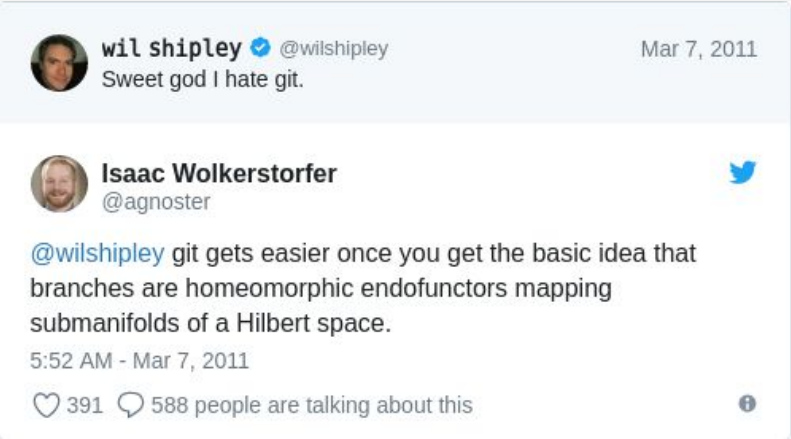
WWU MÜNSTER

Git(Lab) Tutorial and Hands-On



Alexander Fulst

living.knowledge

Analysis Workshop | 34th KATRIN Collaboration Meeting
Institut für Kernphysik | WWU Münster
21.02.2018

WWU MÜNSTER

Git(Lab) Tutorial and Hands-On

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

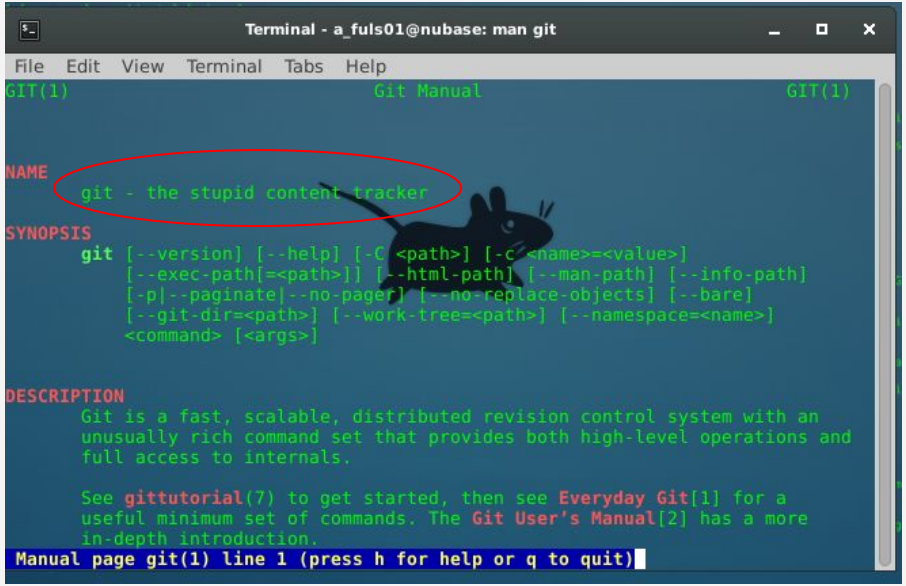
Alexander Fulst

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.

