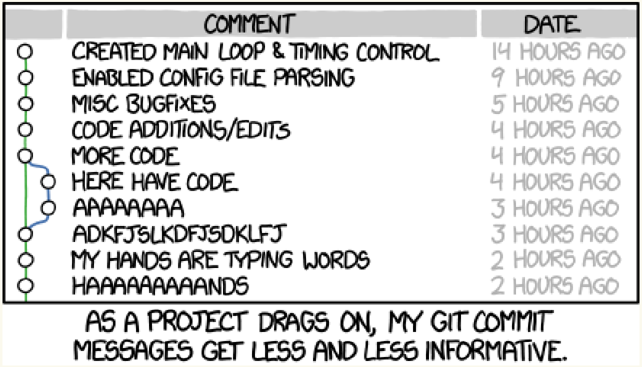
Linux Kernel Programming
Version Control with Git

Pierre Olivier

Systems Software Research Group @ Virginia Tech

March 23, 2017



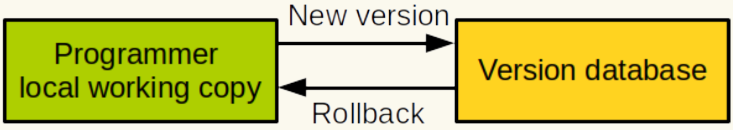
- Source: `https://xkcd.com/1296/`

Outline

Outline

Version Control
Generalities and local VCS

- *Version Control Software*:
  - **Track changes in a codebase**
    - Fast rollback to a previous state when something is broken
    - Easy identification of changes (ex: patch generation)
- Different models:
  1. **Local VCS**
     - ex: GNU RCS

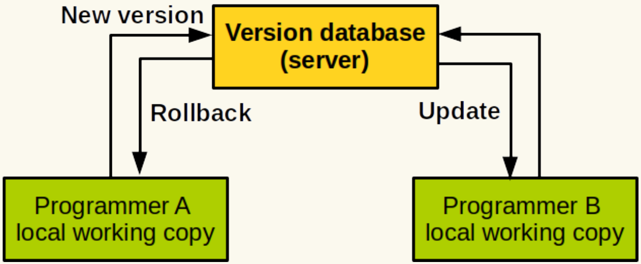


     - Issue: several programmers

Version Control

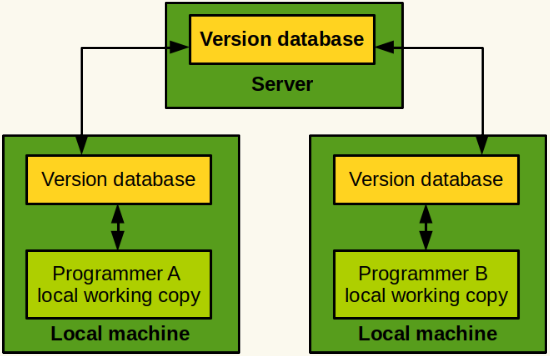
Centralized VCS

- Different models (continued)
  2. **Centralized VCS**
     - Ex: Subversion (SVN)
     - Issue: server is a single point of failure


New version
Version database (server)
Rollback
Update
Programmer A local working copy
Programmer B local working copy

