GitLab at Faculty of Informatics

Roman Lacko

xlacko1@fi.muni.cz
Faculty of Informatics
Masaryk University

3. 10. 2018

Git and GitLab

A brief introduction

Git and GitLab

Properties and Goals

- **version control system**
- distributed
- snapshot-based (as opposed to *delta-based*)
- history authentication

Goals

- speed
- simplicity
- non-linear development ("*branching*")
- full distribution
- scalability

Git and GitLab

A very brief history of Git

Linux Kernel VCSs

1991 – 2002	patches and archives
2002 – 2005	BitKeeper (propriertary DVCS)
since 2005	Git

- first release **7 April, 2005**
- written in C, Bash, Perl, ...

Why Git?

`man git`: *the stupid content tracker*

- Directed Acyclic Graph

Git and GitLab

A very brief history of Git

Linux Kernel VCSs

1991 – 2002	patches and archives
2002 – 2005	BitKeeper (propriertary DVCS)
since 2005	Git

- first release **7 April, 2005**
- written in C, Bash, Perl, ...

Why Git?

`man git`: *the stupid content tracker*

- Directed Acyclic Graph
- Key-Value Database

Git and GitLab

Git Repository Managers

Additional features on top of Git repository

- authentication
- access control
- collaboration mechanisms
  e.g. fork, pull request

Software development tools integration

- issue tracking
- documentation (wiki)
- automatic build and deployment

- **open source**: **GitLab**, Gitolite, Gerrit, ...
- **proprietary**: GitHub, BitBucket, ...

Git and GitLab
GitLab

Features

- groups
- Markdown and AsciiDoc wiki (another Git repository)
- static page generator
- issue tracking, boards, milestones
- continuous integration
- web IDE
- push policies
- ...

Git and GitLab

GitLab FI

`https://gitlab.fi.muni.cz/`

- **GitLab Ultimate**
- virtual machine in Stratus.FI cloud
- 4 VCPU, 8 GiB RAM
- 256 GiB repositories (57 GiB used)

- 5600 projects
- 1900 users (1200 active)
- 40 groups

Git Workflows

Collaboration guidelines

Git Workflows
Best practices

Divide project into several repositories

- plan ahead
- later splitting is usually painful

Git Workflows

Best practices

Divide project into several repositories

- plan ahead
- later splitting is usually painful

Commit often

- commit only related changes
  ```
  git commit -p
  ```
- do not commit large chunks
- do not commit untested work
- write short but descriptive commit messages

Git Workflows

Best practices

Git is not a backup system

- do not keep unrelated files
- do not commit generated files
- avoid storing large files
  Git LFS

Git Workflows

Best practices

Git is not a backup system

- do not keep unrelated files
- do not commit generated files
- avoid storing large files
  Git LFS

Learn to use Git's safety belts

- most **unpublished** mistakes are recoverable
- `git commit --amend`
- `git revert`
- `git reflog`

Git Workflows

Best practices

Keep the history clean

- avoid unnecessary merges
- `git pull --rebase`
- `git stash`

Git Workflows
Best practices

Keep the history clean

- avoid unnecessary merges
- `git pull --rebase`
- `git stash`

Do not change published history

- `git push --force`
- use *protected branches* feature if possible
- scorn people who break this rule

Git Workflows

Miscellaneous Tricks

- set `user.name` and `user.email` in Git configuration

```
git config KEY VALUE
```

- make the initial commit *empty*

```
git commit --allow-empty
```

- use and maintain the `.gitignore` file

Git Workflows

What is a workflow?

- set of rules for managing the repository
- useful for collaborative projects
- no universally best strategy

Categorization aspects

- branching model
- distribution model

Git Workflows

Branching Models

Single Branch

- there is only one official branch, `master`
- developers can have local (private) branches
- branch cleanup before merge

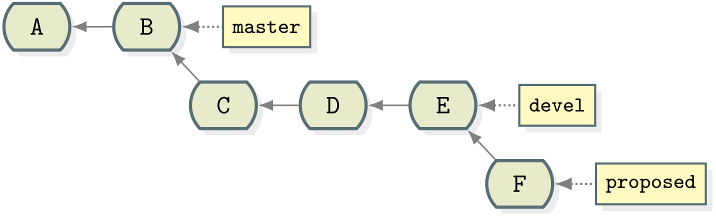
  rebase, squash, amend

Git Workflows

Branching Models

Long-Running Branches

- branches represent different levels of stability
- `master` – stable branch
- `devel` – "*bleeding edge*" features
- `proposed` – untested features
- ...