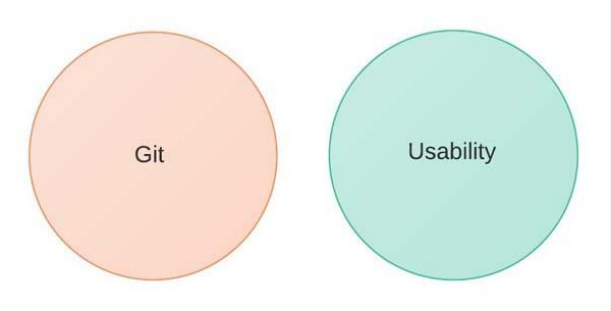
Output of *man git*

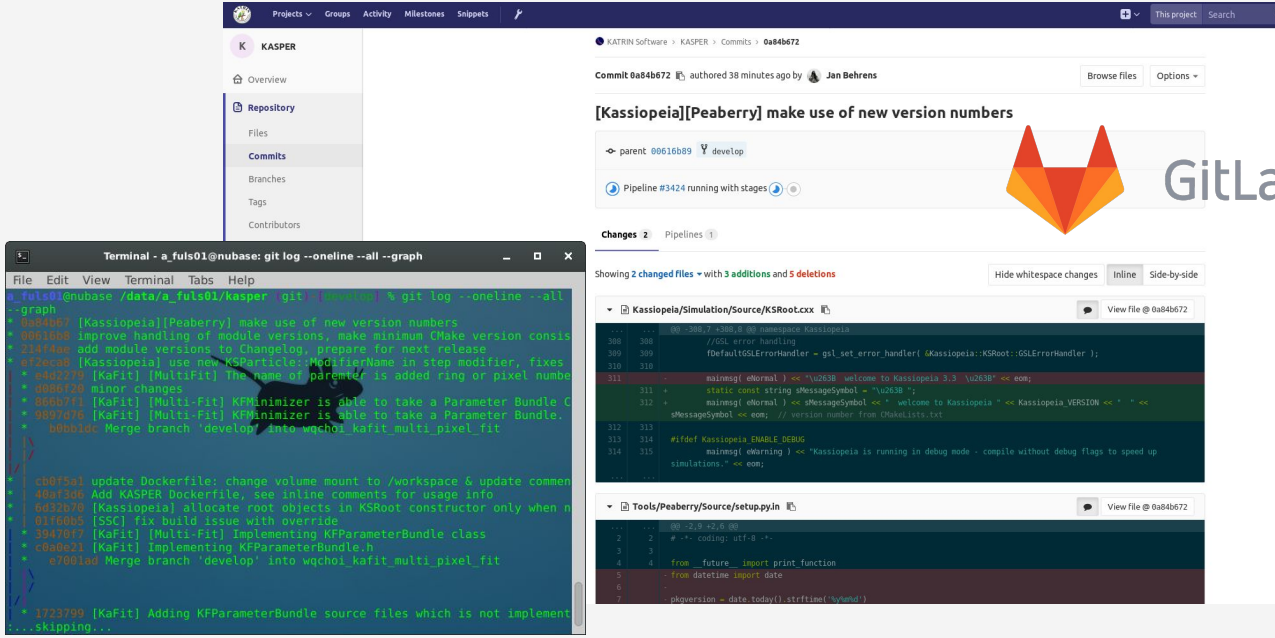


Two Ways to use Git

A mix of command line and web interface

Exclusively from command line





The first thing to do: adding your ssh key

Projects Groups Activity Milestones Snippets Search

User Settings
- Profile
- Account
- Chat
- Access Tokens
- Emails
- Password
- Notifications
- SSH Keys
- GPG Keys
- Preferences
- Authentication log

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
AAAAB3NzaC1yc2EAAAADAQABAAABAQCjTXrRUk297rVCijigFsOrrmjDGYLLwdgAu14NhP9gqD
5UYN90OXb+KmsCbeAddIdfb6XrS/Qchr/C2plWal9acfUKu2BPp
/uuuUSUJvIrEHgpvl+6+JUA2yUsABJEXHQYTjk
/ty8QwfDkoRNmrd07lrtd7MYq77BElCl8fEEcuLafWDMzfRZhSDTcaglg6m68yyVDbltr8rDNEsjwvr
YCFT5A1cQUk+1xVEmBV6mqH3DwLqhPkKwFv0aDnJVoR8mXi+IZHGw+HByQj2OAPOE11MGA
yULvCyRSOZvcx/f/ilfWWCqQE9p7mTW0/zrcQOhQ8I2NjbCEaWJrzSfOPtx9 a_fuls01@nubase