Version Control
Distributed VCS

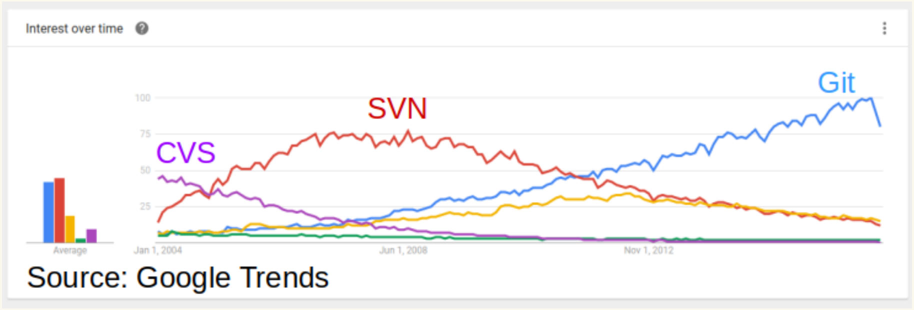
3. **Distributed VCS**
   - Ex: *Git*



Outline

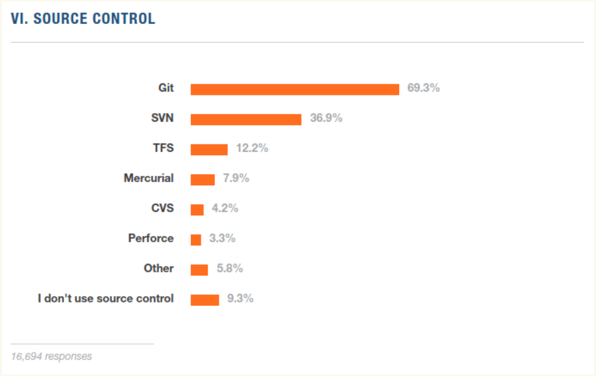
Git: generalities

Git popularity



- `http://bit.ly/2jE50N9`

Git: generalities
Git popularity



- Source: Stack Overflow (`http://stackoverflow.com/research/developer-survey-2015#tech-sourcecontrol`)

Git: generalities

- Development started in 2005 by the kernel community:
  - Replacing *Bitkeeper* that became non-free
- **Fully distributed**
  - Each programmer gets a copy of the entire history
  - Most of git operations happen in local
- **Simple design, fast**
  - Faster than most of the competitors in most of VCS operations (cloning a repository, applying patches, committing changes, etc.) [3]
- **Scalable**
  - Handles large codebases very well (ex: Linux)
  - Allows numerous parallel *branches* to coexist

Git: generalities
Installing & configuring Git

- Install from a Linux distribution repositories:

```
sudo apt-get install git     # Ubuntu / Debian
sudo yum install git         # Fedora / CentOS / RedHat
```

- Install from sources:
  1. Got to `https://www.kernel.org/pub/software/scm/git/` and grab the latest version `git-a.b.c.tar.zx`
  2. Unpack the archive and `cd` to the directory

```
./configure
make
sudo make install
```

    - The `configure` script might indicate you potential missing libraries

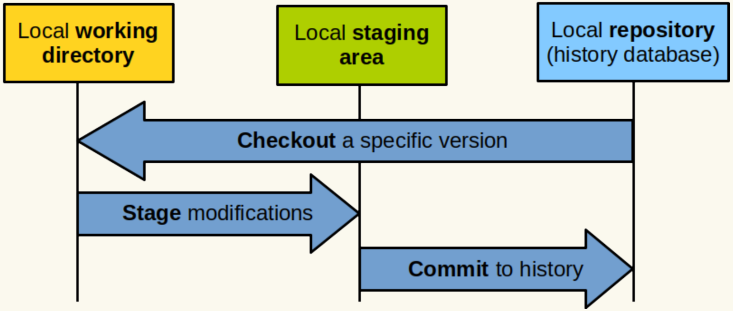
- Minimal configuration:

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

Git: generalities

Git: the three local states

- Git keeps the history database (*repository*) and other metadata in a `.git` folder
  - Hidden folder located at the root of the project directory tree


Local **working directory**
Local **staging area**
Local **repository** (history database)
Checkout a specific version
Stage modifications
Commit to history

- Adapted from [1]

Outline

Basic usage

Cloning a repository from a server

- Copying a remote repository on the local machine: **cloning**
  - Needs a **url** identifying the remote repository
    - Different **protocols** are supported. Examples:
    - `git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git`
    - `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git`
    - `ssh://user@git.kernel.org:/pub/scm/linux/kernel/git/torvalds/linux.git`
  - Contains info on protocol, remote server address, and repository location on the server
- Usage:

```
git clone <url>
```

- More info: `man git clone`
  - Valid for all other git commands referenced here

Basic usage

Checking local copy status

- Status of the working copy is checked through **`git status`**

```
ls
Makefile  my-lib.c  my-lib.h  my-program.c  README

# Modification of my-lib.c and Makefile ...

git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

	modified:   Makefile
	modified:   my-lib.c

no changes added to commit (use "git add" and/or "git commit -a")
```

Basic usage
Preparing new or modified files for commit and committing

- Preparing new or modified files for commit is called **staging** and done through **`git add`**

```
git add Makefile my-lib.c

git status
On branch master
Your branch is up-to-date with 'origin/
     master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to
     unstage)

  modified:   Makefile
  modified:   my-lib.c
```

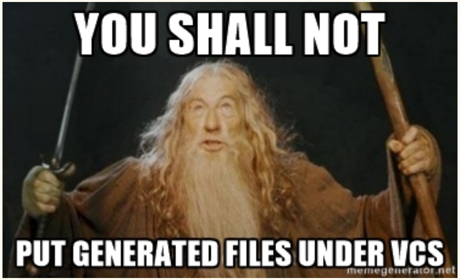
```
touch new-file.txt
git add new-file.txt

On branch master
Your branch is up-to-date with 'origin/
     master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to
     unstage)

  modified:   Makefile
  modified:   my-lib.c
  new file:   new-file.txt
```

