ENLP Lecture 3:

Git Version Control

Nathan Schneider
12 September 2016

What is version control?

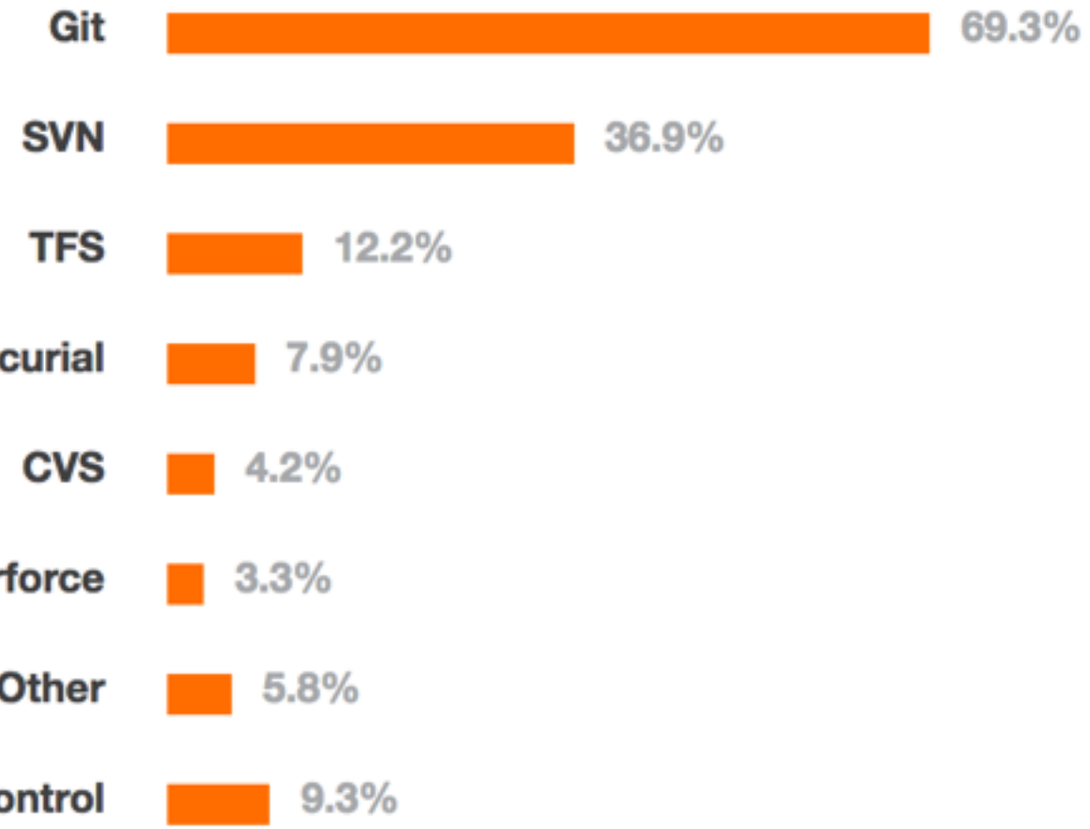
- Software projects can be massive and complex.
  - Hundreds or even thousands of source code files
  - Many developers
- **Version control software** manages this complexity by keeping track of files over time.

Why version control?

- **Keep a history:** Even if you're the sole developer, good to be able to see changes and roll back accidental or problematic ones
- **Multiple developers**
  - Keep track of who made which changes
  - Tools to help reconcile conflicting changes made in parallel
- **Version checkout:** Record the state of the project at a given point in time for testing or release
  - Narrow down to a particular change that introduced a bug

