Git: Making a hash of your source code

Michael Gray
Anacode Team

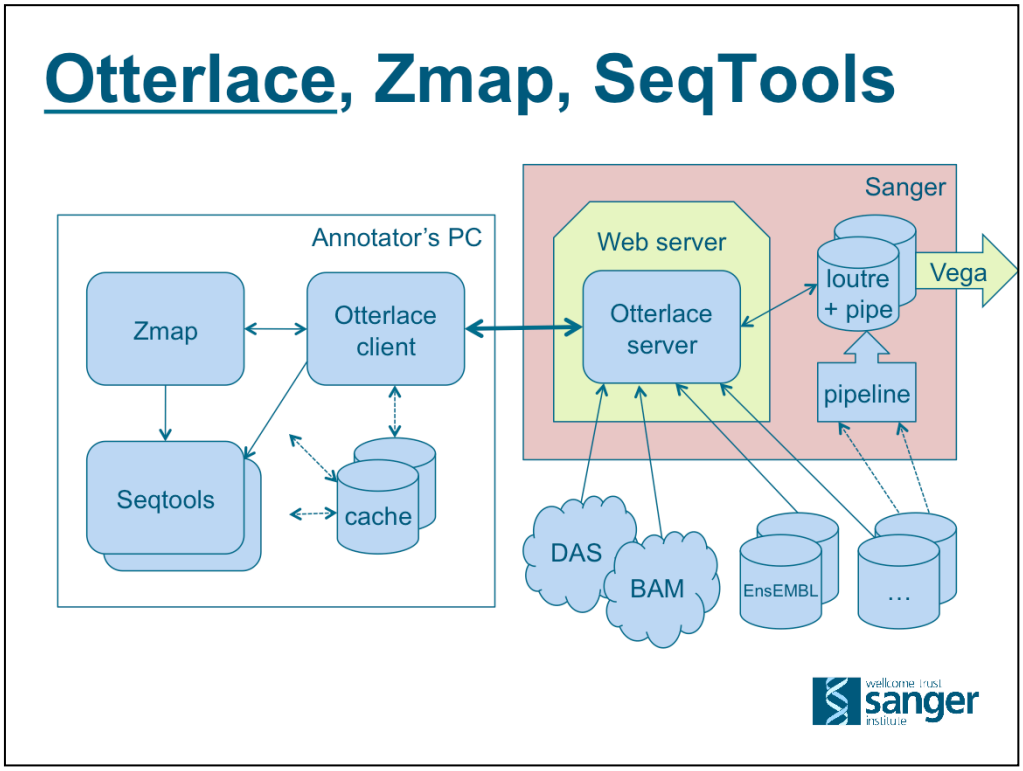
Talk about how the Anacode Team migrated most of our code base from CVS to Git.

Disclaimer: I am not a Git expert.

Git: making a hash of your source code by Michael Gray for Genome Reseach Ltd

is licensed under a Creative Commons Attribution 3.0 Unported License.

http://creativecommons.org/licenses/by/3.0/deed.en_GB



Otterlace is the glue in the annotation tools used by the Havana team.

It works with Zmap, which displays transcripts and columns of evidence, and with blixem, dotter and the other alignment tools in Seqtools.

Otterlace gathers and presents the evidence sources, including those generated in our anacode pipelines, and captures and saves the gene annotation.

All communication is via http to the otterlace server, allowing annotators to work anywhere with an internet connection.

Evidence sources include DAS & BAM, EnsEMBL (local mysql) and PSL format databases (UCSC).

Otterlace: the code

Perl

- **OO**
- **Tk, EnsEMBL, CPAN**

- **~ 33,000 lines of code in ~ 200 files**
- **~ 4,000 lines of comments**
- **~ 10,000 lines intentionally left blank**

http://www.sanger.ac.uk/resources/software/otterlace/

wellcome trust sanger institute

Object-oriented perl: traditional and recently moose.
Tk for GUI, EnsEMBL API, and lots of useful CPAN.

Otterlace client and server stats: quite big! (Total heading towards 50,000)

More info at the URL.

How it was in the beginning

CVS

- **Forking difficult**
  - Actually, *merging* is difficult
  - Fork off and don't come back
- **Infrequent releases**
- **Inter-team coordination:**
  Room for improvement

wellcome trust sanger institute

CVS makes it quite onerous to handle parallel branches and especially to merge changes back in.

This lead to a reluctance to have too many.

In turn this put a drag on releases.

We depend on coordinated releases of Zmap & Seqtools from the annotools team and they were suffering in the same way, making coordination less than super.

Regular releases.

