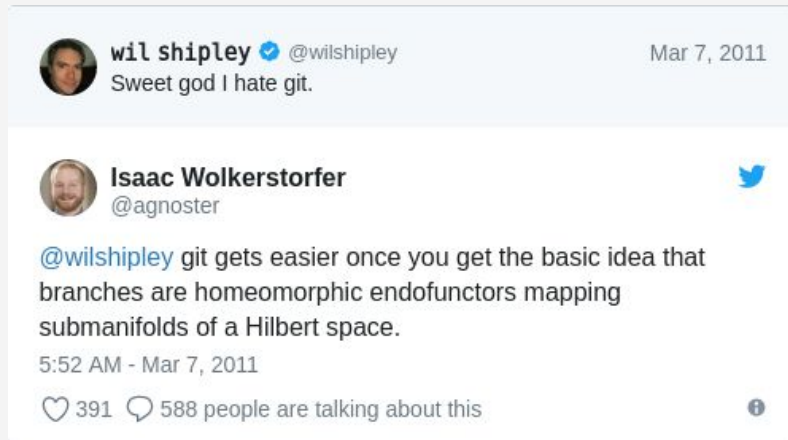


# Git(Lab) Tutorial and Hands-On

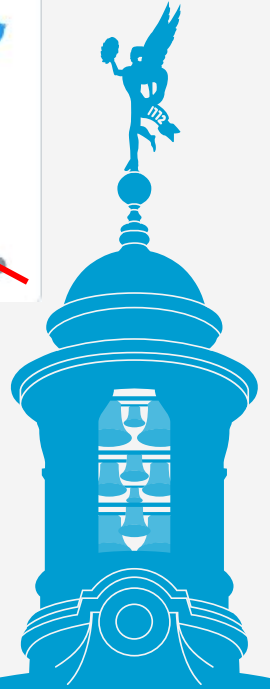
Alexander Fulst



# Git(Lab) Tutorial and Hands-On

Or: Why branches aren't homeomorphic endofunctors mapping submanifolds of a Hilbert space

Alexander Fulst

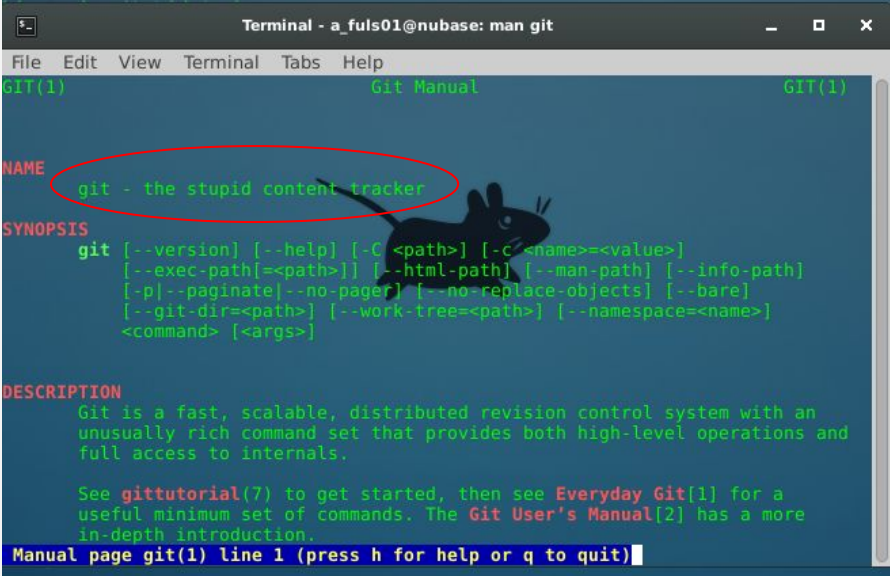


# What is Git?

From <https://en.wikipedia.org/wiki/Git>

Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source code management in software development, but it can be used to keep track of changes in any set of files. As a distributed revision control system it is aimed at speed, data integrity, and support for distributed, non-linear workflows.

Output of *man git*



```
Terminal - a_fulst01@nubase: man git
File Edit View Terminal Tabs Help
GIT(1)                               Git Manual                               GIT(1)

NAME
  git - the stupid content tracker

SYNOPSIS
  git [--version] [--help] [-C <path>] [-c <name>=<value>]
  [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
  [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
  [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
  <command> [<args>]

DESCRIPTION
  Git is a fast, scalable, distributed revision control system with an
  unusually rich command set that provides both high-level operations and
  full access to internals.

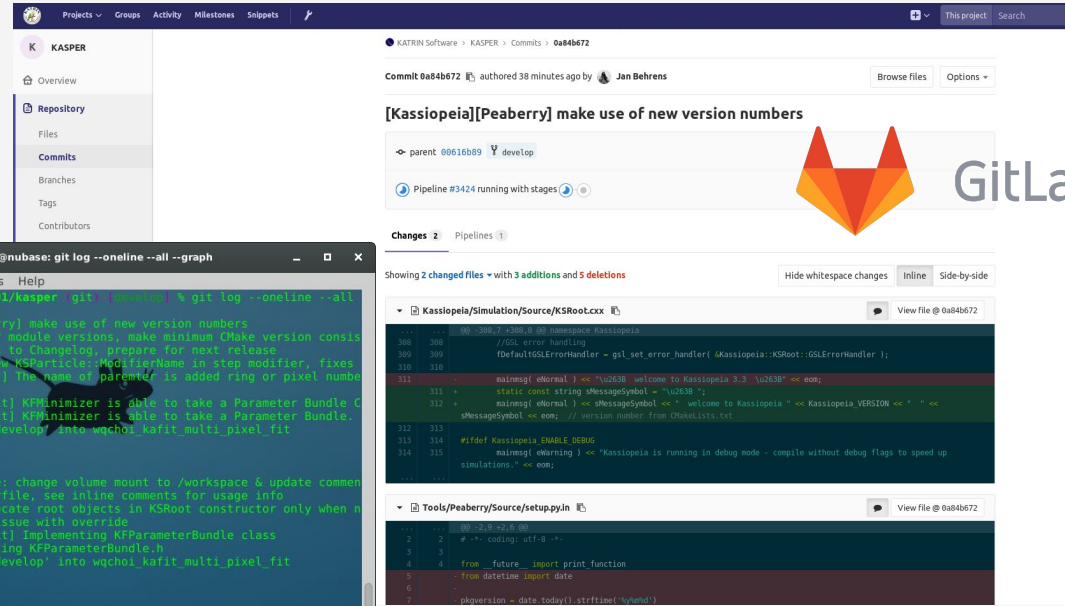
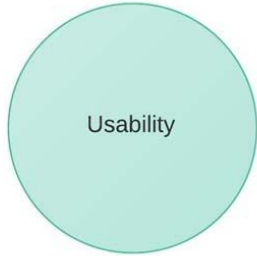
  See gittutorial(7) to get started, then see Everyday Git[1] for a
  useful minimum set of commands. The Git User's Manual[2] has a more
  in-depth introduction.

Manual page git(1) line 1 (press h for help or q to quit)
```