Title

a_fuls01@nubase

Add key

Your SSH keys (7)

a_fuls01@maf2

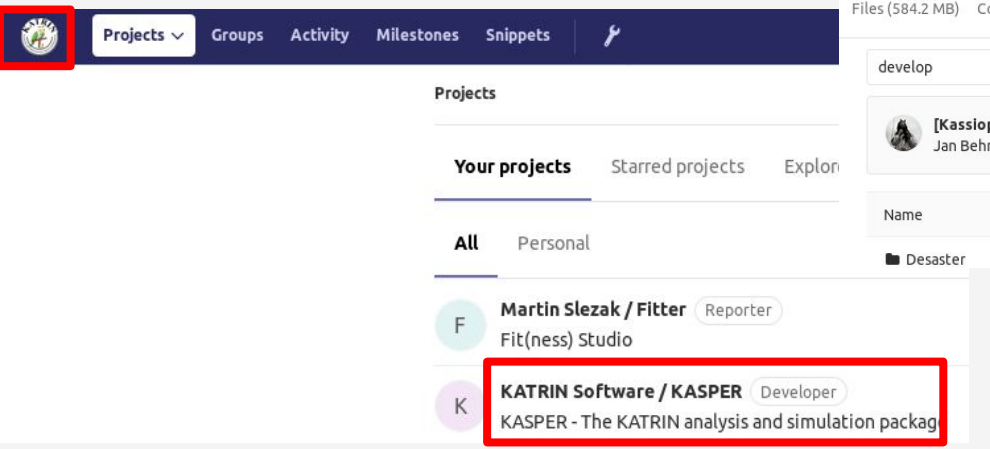
Alexander Fulst
@afulst

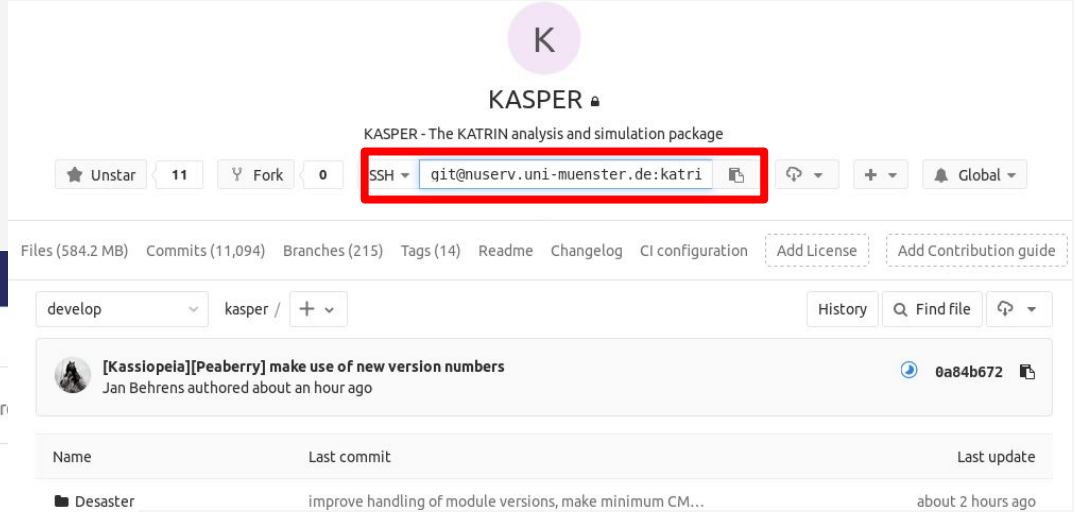
Profile
Settings
Help
Sign out

ssh-keygen
cat ~/.ssh/id_rsa.pub

Additional setup in your terminal
git config --global user.name "Alexander Fulst"
git config --global user.email "alexanderfulst@wwu.de"

