Git – distributed version-control system

Martin Štrambach

Series of Corona seminars

30.3.2020
Motivation

1. I need order in my source files.
2. I need to collaborate with my colleagues.
3. I need to back up my files.
4. I need to fix a broken code which was working a long time ago.
What is Git?

- Initial release 7 April 2005.
- Current version 2.26.0 (22 March 2020).
- Initially developed by Linus Torvalds.
- Developed for Linux kernel versioning\(^1\).
- Originally only low level commands.
- Since version 1.6 reference implementation.

\(^1\)https://github.com/torvalds/linux

Figure Linus Torvalds
Distributed vs centralized version control

Distributed:

- Each repository contains complete history.

Centralized:

- One ground truth.
Distributed vs centralized version control

Distributed:
- Each repository contains complete history.
- Allows private work (fast operations).

Centralized:
- One ground truth.
- Needs internet connection.
Distributed vs centralized version control

Distributed:

▶ Each repository contains complete history.
▶ Allows private work (fast operations).
▶ Git, Mercurial, BitKeeper

Centralized:

▶ One ground truth.
▶ Needs internet connection.
▶ Apache Subversion (SVN), Autodesk Vault
Where to get Git

- **Windows**: download from [https://git-scm.com/downloads](https://git-scm.com/downloads)
- **macOS**: brew
- **Linux**: `apt-get`, `zypper`, etc.
Local repository

- Contains git metadata in .git folder.
- New tracked files can be added via `git commit`.

Initialize empty repository

```
git init
```
Working areas

- working directory
  - `git add`
  - staging area
  - `git commit`
- repository
History and changes

### Status of all working areas
- `git status`

### Changes in working area
- `git diff`

### Changes in stage area
- `git diff --cached`

### History of commits
- `git log [REV]`
Good commit is one logical part of your work.

Commit message should be in English. It’s an imperative sentence which ends with a dot.

No tests should be broken.

Add changes to stage area

```
git add PATH
```  

Create revision

```
git commit
```  

Create revision with a message

```
git commit -m "commit message"
```
Revision specification (REV)

Absolute:
- Hash
- Branch name
- Tag name
- HEAD (symbolic reference)

Relative:
- REV\(^\wedge\) parent of revision
- REV\(^{\wedge\wedge}\) parent of parent of revision
- REV\(^{\sim}\)NUM - NUM commits before REV
Branches

- Branch is a pointer to a commit.
- One branch always exists (implicitly master)

Create a branch

```bash
git branch NAME [REV]
```

List of existing branches

```bash
git branch [-v]
```

Change a branch

```bash
git checkout NAME
```

Create and checkout a branch

```bash
git checkout -b NAME [REV]
```
Merge another branch to the current branch

```
git merge NAME
```

Three possible outcomes:

- Fast-forward
Merge another branch to the current branch

git merge NAME

Three possible outcomes:

- Fast-forward
- Merge without conflicts
Merge another branch to the current branch

```bash
git merge NAME
```

Three possible outcomes:

- Fast-forward
- Merge without conflicts
- Merge with conflicts
Reset changes

git reset [ --hard | --mixed | --soft ] REV

1. Rewinds HEAD reference
2. Updates content
   - soft - discards commits, changes remain in stage area
   - mixed - discards commits, changes are moved to working area (default)
   - hard - discards commits and changes (including uncommitted changes)
Rebase

**Initial history**

A---B---C topic
/
D---E---F---G master

**Rebase command**

git rebase master
or
git rebase master topic

**Altered history**

A---B---C topic
/
D---E---F---G master
Rebase

Initial history

H---I---J topicB
/
E---F---G topicA
/
A---B---C---D master

Rebase command

git rebase --onto master topicA topicB

Altered history

H'--I'--J' topicB
/
| E---F---G topicA
|/
A---B---C---D master
Cherry-pick

Useful for moving changes from one branch to another.

Cherry-pick command

```
git cherry-pick [--edit] REV
```

`--edit` lets you modify commit message
Stash

It is a special branch for stashing changes from working directory away. Works as a stack.

Stash changes

```git stash```

Stash changes only from working area

```git stash --keep-index```

Apply changes from the last stash

```git stash apply```

Apply changes from the last stash and discard the stash

```git stash pop```
Solving conflicts

Conflicts can happen during any operation which moves changes (merge, rebase, cherry-pick, stash, ...).

Solve situation with a conflict
Select correct variant of the code
`git add`
`git OPERATION --continue`

Abort operation with a conflict
`git OPERATION --abort`
Tags

Simple tag

```bash
git tag NAME [REV]
```

Anotated tags

```bash
git tag -a NAME [REV]
```

Local tags must be pushed to remote repository with option `--tags`
Configuration can be specified in a global (---global) or a local (---local) configuration file.

Setup user

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

Setup default text editor

```bash
git config --global core.editor vim
```

Create aliases for commands

```bash
git config --global alias.st status
```
.gitignore

- defines files which are intentionally untracked
- each line defines a pattern
- global vs local (local versioned with the repository)
- * any
- ! negation
- # comment
- \ escape symbol
Remote repositories

Hosted repositories:

- providers like: github.com, gitlab.com, etc.

Repository can be connected either via http/https or interactively via SSH. You can have multiple remotes to one repository. Default remote is called origin.

Clone repository from a remote server

```
git clone URL [DIRECTORY]
```

List all remotes

```
git remote -v
```
Push/Pull

Push changes to a server

```
git push
```

Push changes to a server and set upstream branch

```
git push -u REMOTE BRANCH[:REMOTE_BRANCH]
```

Pull changes from a remote

```
git pull [REMOTE [REMOTE_BRANCH[:BRANCH]]]
```

```
git pull does git fetch and git merge of a remote branch to the local one.
```

Force push to remote (changes history)

```
git push --force
```

## Useful commands

**Pretty print of history**

```bash
git log --oneline --graph --decorate --all
```

**Find a breaking commit**

```bash
git bisect start
git bisect bad
git bisect good REV
```

**Add patch interactively**

```bash
git add -p [PATH]
```

**Interactive rebase**

```bash
git rebase -i REV
```
Workflows

- Feature branches
- Develop branches
- Release branches
- Hotfixes
- Master branches

- Time

- Feature for future release
- Major feature for next release
- Incorporate bugfix in develop
- Severe bug fixed for production: hotfix 0.2
- From this point on, “next release” means the release after 1.0
- Only bugfixes
- Bugfixes from rel. branch may be continuously merged back into develop
- Start of release branch for 1.0
- Tag 0.1
- Tag 0.2
- Tag 0.1
- Tag 1.0
Questions?

Martin Štrambach
strammar@fel.cvut.cz