

pCT Bergen Meeting

Gitlab best-practice

Matthias Richter

Jan 12 2017

Outline

- Part 1: A short introduction
- Part 2: Hands on

Very much my personal view and experience. There is a lot of information available and we have to create our strategy and build our knowledge.

Policies

Keep track of your development

- make frequent commits
- write meaningful commit and log messages
- use branches to structure your work, e.g. different features, prototyping, ...

Publish early, publish often

- Make code available to your colleagues
- Participate in and receive code reviews

Git: General Remarks

Versioning system, strong focus on distributed repositories

Git separates a contribution to a project into three stages

- ① `git add`: add new files to project or add changed files to be committed
- ② `git commit`: check in changes to local working clone
- ③ `git push`: push status of local clone to a remote repository

... and adds a new concept

- `git pull`: pull updates from (any) remote repository

Gitlab - a smart interface to Git

Gitlab is an interface to `git`
not only repository hosting, but allows efficient collaborative work

- Easy code sharing and discussions
- Code review
- Documentation
- Bug and issue tracking
- Planning
- Distributed maintaining, every developer can take over tasks from the main maintainers
- ... and a lot more

Working with Gitlab

There are two different roles:

- **User role:** A *user* wants to download the code, compile it and use it.
- **Developer role:** A *developer* contributes to the development.

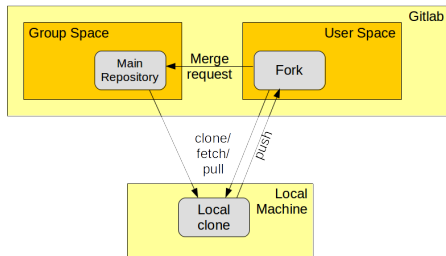
Often one starts as a user and moves to developer role later

Gitlab supports multiple repository copies

- git supports/implements distributed repositories copies
- git itself does not even make an assumption about a master repository
- It is however good to have such a master repository and dedicated strategy how to contribute to it.

User space in Gitlab/Github

- Every user has his/her own space at the server side, in addition there can be groups with dedicated group space
- A project is created in a user or group space
This is usually the master repository
- Repositories can be *forked*, this creates a repository copy inside Gitlab with dedicated access rights



The developer fork is the main feature in Gitlabs/Githubs concept of distributed development, code sharing and code review.

The repository copies are synchronized via *merge requests*

The pCT group at UiB Gitlab

<https://gitlab.uib.no/pct>

Projects in the Gitlab pct group

Initial projects:

- Start with a project per work package (WP n)
- There will be significant overlap between work packages, we might want to consider renaming and a different structure early next year
- e.g. WP1 and WP7 call for a common repository (common package dependencies, data formats)

External projects:

- External projects can either be taken directly from a remote repository, or can be imported as a separate project in our Gitlab space
- “Lyon”-code imported to <https://gitlab.uib.no/pct/mlpTracking> (name might change)
- Work-flow proposal on the wiki: https://wiki.uib.no/pct/index.php/Documentation#Importing_an_external_package

Branches in the main repository

Suggestion:

- **production**: the latest production code, in this branch we have release tags
- **master**: the latest stable release
- **dev**: the development branch

In addition to those main branches there can be feature branches where development happens detached from the main branches.

A feature branch is based on the dev branch and has a limited lifetime.

Tutorial

Gitlab group page

<https://gitlab.uib.no/pct>

The screenshot shows the GitLab group page for 'pct'. At the top, there is a navigation bar with the group name 'pct', a search box, and navigation links for 'Group', 'Activity', 'Labels', 'Milestones', 'Issues', 'Merge Requests', and 'Members'. Below the navigation bar is a profile section for '@pct' with a question mark icon and buttons for 'Leave group' and 'Global'. The main content area is titled 'All Projects' and includes a 'Filter by name' input field, a 'Last updated' dropdown menu, and a 'New Project' button. A list of projects is displayed, each with a colored circle icon, a name, a description, and a lock icon on the right.

Project Name	Description
tutorial	A tutorial project
wp4	Online systems Hardware Infrastructure, Software Infrastructure, DAQ software, DCS + Trigger, Data quality monitoring
wp7	Reconstruction software Calorimeter response, Calorimeter track reconstruction, Reconstruction of 3D trajectory – track vector matching, 3D stopping power ...
mlpTracking	The most-likely-path tracking code of the Lyon group
wp6	Commissioning Commissioning of the PRM in beams, Performance evaluation in a pre-clinical environment, i.e. with phantoms
wp5	Assembly and System integration Assembly of chips into HICs/staves, Assembly of staves into layers, Integration of layers into a compact detector, Mechanical a...
wp3	Data readout. Optimisation of the readout electronics architecture, SystemC simulation of rates, Development and testing of readout electronics, Setting up a f...
wp2	Chip submission and sensor characterization Improved sensor and data encoding design, Chip submission, Testing of prototypes
wp1	WP1: Physics simulation, verification and design optimization Beam scenarios Detector specifications Radiation environment Evaluation of rates Beam tests of ...