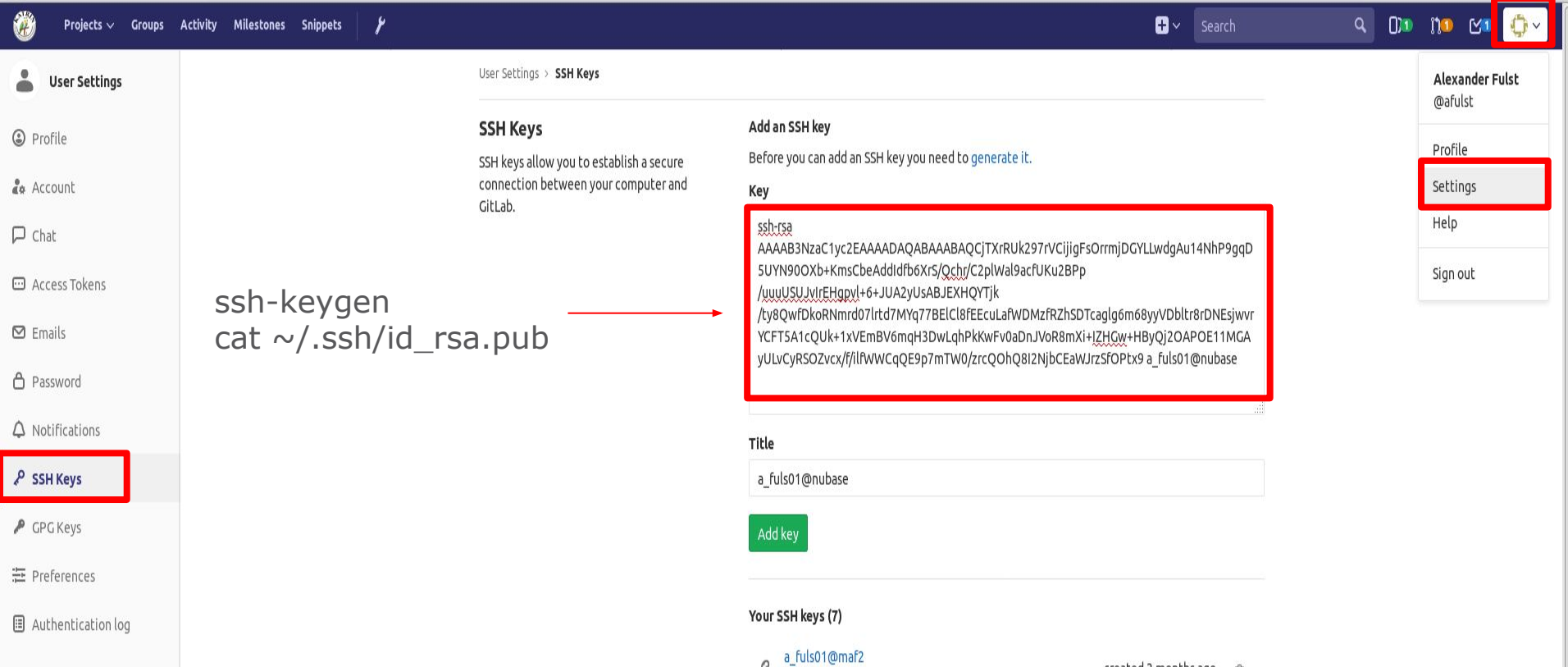
# Two Ways to use Git

A mix of command line and web interface

Exclusively from command line



The screenshot shows the GitLab web interface for a repository named 'KASPER'. The commit being viewed is '0a84b672' by 'Jan Behrens', titled '[Kassiopeia][Peaberry] make use of new version numbers'. The commit message includes details about version handling and KFit modifications. The interface shows a diff for 'Kassiopeia/Simulation/Source/KSRoot.cxx' and 'Tools/Peaberry/Source/setup.pyin', with line numbers and changes highlighted. The commit history sidebar on the left shows a list of recent commits with their messages and authors.



User Settings > SSH Keys

## SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

### Add an SSH key

Before you can add an SSH key you need to [generate it](#).