Properly controlled testing and production releases.

Why Git?

- **Reliable**
  - **Snapshots**
  - **Checksum of every file in each commit**
- **Cheap branching, easy merging**
- **Local**
- **Distributed**
- **Small, fast, free & open-source**
- **Mature & heavily used**

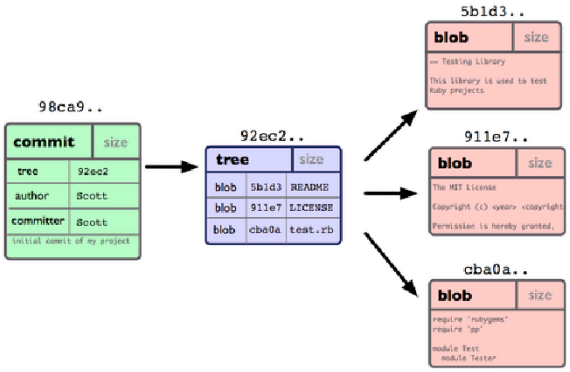
wellcome trust sanger institute

Git is based on snapshots, not deltas.

Everything is check-summed with a SHA1 hash – hence pun in the title.

Let's have a look at a git commit

A git commit



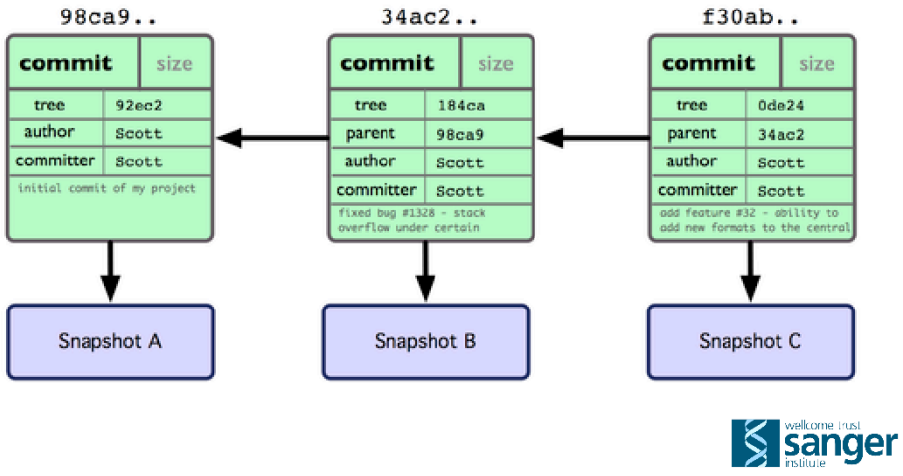
Work from right to left:

•A snapshot of the file

- •Here there are three files – each with SHA1 hash of contents
- •In one directory – again with SHA1 hash of its contents
- •Making one commit with a few headers, a message, and a SHA1 hash – which becomes the commit ID

What happens if we make some changes and create a new commit?

More git commits



A new snapshot is created, and the new commit contains a link to its parents.

Identical blobs are automatically shared because they are referenced by the hash. (This is highly likely for most files in any two nearby commits.)

So git is quite efficient.

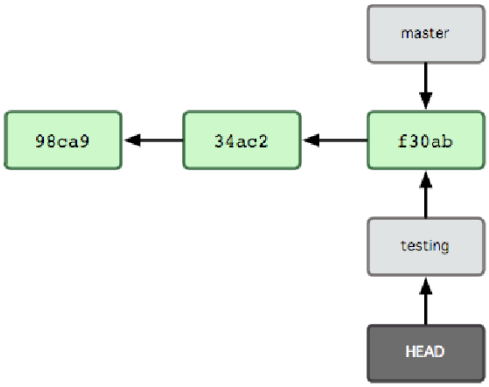
Why Git?

- **Reliable**
  - **Snapshots**
  - **Checksum of every file in each commit**
- **Cheap branching, easy merging**
- **Local**
- **Distributed**
- **Small, fast, free & open-source**
- **Mature & heavily used**

wellcome trust sanger institute

Making branches in Git is very lightweight, and git is good at merging branches back together.

A new branch



Branches are (almost) just labels pointing at a particular commit.