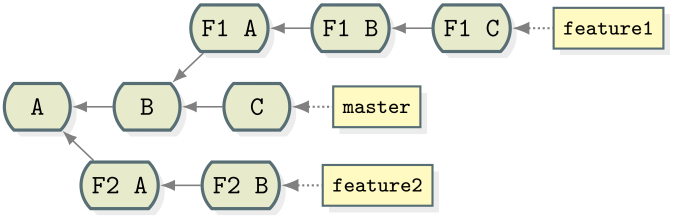


Git Workflows

Branching Models

Topic or Feature Branches

- branches represent different features
- merged into `master` on completion
- sometimes uses *rebasing* instead of *merging*

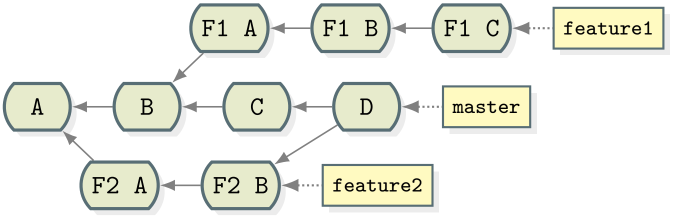


Git Workflows

Branching Models

Topic or Feature Branches

- branches represent different features
- merged into `master` on completion
- sometimes uses *rebasing* instead of *merging*

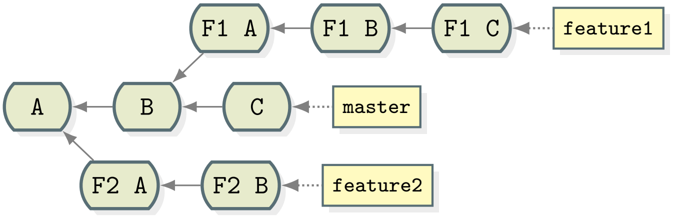


Git Workflows

Branching Models

Topic or Feature Branches

- branches represent different features
- merged into `master` on completion
- sometimes uses *rebasing* instead of *merging*

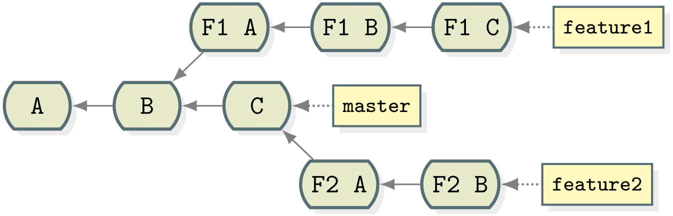


Git Workflows

Branching Models

Topic or Feature Branches

- branches represent different features
- merged into `master` on completion
- sometimes uses *rebasing* instead of *merging*

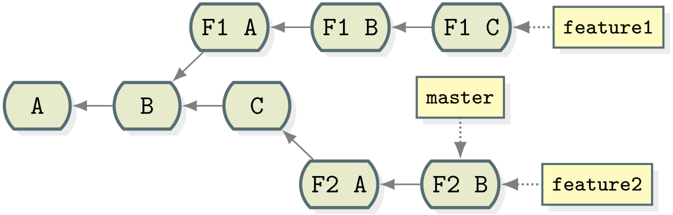


Git Workflows

Branching Models

Topic or Feature Branches

- branches represent different features
- merged into `master` on completion
- sometimes uses *rebasing* instead of *merging*



Git Workflows

Distribution Models

Centralized Repository

- *one repository to rule them all*
- every member pulls and pushes to a single repository
- the simplest strategy

- works for small teams
- projects migrated from different VCSs

Git Workflows

Distribution Models

Integration Manager

- a single *official* repository
- developers have *public* and *private* clones

- new features are published in public repositories
- official repository maintainer is asked to pull changes

- easier with repository managers
- repository forks
- pull requests (GitHub), merge requests (GitLab)

Git Workflows
Distribution Models

Dictator and Lieutenants

- optimized for huge projects, e.g. Linux Kernel
- mostly combined with Feature Branches Models

- developers work in feature branches
- lieutenants merge these branches on their own `master`
- the dictator merges lieutenants' `master` into his own
- the dictator pushes his `master` to the official repository
- developers rebase their branches on top of new `master`

Git Workflows
Examples

Master Only Workflow

- Single Branch + Single Repository
- project maintainer approves changes

GitHub Workflow

- Feature Branches + Single Repository
- code review before merging to `master`

Git Workflows
Examples

GitFlow

- combination of Long-Running Branches and Feature Branches
- usually Single Repository model

- `master` – deployed in production, hotfixes
- `release` – stable code
- `devel` – approved features

- feature branches based on `devel`
- hotfix branches based on `master`
  must be merged to `release` and `devel` as well

GitLab Repository

Basic setup and features

GitLab Repository

Project path

```
https://gitlab.fi.muni.cz/PATH/NAME.git
ssh://git@gitlab.fi.muni.cz:PATH/NAME.git
```

- project path should be chosen carefuly
- consider group namespaces for long-term projects
- projects referenced from theses or papers

Personal namespaces are **not** permanent.