**Key**

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQCTXrRUK297rVCijjgFsOrmjDCYLLwdgAu14NhP9gqD
5UYN900Xb+KmsCbeAddIdfb6xrs/Qchr/C2plWal9acFUKu2BPP
/uuuUSUJvItrEHgpvl+6+JUA2yUsABJEXHQYTjk
/ty8QwFdkoRnmrd07lrd7MYq77BEICl8fEEcuLafWDMzFRZhsDTcaglg6m68yyVDbItr8DNEsjwrw
YCFt5A1cQUk+1xVEmBV6mqH3DwLqhPkKwFv0aDnJVoR8mXi+IzHQw+HByQj2OAPOE11MGA
yULvcyRSOZvcx/fjilfWWCqQE9p7mTW0/zrcQOhQ8I2NjbCEaWJrzSFOPtx9_a_fuls01@nubase
```

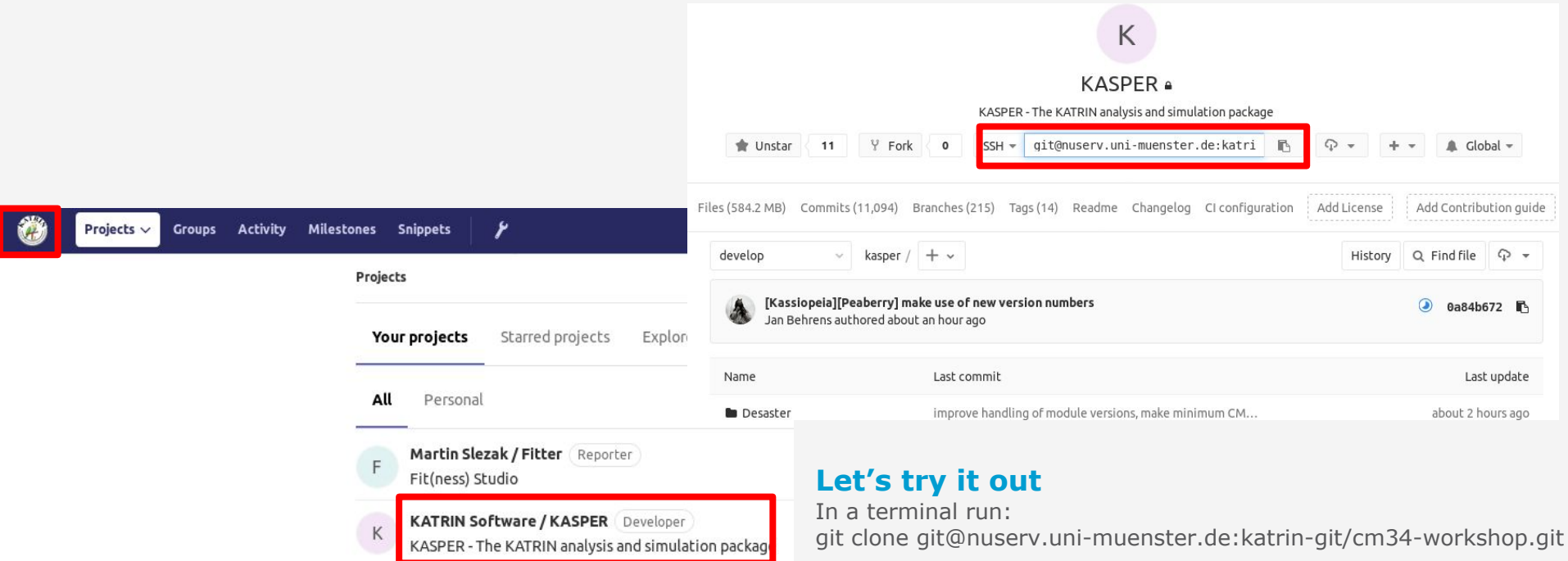
**Title**

**Add key**

**Your SSH keys (7)**

- a\_fuls01@maf2

# Getting started: Cloning projects

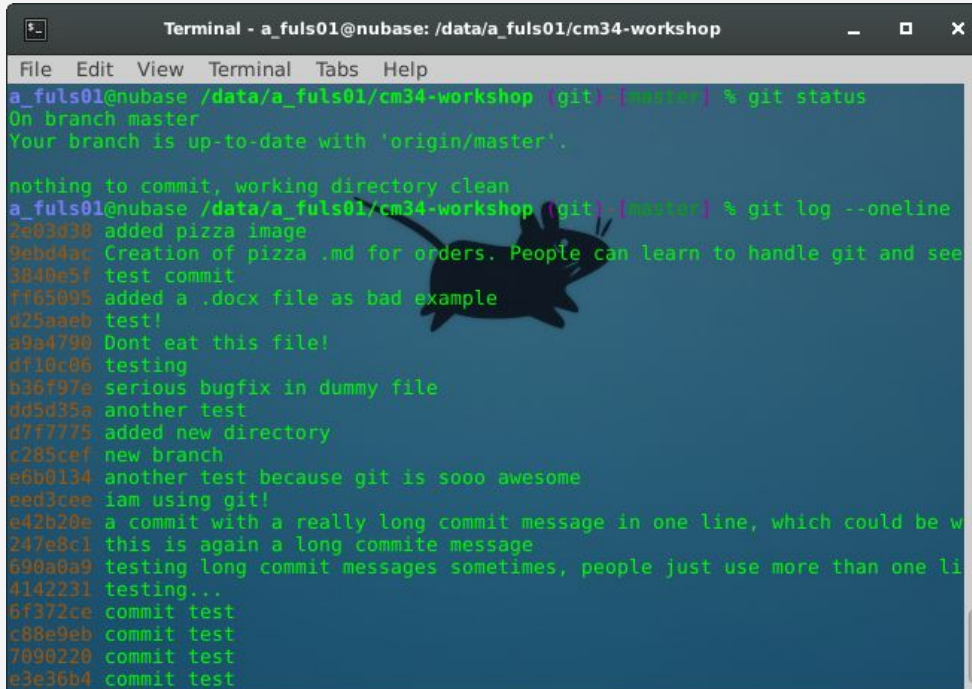


The screenshot shows the GitHub interface for the 'KASPER' repository. The repository name is 'KASPER' and the description is 'KASPER - The KATRIN analysis and simulation package'. The repository has 11 stars and 0 forks. The SSH cloning URL is highlighted with a red box: `git@nuserv.uni-muenster.de:katrin`. The repository is currently on the 'develop' branch. A commit by [Kassiopela][Peaberry] is shown, titled '[Kassiopela][Peaberry] make use of new version numbers'.

The 'Projects' section on the left shows a list of projects. The 'KATRIN Software / KASPER' project is highlighted with a red box. The project name is 'KATRIN Software / KASPER' and the role is 'Developer'. The project description is 'KASPER - The KATRIN analysis and simulation packag'.

**Let's try it out**  
In a terminal run:  
`git clone git@nuserv.uni-muenster.de:katrin-git/cm34-workshop.git`

# Getting information about the git repository:



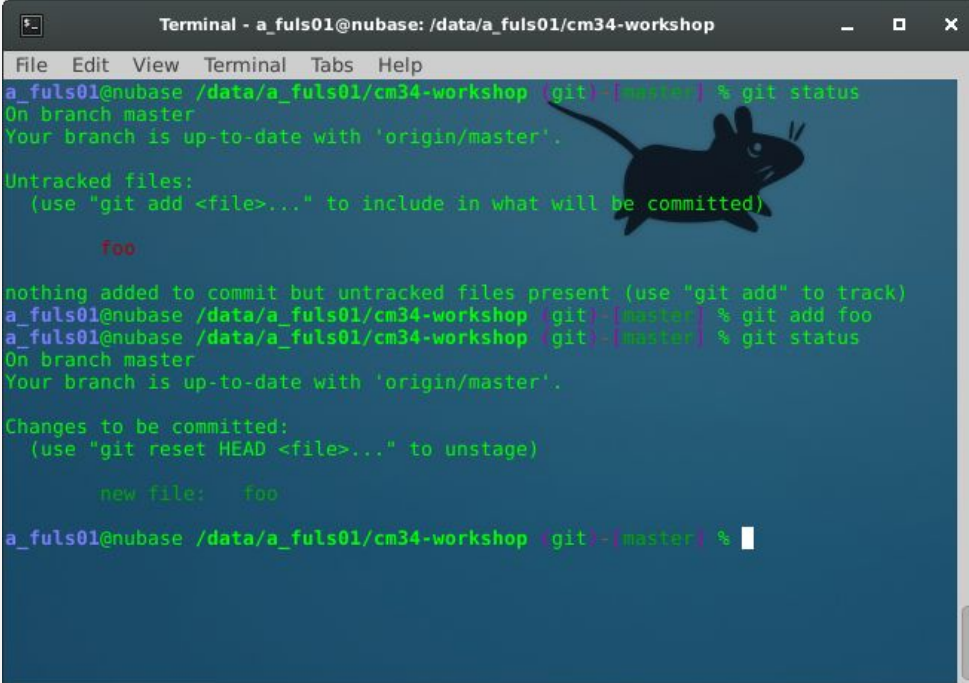
```
Terminal - a_fuls01@nubase: /data/a_fuls01/cm34-workshop
File Edit View Terminal Tabs Help
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % git log --oneline
2e05030 added pizza image
9ebd4ac Creation of pizza .md for orders. People can learn to handle git and see
3040e5f test commit
ff65095 added a .docx file as bad example
d25aaeb test!
a9a4790 Dont eat this file!
df10c06 testing
b36f97e serious bugfix in dummy file
dd5d35a another test
d7f7775 added new directory
c285cef new branch
e6b0134 another test because git is sooo awesome
eed3cee iam using git!
e42b20e a commit with a really long commit message in one line, which could be w
247e8c1 this is again a long commite message
690a0a9 testing long commit messages sometimes, people just use more than one li
4142231 testing...
6f372ce commit test
c88e9eb commit test
7090220 commit test
e3e36b4 commit test
```

- **git status** gives information about the branch you are on and changes to files to made
- **git log** shows a history of the repository with commits, authors, timestamp and commit messages
  - ◆ Shorten the output with **--oneline**
- Many more options, as with any git command type **git log --help**

# Adding, changing and saving files

- Let's start by creating a file **foo**
  - ◆ It contains the single word **bar**
- You need to tell git to track the new file with **git add foo**
  - ◆ You can add whole directories at once or use **git add --all**
  - ◆ **git add** is also needed when you change a file
- **foo** is now staged