- The actual commit is done through **`git commit`**
  - Need to enter a commit message, summary of the changes
  - After that the changes are actually recorded in the local history database

Basic usage
To add or not to add



- Waste of space and bandwidth
- Use the `.gitignore` file

Basic usage

Files: undoing things, renaming

- Remove a file from version control (deletes the file!):

```
git rm <file>
```

  - If the file has local changes or is staged:

```
git rm -f <file>
```

- Remove file from staging area:

```
git reset <file>
```

- Revert local changes (before staging) and rollback a file to the last commit:

```
git checkout <file>
```

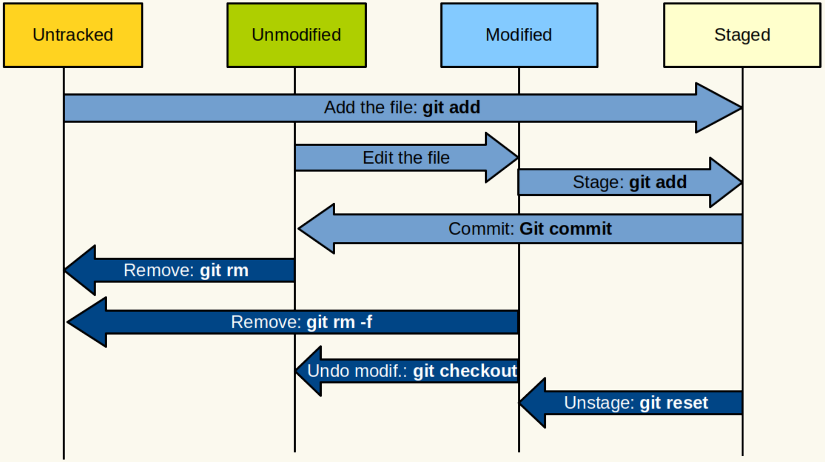
- Rename/move a file under version control:

```
git mv <file>
```

  - Automatically staged

Basic usage

States of a file



Adapted from [2]

Basic usage

Log, rollback

- `git log` displays a log of the commit messages ordered in time

```
commit 45834fb5e08f4e41d37016de54cbdf19872809dc
Author: Pierre Olivier <polivier@vt.edu>
Date:   Wed Jan 11 19:37:52 2017 -0500

    Modified even more stuff.

commit 747982b2bd5f31e1ee1b0997aabe7e0b116fcdf2
Author: Pierre Olivier <polivier@vt.edu>
Date:   Wed Jan 11 19:09:42 2017 -0500

    Modified some important stuff.

commit 7cdfd7cdee05e3306f56d62cd1efcd00f7d8fd58
Author: Pierre Olivier <polivier@vt.edu>
Date:   Wed Jan 11 19:07:08 2017 -0500

    1st commit: initialized some files.
```

- Display for each commit its unique identifier: **hash**
  - Rollback to a previous commit: `git checkout <hash>`
  - Back to the most recent commit: `git checkout <branch>`

Basic usage

Communicating with the server

- Propagate changes to the server: **`git push`**

```
git push
```

  - Sends to the server all the local commits it does not currently contain

- Update local history database from the server: **`git pull`**

```
git pull
```

  - Retrieve commits from other users

Basic usage
Conflicts

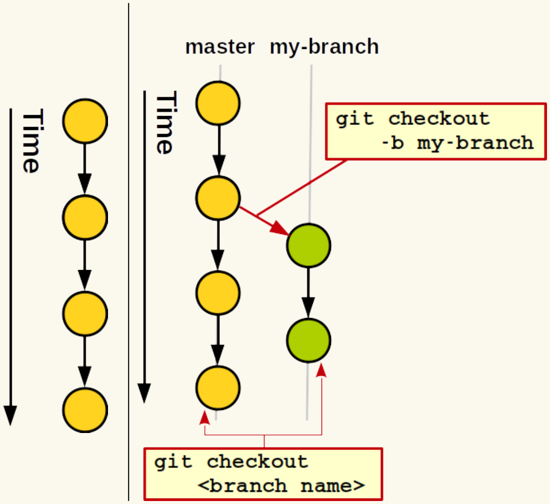
- When the remote commits retrieved through git pull concern file A and there are some non-pushed commits to file A in your local history database:
  - Git first tries to automatically merge the two sets of commits according to some algorithm
  - If this fails, (modified lines are the same or binary file): **conflict**
- **Solving the conflict is needed before completing the pull operation/committing/pushing**
  - Text file:

```
non-conflicting line
another non conflicting line
<<<<<<<< HEAD
line in local working copy
========
line in remote copy
>>>>>>>> <remote commit id>
```

- To solve: edit the file, `add` it then commit

Outline

Branching
What is a branch?