GitLab Repository

Sharing

Visibility

- easiest sharing option
- *Private* (default in GitLab FI), *Internal* or *Public*

Members and Groups

- fine-grained access control
- members with different roles
- *Maintainer*, *Developer*, *Reporter* or *Guest*

- project can be shared with a group
- this does **not** move the project to a different namespace

GitLab Repository
Secure Shell Access

User (SSH) Key

- grants access to all projects of the user
- unique in the entire GitLab instance
- unsuitable for automated repository access

Deploy Key

- limited to project scope
- can be enabled for more than one project
- designed for automatic deployment

GitLab Repository

Repository policies

Voluntary security features

Push options

- enforce verified commiter e-mail
- require digital signature
- commit message requirements

Branch and Tag protection

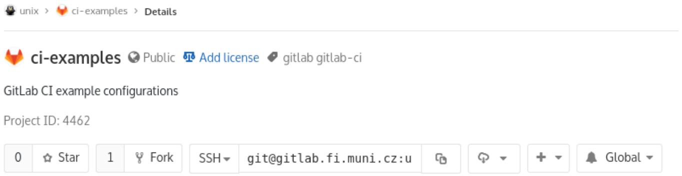
- disallow developers to **force** push
- prevent unauthorized merge into protected branch
- protect specific tags

GitLab Repository

Workflow support

Repository fork

- clones the project to the user's namespace
- *upstream - downstream*


unix › ci-examples › Details

ci-examples Public Add license gitlab gitlab-ci

GitLab CI example configurations

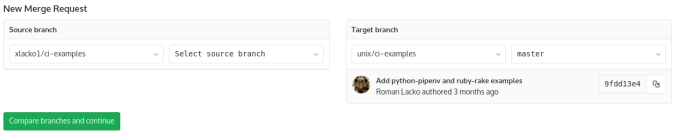
Project ID: 4462

GitLab Repository

Workflow support

Merge request

- notifies the developer
- allows code review
- support for automatic testing
- variety of merge strategies


New Merge Request

Source branch: xlacko1/ci-examples — Select source branch

Target branch: unix/ci-examples — master

Add python-pipenv and ruby-rake examples
Roman Lacko authored 3 months ago
9fdd13e4

GitLab API

Scripting and task automation

GitLab API
Overview

HTTP-based RESTful API

- HTTP/2.0
- REST properties
- JSON data

- endpoint: `https://gitlab.fi.muni.cz/`**`api/v4`**

GitLab API

Usage

Authentication

- Personal Access Tokens
- Session Cookies
- Impersonation Tokens (administrators only)

Direct access

```
$ curl -L -H 'Private-Token: <...>' https://gitlab.fi.muni.cz/api/v4/projects
```

GitLab API

Clients

- implementations for various languages
- Ruby, Python, Perl, Java, .NET, ...
- easy intergration into other tools

```
use GitLab qw(:project_visibility);

my $gitlab = GitLab::API->new(
   Host         => "gitlab.fi.muni.cz",
   AuthToken    => "************",
);

foreach my $login (get_students) {
    my $project = $gitlab->project_by_id(id => "$login/pb161");

    if ($project->{visibility} ne PROJECT_PRIVATE) {
        print STDERR "$login busted!\n";
    }
}
```

GitLab API

Major applications at FI

Student homework repository audit

- project visibility
- members
- events
- commit authors

Integration with FI services

- group and subgroup members synchronization
- repository quotas
- external accounts
- blocked accounts

GitLab Continuous Integration

Webhooks, automatic tasks, tests and deployment

GitLab Continuous Integration

Webhooks

- trigerred on certain events (push, new issue, build fail...)
- user-defined HTTP callbacks

- `POST` request with event details
- token and SSL verification
- branch filtering

Applications

- course web generators (PB071, PB161, PV264, ...)
  static webpage generators (Jekyll, Hakyll)
- build fail notifications
- group membership on first login
  system webhooks

GitLab Continuous Integration

GitLab CI Runners

- dedicated services
- *specific*, *group* and *shared*
- periodical requests for jobs

- `.gitlab-ci.yml` configuration file
- describes tasks for the runner

- project build
- integration tests on merge requests
- deployment

GitLab Continuous Integration

Specific and group runners

- accept jobs from given projects and groups
- optimized for target projects

- easy to set up
- can run on workstations and notebooks

GitLab Continuous Integration

Specific runner

`gitlab-ci.fi.muni.cz`

- Docker containerization
- variety of images available
- new or custom images on demand

- requires `shared-fi` project tag

GitLab Continuous Integration

Applications

- lab-specific tasks
- homework testing
- webpage deployment
- `unix/ci-examples.git` repository

References

Uncovered features, additional resource

References

Advanced features

- Issue tracker, Boards, Milestones
- Epics
- Pages
- OmniAuth
- Container Registry
- Kubernetes

- new features are still being developed

References

Git

- `man gittutorial`
- Pro Git Book (2nd Edition)
- Git Best Practices
- Git-Tower Best Practices

Git Workflows

- `man 7 gitworkflows`
- Atlassian BitBucket Workflows
- Git Workflow Guide
- ...
- just google `git workflows`

References

GitLab

- GitLab Feature List
- GitLab API Documentation
- GitLab User Documentation

FI resources

- Technical Information on GitLab
- Technical Information on Stratus.FI Cloud

References

Try GitLab

- `https://hub.docker.com/u/gitlab/`
- GitLab Docker Image
- free CE version
- 30 days evaluation EE version