```
Terminal - a_fuls01@nubase: /data/a_fuls01/cm34-workshop
File Edit View Terminal Tabs Help
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % git status
On branch master
Your branch is up-to-date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        foo

nothing added to commit but untracked files present (use "git add" to track)
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % git add foo
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % 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)

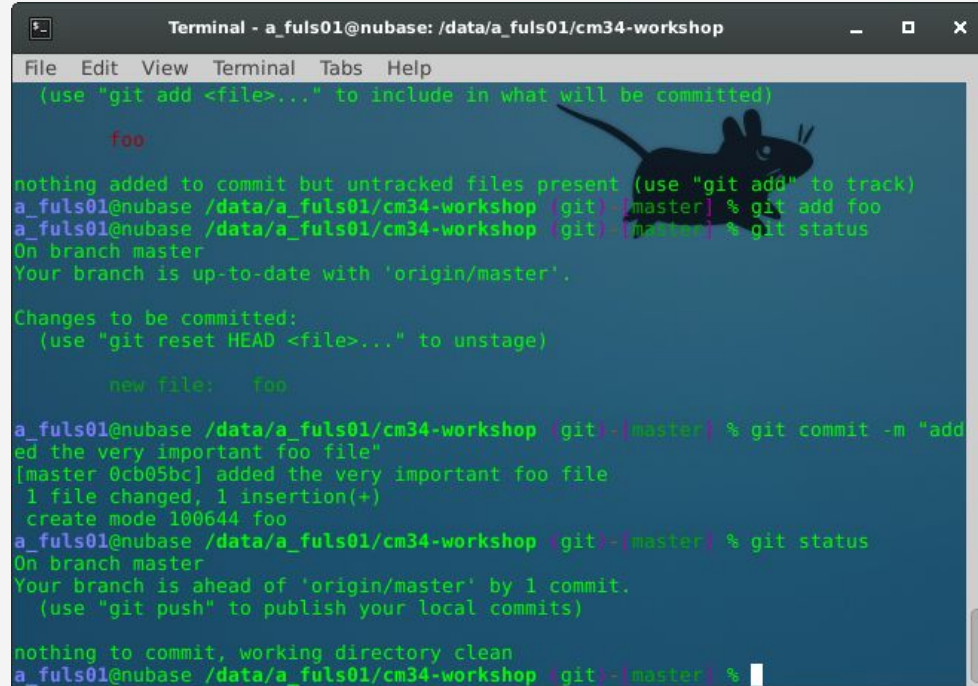
        new file:   foo

a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] %
```



# Adding, changing and saving files

- Let's start by creating a file **foo**
  - ◆ It contains the single word **bar**
- You need to tell git to track the new file with **git add foo**
  - ◆ You can add whole directories at once or use **git add --all**
  - ◆ **git add** is also needed when you change a file
- **foo** is now staged
- When you are done adding files you can commit them with **git commit -m "commit message"**
  - ◆ Until now changes are only local!
  - ◆ Use **git push** to update the remote repository



```
Terminal - a_fuls01@nubase: /data/a_fuls01/cm34-workshop
File Edit View Terminal Tabs Help
(use "git add <file>.." to include in what will be committed)

foo

nothing added to commit but untracked files present (use "git add" to track)
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % git add foo
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % 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)