Getting started: Cloning projects

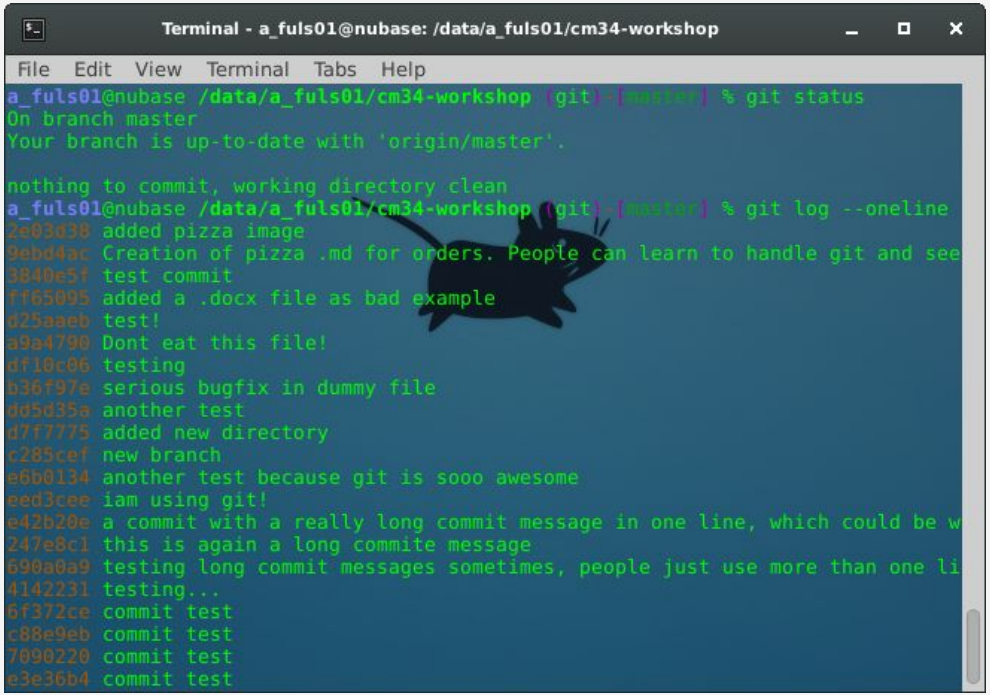




Let’s try it out

In a terminal run:
git clone git@nuserv.uni-muenster.de:katrin-git/cm34-workshop.git

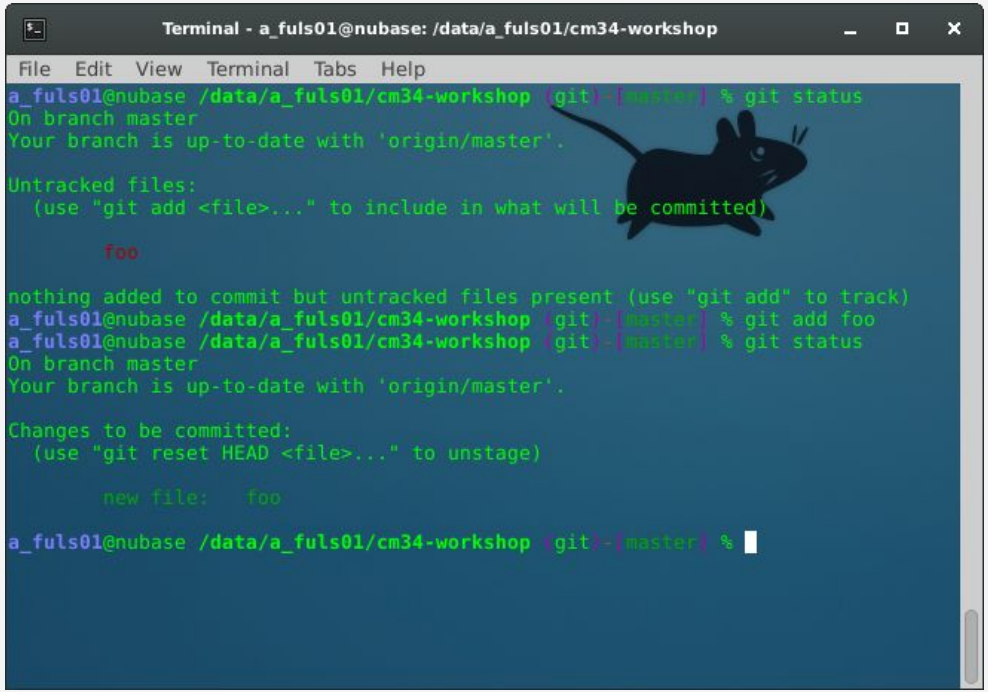
Getting information about the git repository:



