git and its History

Once upon a time there was diff

The Unix world (GNU, Linux) was heavily based on text files

- Text is always readable (it's friendly to men)
- Text is accessible for impaired people (it's friendly to all men)
- A simple editor can change any text file (it's friendly to machines)
- ASCII text is portable (it's friendly to all machines)

History, at last, is validating our views:

- All successful network protocols are text protocols
- HTML is a text format
- XML (depicted as the solution to all sins) is text
- ODF is text (though embodied in a zip file)

With every task, with every data, people need to know what changes they accumulated while working

One of the first Unix programs was "diff"

- Diff assumes that text files are line-oriented
- The program reports to stdout the differences between two files
- Differences are shown as a textual printout

Diff generated patch

Baby diff could return commands for "ed", the editor:

- **diff -e <oldfile> <newfile>**

In his youth, he learnt to show context around differences to be more pleasant for men:

- **diff -c <oldfile> <newfile>**

Once grown up, he learnt to be terse to be more pleasant for machines and networks:

- **diff -u <oldfile> <newfile>**

The "patch" command, initially written by Larry Wall, uses diff's output to change <oldfile> into <newfile>

- **It's used to "patch" a wrong file into a correct one**
- **Patch files are just diff's output saved to file**

And SCCS came. He made RCS, CVS, SVN

SCCS and his children keep track of versions and revisions

- They save all the revisions, by storing diffs
- The new ideas of "checkout" and "commit" appear
- Programmers can use symbolic names («tags»)
- The tools automatically generate version numbers (e.g.: 1.432)
- Every commit is augmented by a log message

CVS, quite widespread, is mainly just RCS over the network

- Its own files live in a centralized repository
- Several uses can concurrently access the repository to read or write
- The network protocol is simple
- Conflicts are handled, though with some limitation

What's wrong with CVS:

- Revisions are only maintained within each individual file
- If you rename a file you loose its history
- If you split or merge files, you loose their history
- Committing several files is not an atomic operation

Hash, CRC, MD5, SHA1, rsync, PowerPC

The idea of hashing predates diff and patch

- Theoretically, a hash is an injective, non-invertible, math function
- In practice, it's something that represents a file with a number
- It's a basic concept people study (and forget) in the first CS course

CRC32: polynomial algorithm run on a bit sequence

- The CRC in the "cksum" command is a Posix standard

MD5: message digest, by Dan Rivest

- Hash algorithm returning 128 bits, decently secure

SHA1: secure hash algorithm (one)

- Hash algorithm returning 160 bits, more secure

rsync is a remarkable application of the hash idea

- It allows synchronization of files or file trees
- It only exchanges the differences, limiting network traffic

Even PowerPC virtual memory is based on a hash table

- Page tables are just one level (on x86 it was 2 levels, now 3)
- Memory size, and thus conflict rate, is configurable

Then Torvalds came, and Stallman got upset

Linus used to refuse SCM tools, for three reasons:

- CVS is crap
- CVS is real crap
- SVN ("cvs made the right way") can't be but sweeter crap

Larry McVoy, a previous kernel/scm hacker in Sun, offers help

- He would write BitKeeper according to Linus' needs
- He would allow free software to use BitKeeper
- He would sell Bk with proprietary license terms

On year or so later Bk is ready, and the kernel starts using it

- Centralized repository on bkbits.net
- Restrictive license, repeatedly modified over time
- RMS and many of his area shout out in disgust
- Some core hackers refuse to use Bk
- Linus is overall satisfied

But, as renown, power (of license) makes men greedy

- An "arch" developer was immediately revoked his Bk license
- Over time the license turned more and more restrictive, until...

Linus made git in his own resemblance

Git is a distributed version control system

- There is no centralized repository
- Each programmer gets hold of the whole history, locally
- If everyone pulls from Torvalds, it's just because he has the best bits

Git manages the whole project as a unit

- It doesn't keep separate history for each file
- It supports renaming, copying, merging, splitting

Each version is blessed with its own SHA1 hash

- There are no more sequential numbers that are hard to remember

Git has its own tools to share information over the network

- It has its own client/server network protocol to exchange data
- It can leverage on email as primary exchange channel

A different approach to file history

Git handles 4 different types of objects

- Files (called «blobs»)
- Directories (called «trees»)
- Commits («commits»)
- Tags («tags»)

All objects are immutable

- Each object is identified by its own hash
- Git stores objects in a directory «objects», according to their hashes
- An object may include hashes that link other objects

The status and complete history of a file-set is identified by the final commit in such history

- Every commit is represented by its SHA1 hash
- If two users own the same commit they automatically own the same files
  - To authenticate a file-set and history you just sign the hash

Branch management, merging, conflicts

A "branch" is one of several histories of the project

- Each programmer works on her own branch
- In a single directory a programmer usually hosts many branches
- No branch makes any reference to other branches

Branches are only compared when needed

- Every branch is independent, but when merging

There is no such thing as a main branch, like in SVN

- Actually, the concept of trunk is not applicable at all

Git only merges local branches or external patches

- Everything must be already committed
- No conflict can ever lead to information loss

We have tools to move branches or change past history

- git rebase
- git rebase -i
- git commit --amend

They reconciled and lived happily ever after

Git is released as free software

- **GNU GPL License**

Most software projects are now relying on git for their own development:

- **Xorg**
- **LibreOffice**
- **busybox**
- **perl**
- **qt, gnome**
- **fedora**
- **ffmpeg**
- **buildroot**
- **...**

And most "centralized" sites are not actually centralized.

- **I rarely login into github, gitlab, bitbucket, ...**

Notes about git "advanced" use (1/2)

Some useful features of git that are rarely used

- objects/info/alternates
  - This allows to access objects from another dir
  - Prefer this over $GIT_ALTERNATE_OBJECT_DIRECTORIES

- "git remote" to have local references to other repositories
  - You can fetch all branches from a remote to a safe place
  - You can manually specify how to fetch and push to a remote
- "git cherry-pick" to take an individual commit from a local branch
  - This is especially useful if you commit by file
  - Note that upstream want commits by feature
- "git show <commit>:pathname"
- "git show <hash>"
  - You can recover any file from any commit or any diff
- "git diff <commit> <pathname>
  - You can diff an individual file
- "git diff <hash> <hash>"
  - You can diff releases or files

Notes about git "advanced" use (2/2)

Other advanced commands that are very useful at times

- git blame
  - Show the guilty guy for each line of code
- git bisect
  - Find the faulty commit for a regression
- git merge-base <rev> <rev>
  - Find the last common commit of two branches
- git filter-branch
  - Rewrite stuff keeping history
- git rebase -i
  - Change history at will
- git add -p
  - Select individual patches to be committed
- git commit --amend
  - Modify an existing commit
- git commit -C <rev>
  - Reuse a commit message

Useful Utilities

- **git archive**
  - **Make a tar/zip file**
- **git grep**
  - **Faster alternative to grep -r**
- **git log**
  - **All kinds of loggin information**
- **git revert**
  - **Revert a commit (then you may "rebase -i")**
- **git stash**
  - **Park/unpark local modifications**
- **git describe**
  - **Print a human-reable name for the current status**
- **git fetch**
  - **I prefer "fetch" and "rebase" over "pull": it's safer**
- **git format-patch**
- **git am**
  - **Generate patch files (email msgs) and apply them**
- **git submodule**
  - **Import another project as a checkout in a subdir**