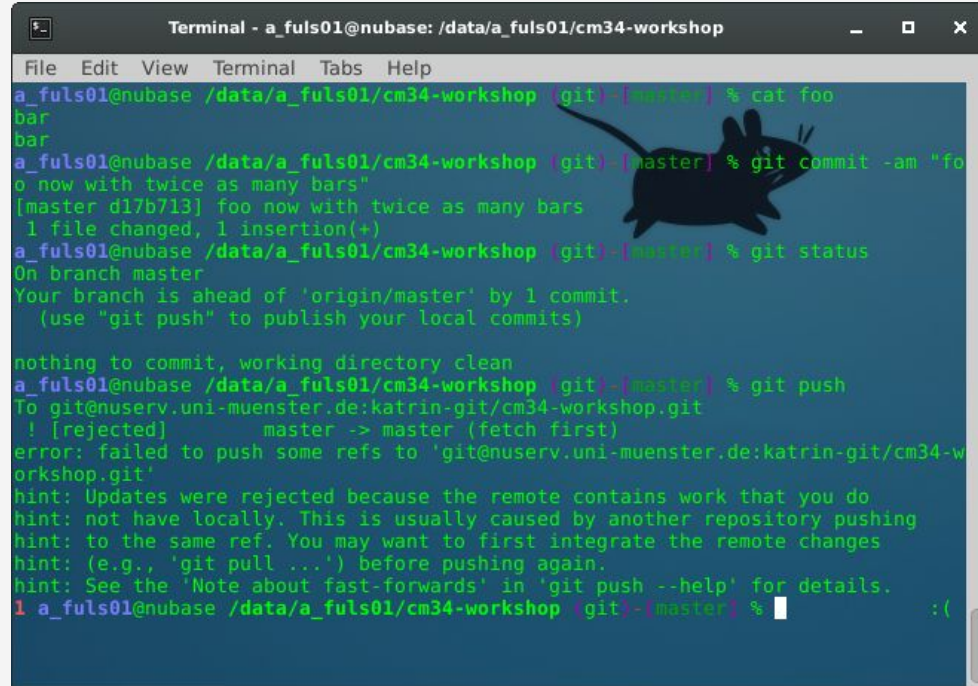
    new file:   foo

a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % git commit -m "add
ed the very important foo file"
[master 0cb05bc] added the very important foo file
 1 file changed, 1 insertion(+)
 create mode 100644 foo
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working directory clean
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] %
```

# Synchronizing with the remote repository and fixing conflicts

- **git push** publishes your local changes to the repository
- **git pull** applies changes in the repository to your local files
- If the remote repository has changes you do not yet have, you cannot push!
  - ◆ First use **git pull**

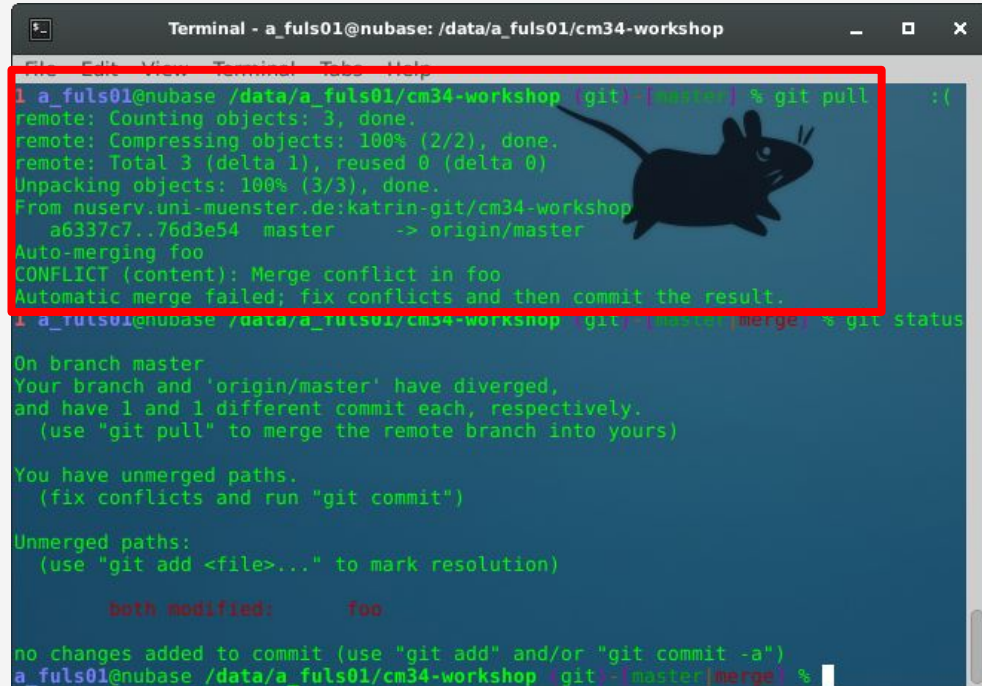


```
Terminal - a_fuls01@nubase: /data/a_fuls01/cm34-workshop
File Edit View Terminal Tabs Help
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % cat foo
bar
bar
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % git commit -am "fo
o now with twice as many bars"
[master d17b713] foo now with twice as many bars
 1 file changed, 1 insertion(+)
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working directory clean
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % git push
To git@nuserv.uni-muenster.de:katrin-git/cm34-workshop.git
 ! [rejected]          master -> master (fetch first)
error: failed to push some refs to 'git@nuserv.uni-muenster.de:katrin-git/cm34-w
orkshop.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
1 a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] %
```

# Synchronizing with the remote repository and fixing conflicts

- **git push** publishes your local changes to the repository
- **git pull** applies changes in the repository to your local files
- If the remote repository has changes you do not yet have, you cannot push!
  - ◆ First use **git pull**
- But what if someone made changes to the same file you want to push



```
Terminal - a_fuls01@nubase: /data/a_fuls01/cm34-workshop
File Edit View Terminal Tabs Help
1 a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % git pull :(
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From nuserv.uni-muenster.de:katrin-git/cm34-workshop
   a6337c7..76d3e54  master    -> origin/master
Auto-merging foo
CONFLICT (content): Merge conflict in foo
Automatic merge failed; fix conflicts and then commit the result.
1 a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % git status