- ➔ ***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***

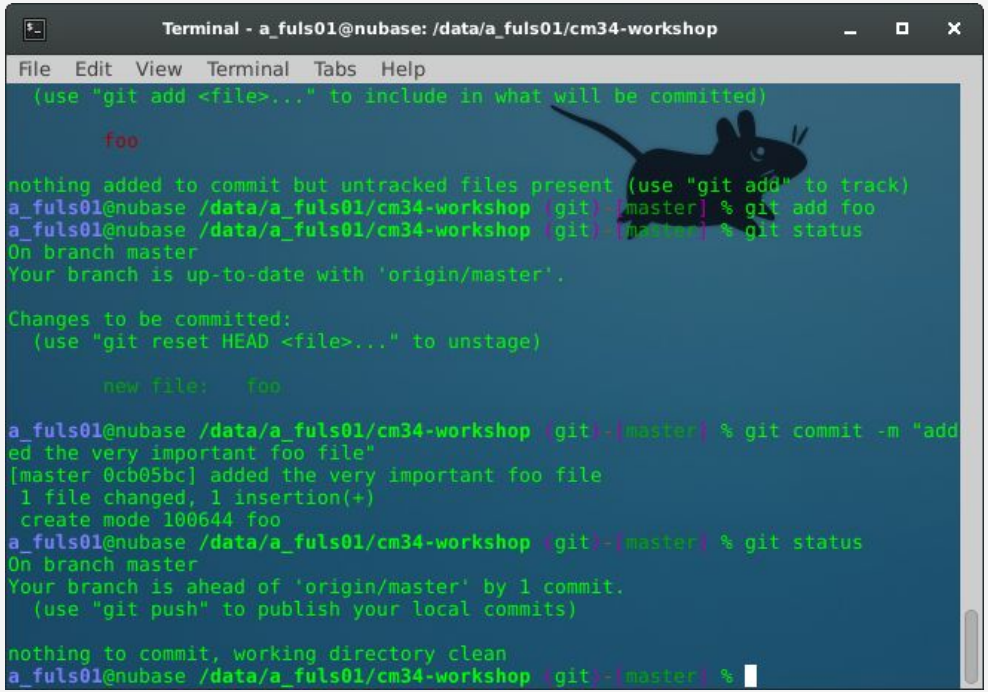
Adding, changing and saving files

- ➔ 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



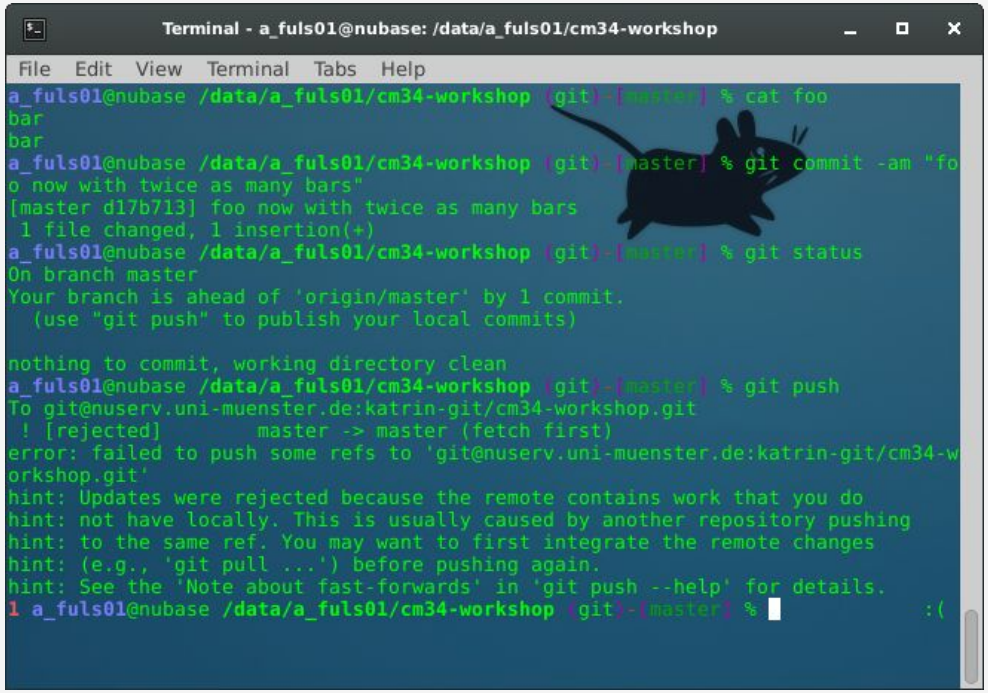
Adding, changing and saving files