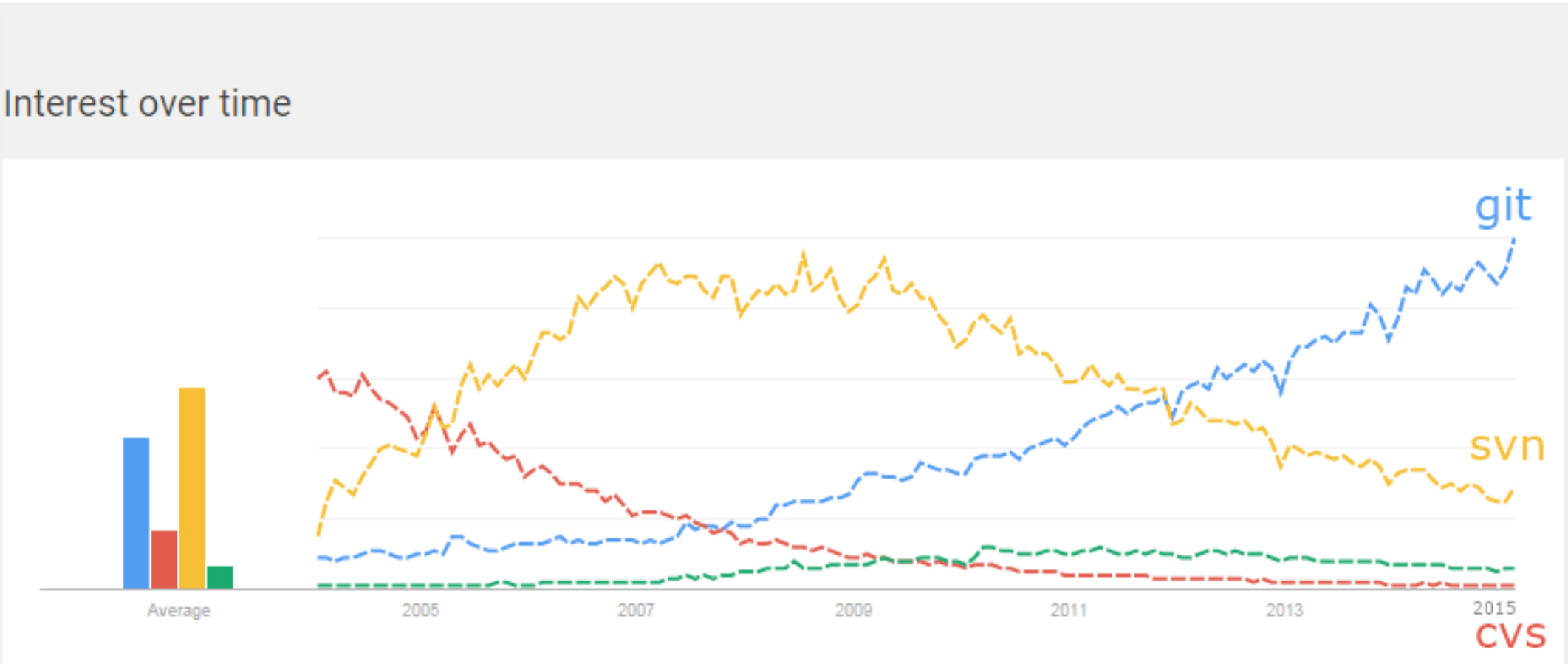
Some popular VCSes

- CVS (really old)
- SVN (old)
- Mercurial
- **Git**



http://stackoverflow.com/research/developer-survey-2015

Some popular VCSes



Google Trends / www.netinstructions.com/the-case-for-git/

Git

- command-line tool: `git`
  - https://github.com/nschneid/git-command-overview
- **GitHub** (github.com) is an extremely popular web host for Git repositories
  - Excellent usability: easy to browse code on the web, open issues for bugs, etc.
  - Public repositories are completely free (encourages open-source development)
- IDEs such as PyCharm have integrated Git support

Git: Fundamentals

- A git repository = a directory where certain files are tracked
  - Metadata (incl. history) is stored in a hidden subdirectory called `.git`

- Git is a **distributed** VCS: there does not need to be a single centralized version of the repo
  - "Cloning" or "forking" a repository makes a complete copy. Changes can then later be **merged** back into the original.

Git: File states

the file differs from git's record of it

`git status` will list such files

untracked — git is ignoring the file

`git add myfile.py`

modified (or deleted) — `git add myfile.py` → **staged** — `git commit -m "note"` → **committed**

git has made note of changes to a file, but not officially recorded them. several files can be staged at once.

`git status` will list such files

git has permanently recorded all staged changes

edit the file

`git commit -a -m "note"` to commit **all** changes to tracked files

Other useful functionality

- **Viewing local changes:** `git diff` [file] to see unstaged changes
- **Viewing history:** `git log` to see commit records, `git log -p` to preview the changes
- **Commit hash:** Each time you commit, the resulting version of the repository is uniquely labeled with a hex string like `9f557835de96d9be5fe5655ebd9186133864354`6
  - **Checking out** an earlier version will replace all tracked files in your file system with the committed files as of that version. (Time travel!)

Other useful functionality

- **Branches:** If you want to work on a new piece of functionality that may take some time, create a new **branch** for those changes. You can make a series of commits on the branch, which keeps them separate from the main version of the code (**master** branch). They can be merged in to the master branch later.

- **Remotes:** If you want to keep your repository in sync with another copy (say, on GitHub), the external location is called a **remote**.

  - `git push` and `git pull` will export and import changes, respectively, to/from the remote repository.

Other useful functionality

- **Stash:** `git stash` temporarily sets aside local (uncommitted) changes without committing them, e.g., if you want to pull somebody else's changes. `git stash apply` then restores the local changes.