- git status

Branch & commit

- git checkout -b mybranch
- git add *
- git commit

Fun fact: the "git branch" command is NOT used to create a branch.

Well get into details on the add shortly...

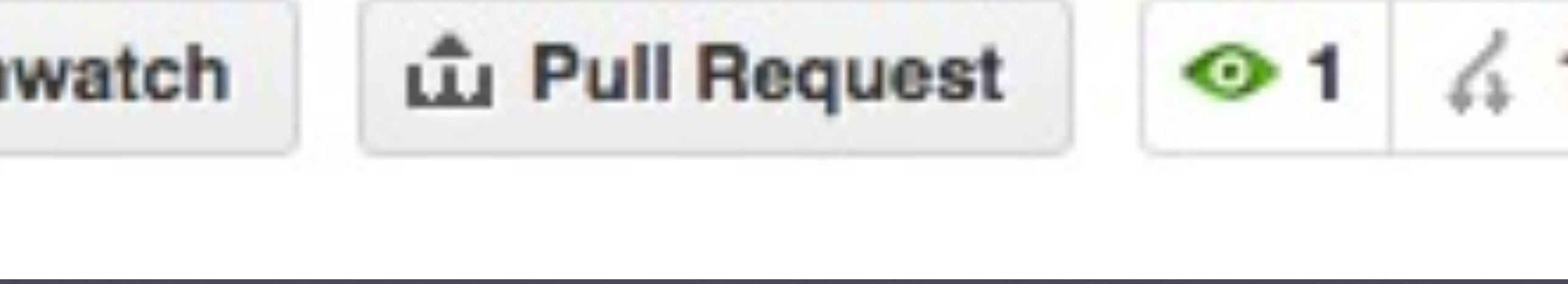
Share it

- Fork on Github
- git remote add myname .....
- git fetch myname
- git push myname mybranch

Up to this point I didn't need to be even set up with a Github account; we've just cloned from the public view of the master repo and made local modifications.

Logging and creating a personal fork on the hosting provider gives me a place where I can share any modifications I've made -- and where github's magic pixie dust can help us.

Pull!!



Because the commit history in your branch includes references to the versions you branched from, git is easily able to find the common ancestor. If the master has moved on from where you branched, it can merge your changes in like applying a patch.

Merging

Conflicts etc

The index

- Stuff in here also called "staged" or "cached"

If "commit" reminds you of transactional systems, there's a good reason for that. While you're fiddling with code you're in a transitional state, where git doesn't know what hangs you've made yet.

Before actually committing, you first have to stage the changes: the "git add" command for instances stages a new or changed file for the next commit.

For extra confusion, "git diff" shows only changed files that have *not* been staged. This can be nice when you want to make sure you've got all your changes ready to commit though.

Rebasing

- Scary!
- Useful when cleaning up a work branch for publishing

Rebasing is one of the scarier git features, and is one you should approach with caution.

Essentially rebase "rewrites history" by replaying the changes from some existing series of commits, and recording them differently.

Working with other developers is like slowly moving where you keep your shoes from the front door to the closet... when you're partway through, someone who comes in isn't sure whether to put their shoes in which place.

Sometimes you move things to the closet but the closet's gone, somebody removed it...

Rebase is like changing the past so the closet NEVER EXISTED!!!

This is useful for a number of scenarios: merging together a series of false steps into a single commit with a working solution, or reordering them to make the editing logic clearer, or reapplying a change against the current version of the master repo to make the resulting merge cleaner.

Never, ever rebase a branch that anyone else may have cloned, as it will confuse the heck out of them when they next pull updates.

Cheat sheets

Abandon all hope!

- git reset --hard HEAD
- git reset --hard origin/master

sometimes you gotta give up

Pulling someone's branch

- git remote add ... ...
- git fetch ...
- git checkout .../...

Merging branches manually

- git checkout dest
- git merge source
- If conflicts, fix & git commit
- Now push it or something!

http://www.mediawiki.org/wiki/Git_conversion/Splitting_tests