- **Flow of consecutive commits separated from other flows** (other branches)
- Create branch using `git checkout -b <branch name>`
- Switch between branches using `git checkout <branch name>`

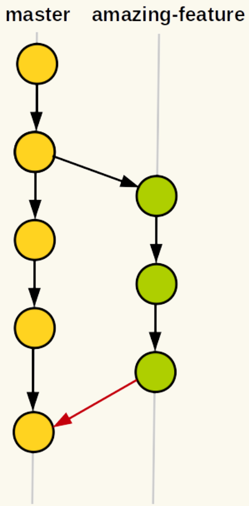
Branching

Why branching?

Branching is useful in multiple cases:

- **Several programmers working on the same codebase**:
  - Per-programmer branches $\rightarrow$ no conflicts
- **Introducing a new feature or a bug fix**
  - Isolate the code related to the feature/bug fix
- **Keeping the `master` branch clean**
  - `master` is the default branch checked out when cloning
  - Development flows are separated into branches
  - `master` is always functional and not in some *work-in-progress* state

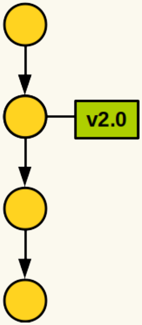
Branching
Merging branches



- **Merging** branch A in B: taking all the differences between
  - applying all the commits of B to the current state of A
  - Conflicts might happen

Branching

Tags



- A **tag** is a snapshot at one specific commit
  - Created through `git tag <tag name>`
  - Used generally to indicate stable versions numbers
  - `git checkout <tag name>`
    - Need to branch to edit from there if needed

Branching
Diffs and patches

- **`git diff`** produces a textual comparison between:
  - Modified files and the last commit: `git diff`
  - Modified files and some specific commit: `git diff <commit hash>`
  - Branches (last commit), tags, specific commits: `git diff <branch/tag/hash> <branch/tag/hash>`
- A **patch** is created by redirecting `git diff` output to a file:

```
git diff v2.0 > modif.patch
```

- To apply a patch on the source commit/branch/tag, put it at the root of the working copy and:

```
patch -p1 < modif.patch
# or:
git apply modif.patch
```

Outline

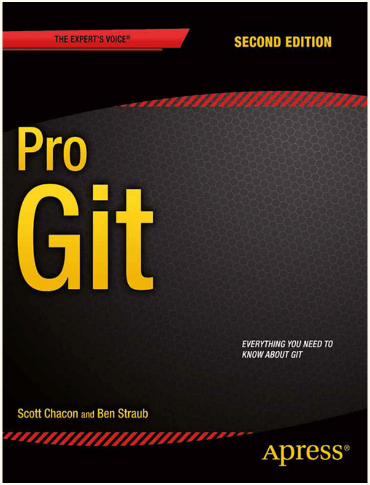
Going further
Misc. information

- **Graphical interfaces:**
  - Github Desktop (Win/Mac), gitk (Linux)
  - `https://git-scm.com/download/gui/linux`
- **Conflicts on binary files:**

```
git checkout --theirs path/to/conflicting/file
git checkout --ours path/to/conflicitng/file
```

- **Working with several remotes servers:**
  - `https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes`
- **Server providers:**
  - GitHub: `https://github.com/`
  - Gitlab: `https://about.gitlab.com/`
  - Plenty of others ...
- **Running your own server:**
  - `https://www.linux.com/learn/how-run-your-own-git-server`

Going further
Documentation


THE EXPERT'S VOICE®
SECOND EDITION
Pro
Git
EVERYTHING YOU NEED TO KNOW ABOUT GIT
Scott Chacon and Ben Straub
Apress®

- Free online:
  - `https://git-scm.com/book/en/v2`
- ISBN-13: 978-1484200773
- ISBN-10: 1484200772

Bibliography I

[1] Git basics.
https://git-scm.com/book/en/v2/Getting-Started-Git-Basics.
Accessed: 2017-01-11.

[2] Git basics: Recording changes to the repository.
https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository.
Accessed: 2017-01-11.

[3] Git benchmarks.
https://git.wiki.kernel.org/index.php/GitBenchmarks.
Accessed: 2017-01-11.