On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commit each, respectively.
(use "git pull" to merge the remote branch into yours)

You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

   both modified:   foo

no changes added to commit (use "git add" and/or "git commit -a")
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] %
```

# Synchronizing with the remote repository and fixing conflicts

- **git push** publishes your local changes to the repository
- **git pull** applies changes in the repository to your local files
- If the remote repository has changes you do not yet have, you cannot push!
  - ◆ First use **git pull**
- But what if someone made changes to the same file you want to push



# Synchronizing with the remote repository and fixing conflicts

- **git push** publishes your local changes to the repository
- **git pull** applies changes in the repository to your local files
- If the remote repository has changes you do not yet have, you cannot push!
  - ◆ First use **git pull**
- But what if someone made changes to the same file you want to push
- Fix the conflict (text editor or **git mergetool**), commit and push the fix

```
Terminal - a_fuls01@nubase: /data/a_fuls01/cm34-workshop
File Edit View Terminal Tabs Help
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master|merge] % cat foo
bar
<<<<<<<< HEAD
bar
=====
foo
>>>>>>> 76d3e54e57d122e9e69c79079d7731723d025ef7
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master|merge] % vim foo
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master|merge] % git commit -
am "bar is better than foo"
[master c743b5e] bar is better than foo
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
(use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)

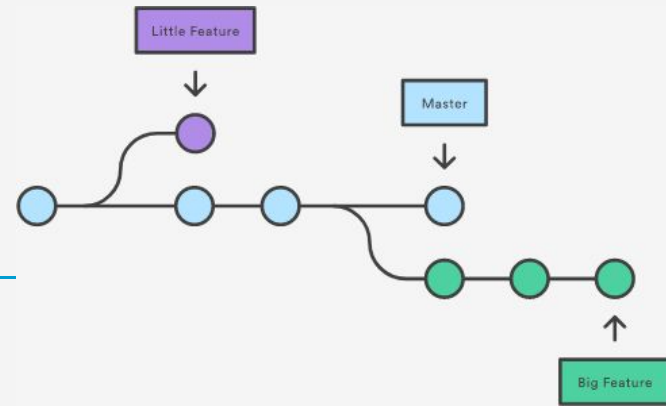
    foo.orig
    foo-

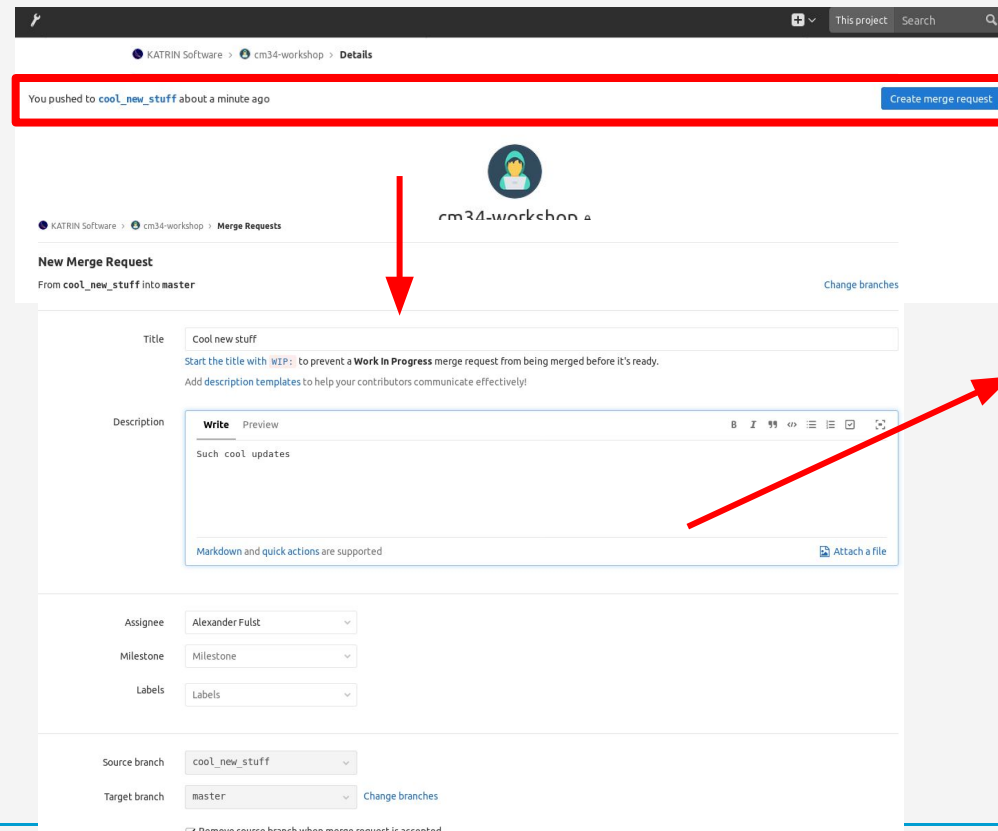
nothing added to commit but untracked files present (use "git add" to track)
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] % git push
Counting objects: 8, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 412 bytes | 0 bytes/s, done.
Total 4 (delta 2), reused 0 (delta 0)
To git@nuserv.uni-muenster.de:katrin-git/cm34-workshop.git