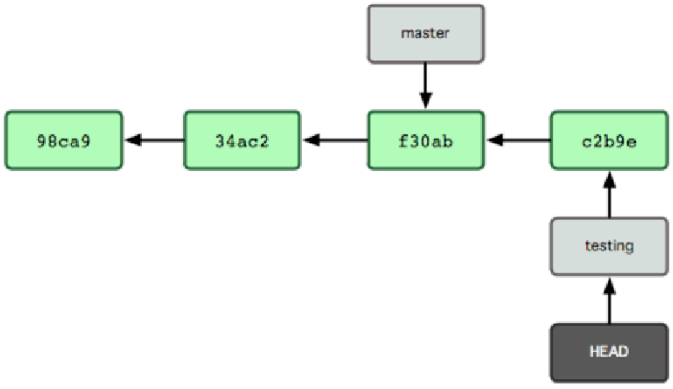
Here, a new testing branch has just been created – by making a new label and pointing it at the current commit.

Nothing has changed from the point of view of actual commits.

The testing branch has been checked out indicated by HEAD pointing at it.

If we make a commit on the branch…

The branch grows



…new commits are added at the HEAD, moving that branch forward.

So the testing branch label now points at the new commit. The master branch is still pointing at the previous commit.

Why Git?

- **Reliable**
  - **Snapshots**
  - **Checksum of every file in each commit**
- **Cheap branching, easy merging**
- **Local**
- **Distributed**
- **Small, fast, free & open-source**
- **Mature & heavily used**

wellcome trust sanger institute

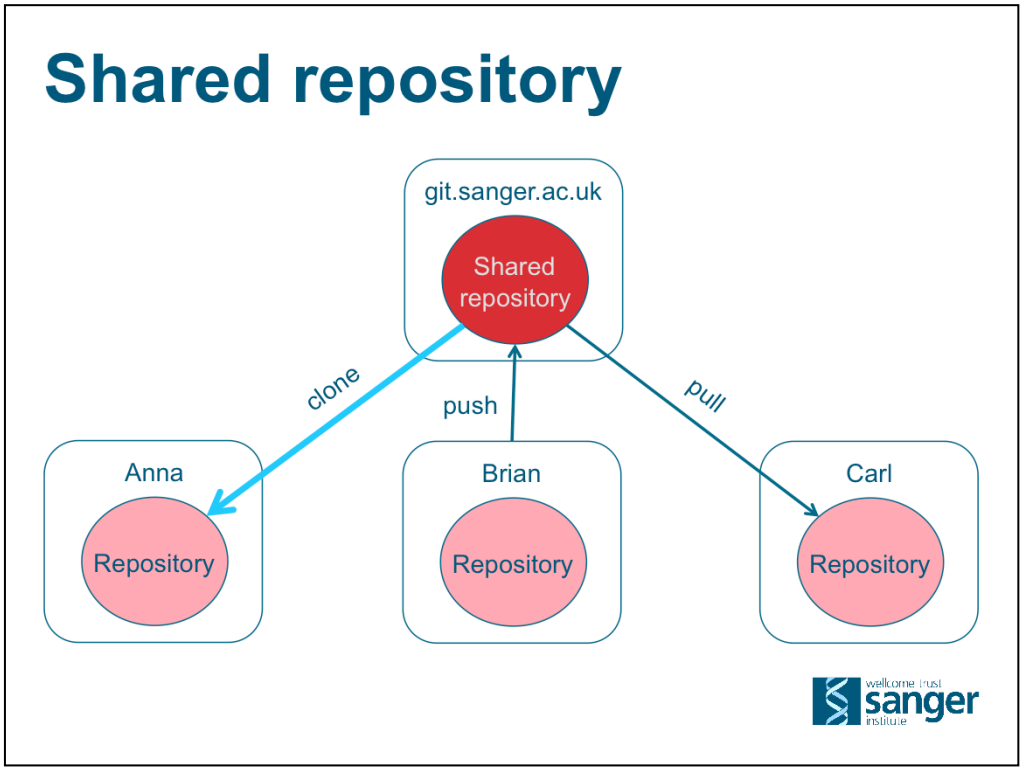
If you're a developer, git works locally on your machine in your own copy of the repository (more on this later)…

Why Git?

- **Reliable**
  - **Snapshots**
  - **Checksum of every file in each commit**
- **Cheap branching, easy merging**
- **Local**
- **Distributed**
- **Small, fast, free & open-source**
- **Mature & heavily used**

wellcome trust sanger institute

…but git has a flexible set of tools for shared development. There are many ways of doing this…



…but for a small team the most straightforward is a shared repository.

In our case the shared repository sits on git.sanger

Anna, new developer can clone a copy of the repository onto her machine. This is a self-contained copy, but set up to track the shared repository.

Brian has some changes (a string of commits on a branch) ready for public consumption, so he pushes them to the shared repository.

Carl regularly pulls changes from the shared repository to make sure his local repo is up-to-date. (And so too should Anna & Brian).