Note:

- Log in with UiB account and request access

Cloning a project

<https://gitlab.uib.no/pct/tutorial>

☰ pct/tutorial 🔍 This project: Search 🔔 🌐 ⚙️

Project Activity Repository Pipelines Graphs Issues 0 Merge Requests 0 Wiki

You pushed to **dev** 4 minutes ago

Create Merge Request



tutorial

A tutorial project

☆ Star 0 🍴 Fork 0 HTTPS https://gitlab.uib.no/pct/tutc 📄 📁 + 🌐 Global

Files (120 KB) Commit (1) Branches (2) Tags (0) Add Changelog Add License Add Contribution guide Set up CI

62590847 Initial commit - 7 minutes ago by Matthias Richter

This is a tutorial project

```
git clone https://user@gitlab.uib.no/pct/tutorial.git
```

Note:

- make sure to switch the link to `https` before cloning
- currently investigating cloning via `ssh`

Using the cloned project

Some commands to investigate the project:

```
cd tutorial
git branch
git branch -a # to show all branches
git remote -v # shows the remote repository
```

Example output:

```
richter@workhorse:~/src/pCT$ cd tutorial
richter@workhorse:~/src/pCT/tutorial$ git branch
* master
richter@workhorse:~/src/pCT/tutorial$ git branch -a
* master
remotes/origin/HEAD -> origin/master
remotes/origin/dev
remotes/origin/master
richter@workhorse:~/src/pCT/tutorial$ git remote -v
origin https://gitlab.uib.no/pct/tutorial.git (fetch)
origin https://gitlab.uib.no/pct/tutorial.git (push)
```

Note:

- Git configures the remote upstream repository, the original cloned repository gets identifier *origin*
- Multiple upstream repositories are possible

Found a bug - create an issue

<https://gitlab.uib.no/pct/tutorial/issues> → **New Issue**

Project Activity Repository Pipelines Graphs **Issues 0** Merge Requests 0 Wiki

This project: Search

New Issue

Title:

Add [description templates](#) to help your contributors communicate effectively!

Description: **Write** Preview

I can not compile the project, I'm using the following commands
...
[Logfile attached](#)

Styling with [Markdown](#) and [slash commands](#) are supported. [Attach a file](#)

This issue is confidential and should only be visible to team members with at least Reporter access.

Assignee: Due date:

Milestone:

Labels:

Note:

- Gitlab issues can also be used for feature requests and ideas

Becoming a developer (1)

Switching roles from *user* to *developer* involves a couple of changes:

- Development involves two “git”-stages:
 - ① Local changes to the work clone: `git add/commit`
 - ② Publish to server: `git push`
- **Rule:** always use branch `dev` for development or an appropriate feature branch: `git checkout dev`

Example:

```
git checkout dev # a new branch can be created with git checkout -b branchname
# do some modifications to existing file or create new file
echo "this is ${USER}'s new feature" > feature_${USER}.txt
# add to the index for committing
git add feature_${USER}.txt
# commit
git commit -m "A new feature by ${USER}"
```

Note: everything is local until now

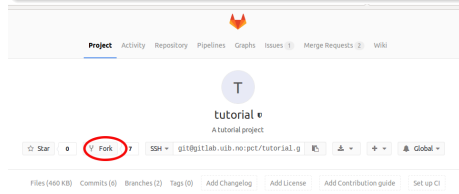
Becoming a developer (2) - Making a fork

A repository fork is a copy of the original repository **within Gitlab**

Why forks?

- a fork is the users workspace in Gitlab (it's not a local clone)
- user has full access to a fork (other than the master repository)
- fork makes development public within the team and makes code sharing and review easy
- fork allows user adjusted code

Looks exactly like master repository, but you have full access and control



Klick "Fork"-button and choose where to fork the project to, in most cases your user space.

Becoming a developer (3) - Adjusting work clone

Your developer space at Gitlab (adjust *user*)

```
https://gitlab.uib.no/user
```

- If your clone was done from the main repository → set upstream repository link to fork (adjust *user*)

```
git remote set-url origin https://user@gitlab.uib.no/user/tutorial.git  
git remote update
```

- Or make a clone directly from the fork

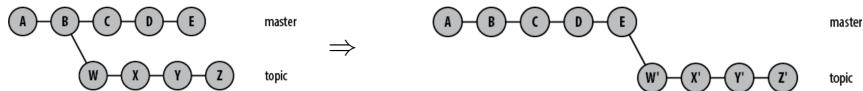
```
git clone https://user@gitlab.uib.no/user/tutorial.git
```

- Make a contribution by pushing code updates to fork

```
git push origin dev
```

Updating the local clone - *rebase*

rebase is a synchronization of branches



Sequence of commits since branching point is “replayed” on top of the base branch.