 76d3e54..c743b5e  master -> master
a_fuls01@nubase /data/a_fuls01/cm34-workshop (git)-[master] %
```

# Branching

- Branching allows you to work on several features or bug fixes at the same time, without interfering with the master / develop branch
  - ◆ For the kasper project you have to use branches if you want to contribute, because you cannot directly push to the *develop* branch
- Create a new branch with ***git branch new\_branch\_name*** or directly switch to it with ***git checkout -b new\_branch\_name***
- Check on which branch you are with ***git status*** or use ***git branch -vv***
- Switch between branches with ***git checkout branch\_name***

- For this example I added two new files in the new branch:
  - ◆ *cool\_stuff*
  - ◆ *more\_stuff*
- I edited *foo* to have yet another bar
- Someone on the *master* branch has overwritten the 'bar's with 'foo's
- I pushed my local branch to the remote with ***git push origin cool\_new\_stuff***
- Further steps are done in the web interface





You pushed to `cool_new_stuff` about a minute ago [Create merge request](#)

**New Merge Request**  
From `cool_new_stuff` into `master` [Change branches](#)

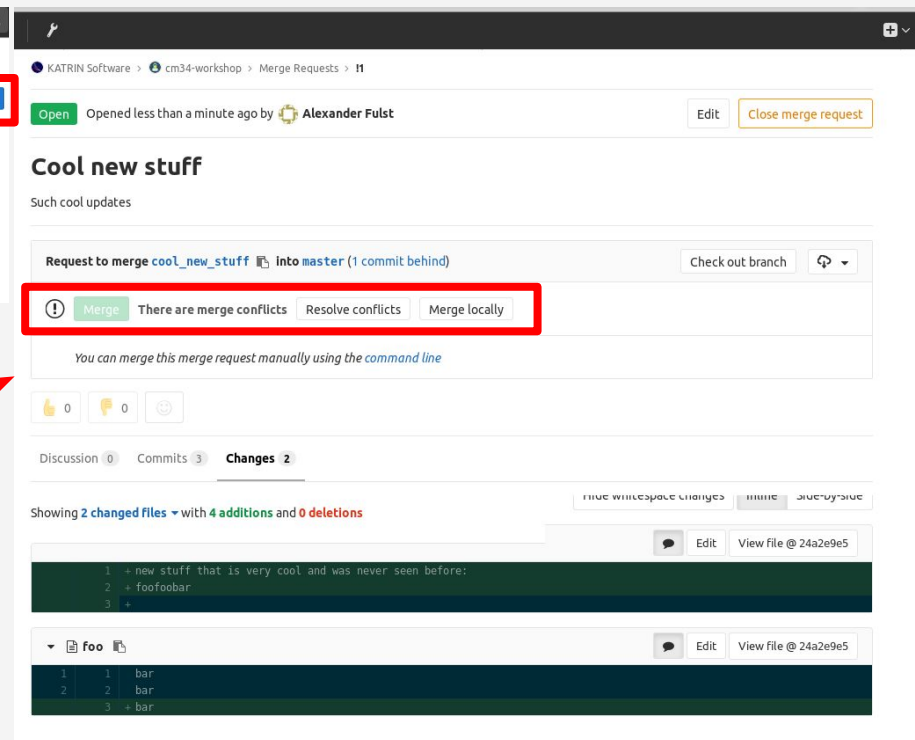
Title: `Cool new stuff`  
Start the title with `WIP:` to prevent a **Work In Progress** merge request from being merged before it's ready. Add [description templates](#) to help your contributors communicate effectively!

Description: `Such cool updates`

Assignee: Alexander Fulst  
Milestone: Milestone  
Labels: Labels

Source branch: `cool_new_stuff`  
Target branch: `master` [Change branches](#)

Remove source branch when merge request is accepted.



[Open](#) Opened less than a minute ago by [Alexander Fulst](#) [Edit](#) [Close merge request](#)

## Cool new stuff

Such cool updates

Request to merge `cool_new_stuff` into `master` (1 commit behind) [Check out branch](#) [↻](#)

[Merge](#) **There are merge conflicts** [Resolve conflicts](#) [Merge locally](#)

*You can merge this merge request manually using the command line*

👍 0 🙌 0 😊 0

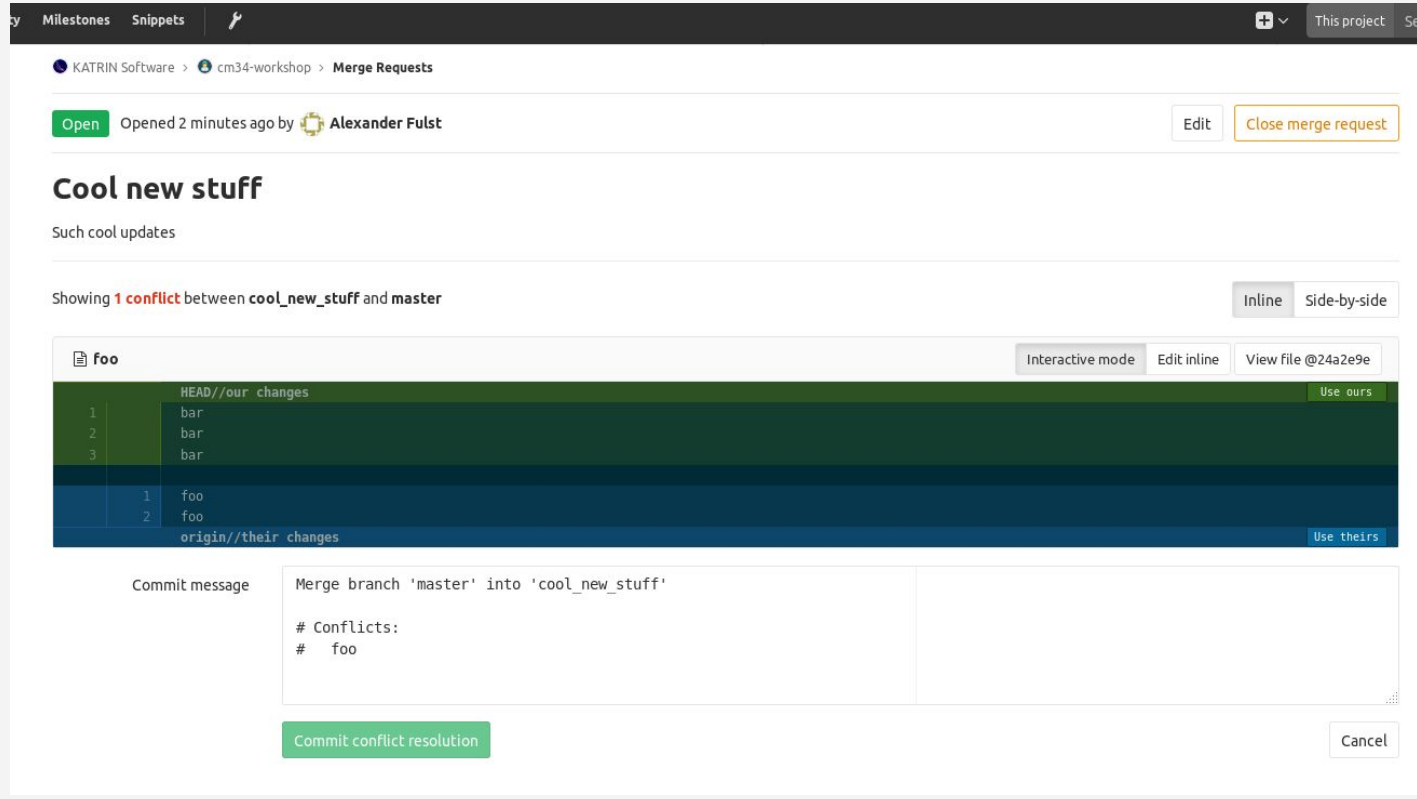
Discussion 0 Commits 3 **Changes 2**

Showing 2 changed files with 4 additions and 0 deletions