Why Git?

- Reliable
  - Snapshots
  - Checksum of every file in each commit
- Cheap branching, easy merging
- Local
- Distributed
- Small, fast, free & open-source
- Mature & heavily used

wellcome trust sanger institute

Further benefits: Small…

Used for Linux kernel development.

Branching is quick and easy. Merging back is usually easy too. So make branches…

Making commits is easy.

More, smaller commits are usually better.

They should be…

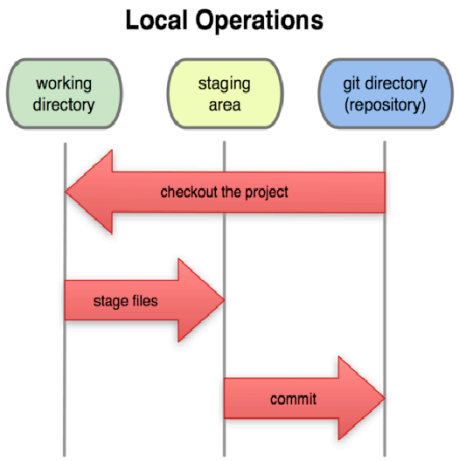
If your commits follow these guidelines, git makes it very easy to cherry-pick a feature or bug-fix commit from one branch onto another.

(Cherry-picking and rebasing are dressed-up merges.)

You're in control of your own full-featured local git repository.

Provided you haven't published your latest branch or tree of commits, you can undo, redo, split, merge & rename commits.

Local development



Git lets you build your commit incrementally into a staging area.

You don't need to put all the changes in your working tree into the same commit, nor even all the changes from one file.

Stage related changes until you have a consistent changeset – and you may still have further changes in your working directory – and then commit when ready. Carry on working or build your next commit.

Powerful tools like git-gui to help.

Once you're happy, push your changes to the shared repository.

Git takes care of noticing conflict, and provides lots of tools for resolving.

To minimise the possibility, pull often from the shared repository to stay up-to-date.

Tools:

•gitk for visualising (see later)

•gitweb

Command line:

•Git grep

•Git bisect

Git flow lite

- **TIMTOWTDI**
- **Choose a workflow to suit**
- **We based ours on *git flow***

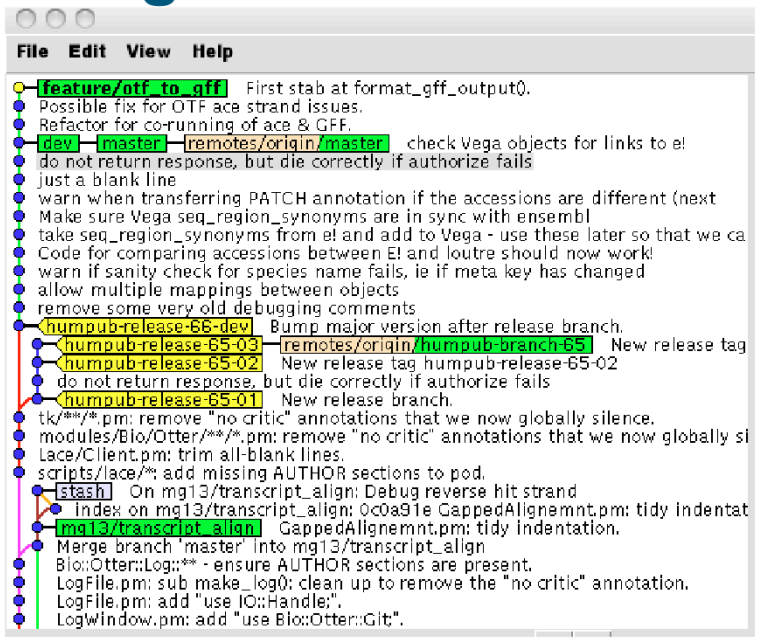
http://nvie.com/posts/a-successful-git-branching-model/

wellcome trust sanger institute

Git is a set of tools, and can be used to implement all sorts of workflows.
There is more than one way to do it.

As a team, we needed to choose a workflow.
Ours is fairly basic but conventions informed by git flow.
(Extensions – we don't use them but annotools do.)

Our git workflow in action



This is a snapshot of the main panel in the gitk tool

Timeline: newest at the top

Main line is our master or development track

There was recently a test release – branch of r65

You can't see the production branch further back

Notice cherry-pick

Feature branches.

Quite a bit of work! Much of it done by mca.