- This results in a sequence of **new commits**.
- Linear commit history, no merge commits

Example: Synchronize sequence of development with main repository

- pull latest state of branch *dev* from main repository:

```
git fetch https://gitlab.uib.no/pct/tutorial.git dev
```

- rebase branch *dev* of local clone to the remote branch

```
git rebase FETCH_HEAD dev
```

See also <https://wiki.uib.no/pct/index.php/Documentation#Preparation>

Rebase: Resolving merge conflicts (1)

A merge conflict arises from conflicting changes from different lines of development, e.g. changes in the same file at the same position.

- Git produces merged version with some special notation
- rebase is stopped, manual intervention is needed.

Example:

```
richter@workhorse:tutorial$ git diff
diff --cc README.md
index 2fdc258,b99ee05..0000000
--- a/README.md
+++ b/README.md
@@@ -1,4 -1,5 +1,9 @@@
    This is a tutorial project

++<<<<<<< 2005439cf1c7250d17dc85281b79b69f56487335
    +THIS NEEDS SOME MORE INFORMATION
++=====
+ some changes by Matthias
+
++>>>>>>> Some changes by Matthias
```

Changed lines are indicated by two columns with +/-, one column for each versions starts with what has changed in the base version and your changes follow after the separator =====

Rebase: Resolving merge conflicts (2)

Looking at changes: the `--ours` and `--theirs` options

diff wrt your version

```
richter@workhorse:tutorial$ git diff --theirs
* Unmerged path README.md
diff --git a/README.md b/README.md
...

+<<<<<<< 2005439cf1c7250d17dc85281b79b69f56487335
+THIS NEEDS SOME MORE INFORMATION
+=====
+some changes by Matthias

+>>>>>>> Some changes by Matthias
```

diff wrt base version

```
richter@workhorse:tutorial$ git diff --ours
* Unmerged path README.md
diff --git a/README.md b/README.md
...

+<<<<<<< 2005439cf1c7250d17dc85281b79b69f56487335
+THIS NEEDS SOME MORE INFORMATION
+=====
+some changes by Matthias
+
+>>>>>>> Some changes by Matthias
```

Note: This is a counter-intuitive feature of git, *rebase* is done from the perspective of the base branch, not the developer branch which is rebased. As a consequence, `--ours` refers to the version in the base branch and `--theirs` to the developer version (your version).

Rebase: Resolving merge conflicts (3)

Options to resolve the conflict:

- 1 Edit the file
- 2 Take your version

```
git checkout --theirs -- filename
```

- 3 Discard your version

```
git checkout --ours -- filename
```

Next steps:

- indicate to git that you have done the work by *adding* the file to the index for the commit. Note: don't use `git commit`.

```
git add filename
```

- continue the rebase process (this does the commit)

```
git rebase --continue
```

Emergency:

- you can always abort and restore the original state

```
git rebase --abort
```

Create a merge request

Push the updated version to the fork

- since the commit line has been changed, the local and remote branches are out of sync, option `-f` (force) has to be used

```
git push -f origin dev
```

Merge request is now created on the gitlab web interface of your fork

- 1 Go to your gitlab user space at <https://gitlab.uib.no/user> (replace user appropriately).
- 2 Find the project fork, e.g. in the list of projects associated with you from the upper main menu.
- 3 In the line with the many columns regarding the repository, click on the "+"-symbol on the right hand side and choose "New merge request"
- 4 Select project and branch for both source and target, and click "Compare branches and continue". Remember: in almost all cases you have to merge to branch dev or other feature branch, only in very rare cases to branch master
- 5 Review the list of commits in this merge request, give it a descriptive title and description, pick an assignee
- 6 Submit the merge request

Additional discussion during the session

Some notes from the discussion

- Guidelines for using Gitlab (*“this is how we use our gitlab space”*)
There is already some information in the wiki, more work needed to complete the guidelines

https://wiki.uib.no/pct/index.php/Documentation#Software_repository

https://wiki.uib.no/pct/index.php/Gitlab_best_practice

- Handling of documentation: MS Word widely used
Proposal: doc files as binary files, use versioning in the doc files, consider separate project which is not to be forked
- Handling of binary files, in particular large binary files, check what is needed