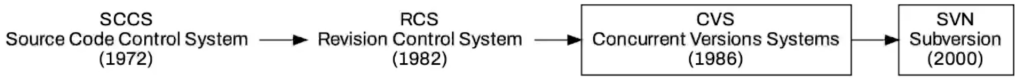
Git: Overview

History:

Notes on the history of git



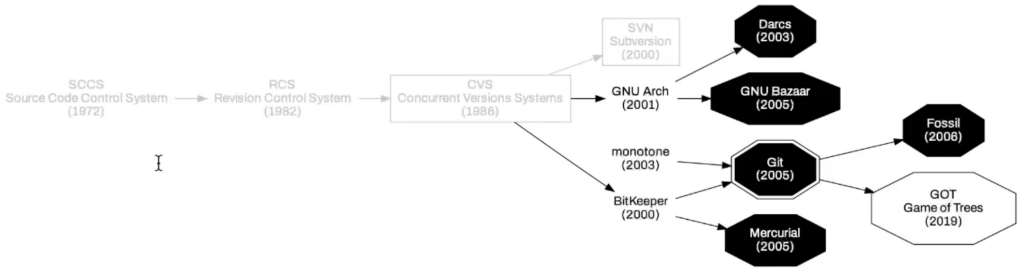
Initially all the version control systems are *centralised*...

- that is they each have an *official* central repository that stores the latest versions.

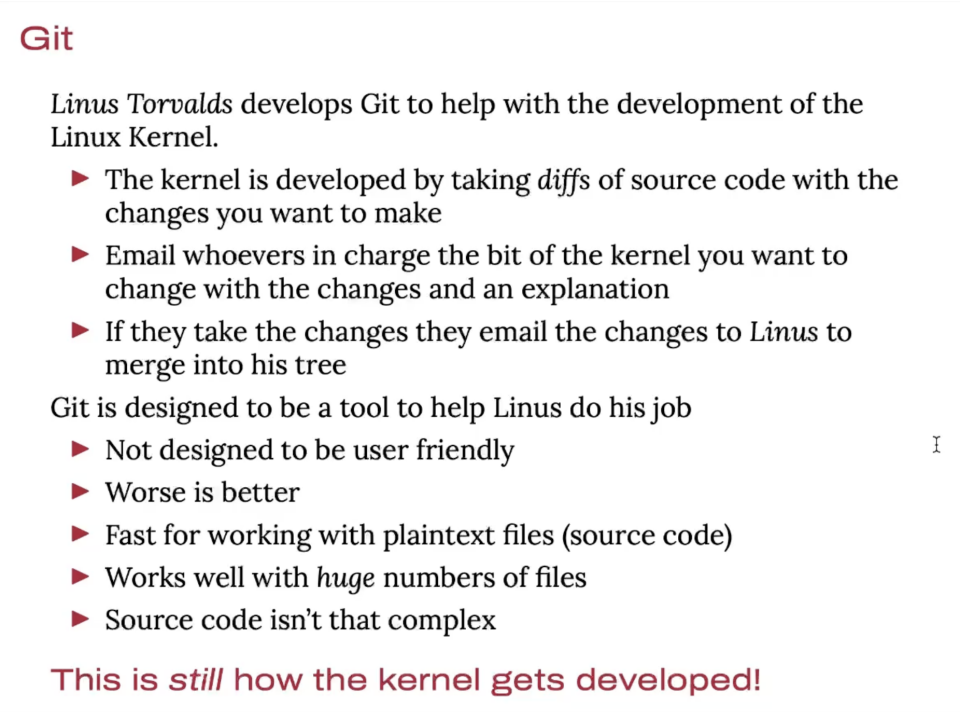
Decentralised Version Control Systems

But around 2000 we start to see a shift away from *centralised* models to decentralised ones

- Every user has a *master* version of the source control
- Changes are accepted from other people through *merges*



- Git is **by far the most popular version control system today**



Getting Started with Git

- to use git we need a git repository
    - select the folder where you want your work to reside:

```
tom@eduroam-54-99 Desktop % cd git_tutorial
tom@eduroam-54-99 git_tutorial % git init
Initialized empty Git repository in /Users/tom/Desktop/git_tuto
```

Instead of adding work that is instantly committed you must first Stage your work

Staging Changes

- instead of adding work and it being instantly comitted, firstly you **stage** your work
    - gather all seperate changes you have made into one place and have them all ready to go at once
- Stage them with the add command:

```
git add .
```

- when you stage a file you're saying this is **going to be** part of a new commit
    - you're adding the changes to Git's Versioning system
    - you haven't saved anything yet! - things can still be changed

When you're happy with the changes you have staged, you commit to them...

Commits

If you're ready to commit:

- everything you've staged so far gets written into the history as a single change
    - with a note explaining it and your name
- things can still change (but its a lot harder to do now)

```
git commit -m 'Initial commit of the program...'
```

- first couple of lines explaining what is happening
- then lets you know how many files were changed etc:

```
[main (root-commit) b377fa3] Initial commit of the greeting program. 1 file changed, 6
insertions(+) create mode 100644 hello.c
```

Tags, Branches, and HEAD

- **ALL** commits are identified by their **hash**
- can name specific commits by using the git tag command:
  - e.g. b377fa3 in the above screenshot
  - what the hash is, is the hash of the changes youve made and the *previous* changes

  - you can **name specific commits** by using the git tag command:

```
git tag
```

  - all commits are made to a **branch** which is a **tag**
    - when a commit is made the branch tag is updated to point to the new commit at the top of the branch
    - the default branch is usually called main (or master)

  there is also a special tag called HEAD

    - HEAD always points to wherever **your code is currently at**
    - minus any unstaged work

  Say you've made lots of changes to a file but not committed them, you'd like to discard these change:

```
git checkout HEAD -- FILENAME
```

  Or if you have changed lots, and want to go back to clean:

```
git reset --hard HEAD // this removes all changes (hard re

git clean -dfx  // deletes all untracked files
```

Say you want to go back to how the code was before the last commit:

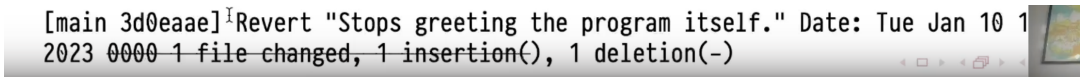
```
git checkout HEAD~1
```

- HEAD is a pointer to our current commit, the ~1 says take us to the state 1 commit before that

Say a commit was a mistake and you want to reverse all the changes it made:

```
git revert HEAD
```

- reversions like this also provide a message:



Top Tips for Git

1. write descriptive commit messages
2. never commit broken code (if it doesn't AT LEAST compile, don't commit)
3. read the man git pages