```
1 + new stuff that is very cool, and was never seen before:
2 + foofoobar
3 +
```

foo

```
1 1 bar
2 2 bar
3 + bar
```



The screenshot shows a Git merge request interface. At the top, there are navigation tabs for 'Milestones', 'Snippets', and a search icon. The breadcrumb path is 'KATRIN Software > cm34-workshop > Merge Requests'. Below this, there is a green 'Open' button, the text 'Opened 2 minutes ago by Alexander Fulst', and buttons for 'Edit' and 'Close merge request'.

### Cool new stuff

Such cool updates

Showing 1 conflict between cool\_new\_stuff and master

Inline Side-by-side

foo

Interactive mode Edit inline View file @24a2e9e

Line	HEAD/our changes	origin/their changes
1	bar	
2	bar	
3	bar	
		foo
1		foo
2		foo

Commit message

```
Merge branch 'master' into 'cool_new_stuff'

# Conflicts:
#   foo
```

Commit conflict resolution Cancel



# Markdown

- Markdown is a lightweight markup language
- It uses a simplistic syntax for text formatting and is supported by GitLab both in *.md* files and in comment fields.
- Useful features:
  - ◆ Code sections with syntax highlighting
  - ◆ Links to files, users, commits, merge requests, ...
  - ◆ Structure and highlight part of your comments
- Help is directly available underneath comment fields in GitLab

```
make install
```

All files will be installed below the `install/` path configured in CMake, e.g. executables are installed in the `install/bin/` directory, and libraries are installed in the `install/lib/` directory.

6. Run the `kasperenv.sh` script to make the Kasper executables available:

```
source /path/to/Kasper/install/bin/kasperenv.sh
```

This script sets up the environment variables for Kasper, e.g. it adds the `bin/` directory to your `$PATH` so you can call executables directly from the commandline.

Furthermore this sets the `$KASPERSYS` environment variable. This step is very important if you use KATRIN-specific configurations or geometry. These configurations use the `$KASPERSYS` variable to include other files.

1. To execute the `kasperenv.sh` script automatically on every login, you can include it in your `~/.bashrc` by adding a line at the end which contains the command above.

If you're using a different shell than Bash, you must edit the appropriate file instead (e.g. `~/.zshrc` if you're using Zsh.)

## System requirements:

### Linux/MacOS (Windows+cygwin should work too, but has not been tested)

Some dependencies are only required if certain module are compiled in.

Dependencies:

- CMake ([www.cmake.org](http://www.cmake.org)) version 2.8 or higher

# Stashing

- Imagine you work on a new feature, but your workflow gets interrupted because you have to implement a critical hotfix.
- ◆ You **could** commit your changes, *checkout* a new hotfix branch and later switch back to your feature branch.
  - ◆ However this leaves you with a commit which has broken code!
  - ◆ You can better use ***git stash*** (***push***) to shelve your changes and ***git stash apply*** to reapply them later

```
Terminal - alex@ideapad: ~/katrin/cm34-workshop
File Edit View Terminal Tabs Help
alex@ideapad ~/katrin/cm34-workshop (git)-[master] % 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:   foo

no changes added to commit (use "git add" and/or "git commit -a")
alex@ideapad ~/katrin/cm34-workshop (git)-[master] % git stash push -m "wip, fixing stuff"
Saved working directory and index state On master: wip, fixing stuff
alex@ideapad ~/katrin/cm34-workshop (git)-[master] % git stash list
stash@{0}: On master: wip, fixing stuff
alex@ideapad ~/katrin/cm34-workshop (git)-[master] % git stash show 0
foo | 1 +
 1 file changed, 1 insertion(+)
alex@ideapad ~/katrin/cm34-workshop (git)-[master] % git stash apply 0
On branch master
Your branch is up to date with 'origin/master'.

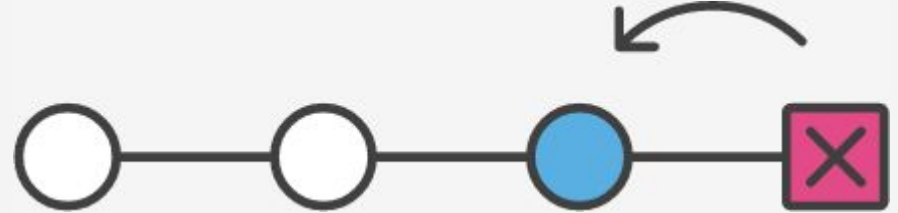
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   foo

no changes added to commit (use "git add" and/or "git commit -a")
alex@ideapad ~/katrin/cm34-workshop (git)-[master] %
```

## What if something goes wrong: Stash, checkout and revert

- If you mess up something but did not yet commit the changes, you can use **git stash**, **git stash drop** to undo the changes.
- You can *checkout* entire commits or specific files from a specific commit:
  - ◆ **git checkout <commit>**  
(**<path to file>**)
  - ◆ This files can then be added / committed like any other change
- **git revert HEAD~1** produces a new commit as inverse of the second last commit, rolling back any changes the commit introduced. This should be preferred about git reset when you work on public branches, because it conserves the history!






- There are further tools like **git reset** and **git rebase** which are really powerful and can rewrite the history of a repository. Therefore they are potentially harmful if you don't know what you are doing. As a rule of thumb they should **never be used on pushed commits of a public branch!**
  - ◆ If you have not pushed your last 2 commits you can use **git reset HEAD~2** to get rid of them.
  - ◆ Those commits will be deleted by git's garbage collection

# Thank you for your attention

In case of fire



-  1. git commit
-  2. git push
-  3. leave building