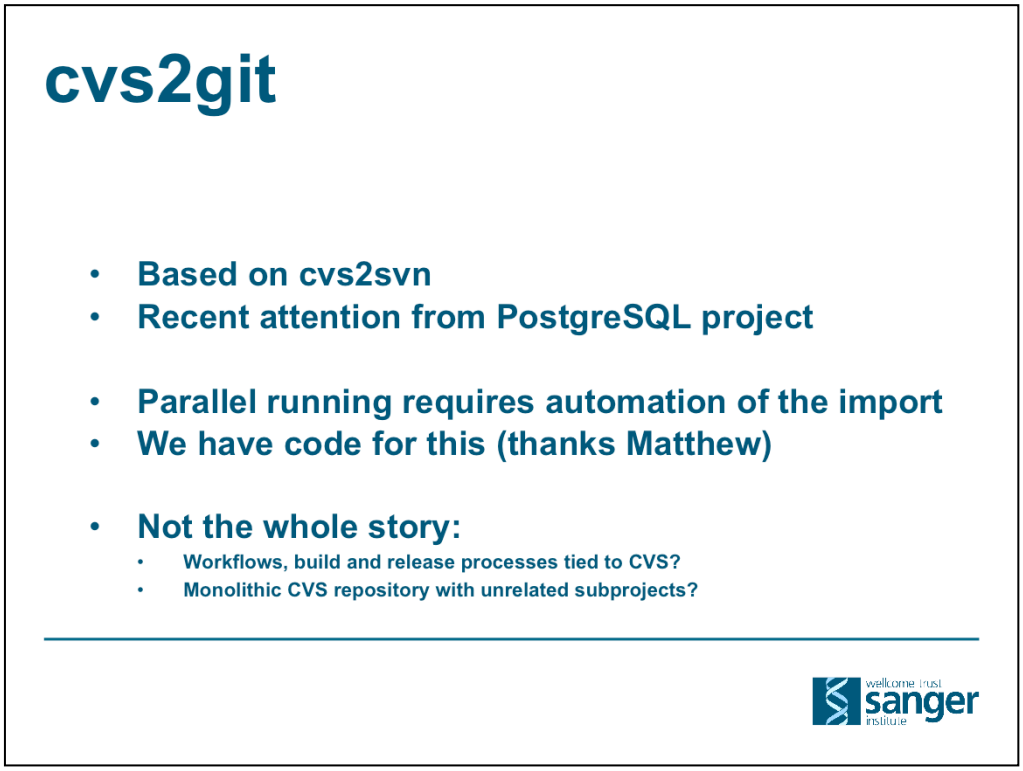
We used cvs2git: more in a moment

We wanted to preserve history: the work was mostly detecting "surgery" done over the years and fixing it up.

Problems caused were import failure, branches whose names were lost, and some diffs between "cvs checkout" and the resulting "git checkout".

Parallel running: useful in itself. Useful for…

Parallel running seems to be sustainable over several weeks and maybe longer. But some care required.



this tool seemed the best to understand the old CVS files, and the problems they may have

Parallel running and "iron out bugs" require automation of the import. We have code for this which works nicely onintcvs1:/repos/cvs and cvs.sanger.ac.uk

I think it's honest to admit that there may be things tying a team to CVS.

Monthly releases

wellcome trust sanger institute

So we now have the tools to move to a comfortable monthly release cycle, shown here as versions.

The purple track is our master branch and is actually continuous from a git point of view.

A new branch is started to make a test release – test64

Dev continues in preparation for 65.

Once testing is complete, we adjust our links to make it live – live64

If a serious bug appears after testing, we can make a hotfix branch, sort the bug, and merge the fix back in.

We then cherry-pick the fix into the dev and test branches, if appropriate.

Not shown: cherry-picking test fixes into dev.

Two or three main active branches (not counting features & hotfixes)

…knowledge, hopefully

Git doesn't make sense straight away – it didn't for me - but the penny drops.

When it does, it feels helpful, powerful & flexible.

Ask advice from regular users.

Gitk is very helpful for visualising where you are.

You will make mistakes but git makes it easier to rewind and try again.

Now that annotools and anacode teams are using git with established workflows, monthly releases have become entirely practicable.

Bug fixes are reaching annotators more quickly.

New features are flowing – not necessarily large ones every month, but we can now plan for their development.

Flip-flop curing and reintroduction of bugs much reduced.

Now running for 9 months!

Special thanks to Matthew who did a lot of the migration work and continues to provide git expertise to the team.

Some images taken from the main git website – CC licence – attribution when slides go ontp Helix.

Acknowledgements

Some images taken from
and copyright owned by:

http://git-scm.com/

re-used here under a
Creative Commons
Attribution licence.