- 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



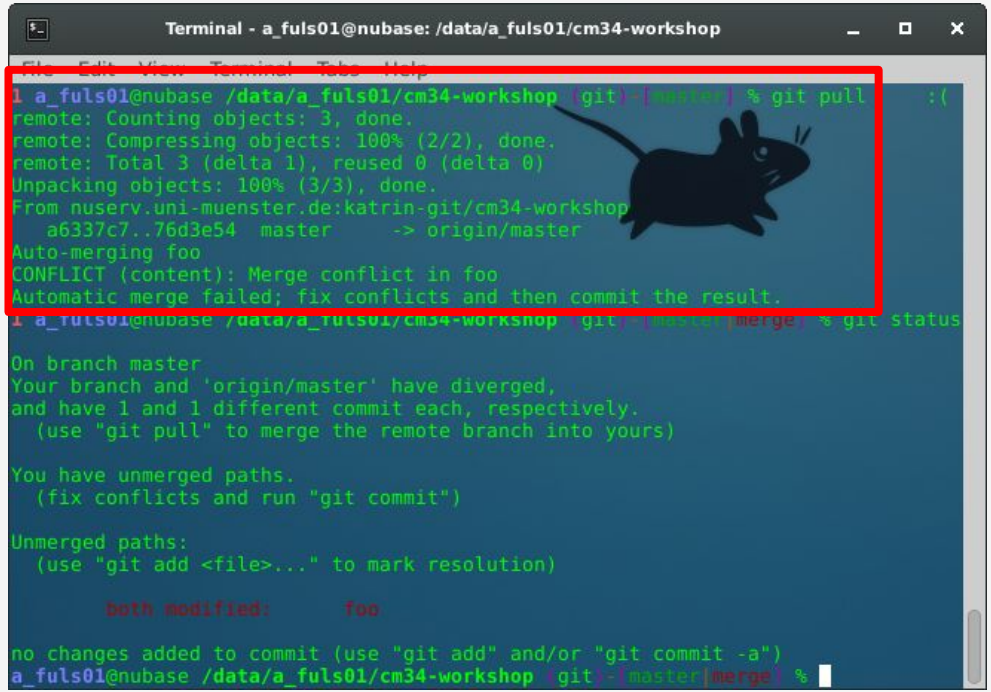
Synchronizing with the remote repository and fixing conflicts

- ➔ ***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***



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

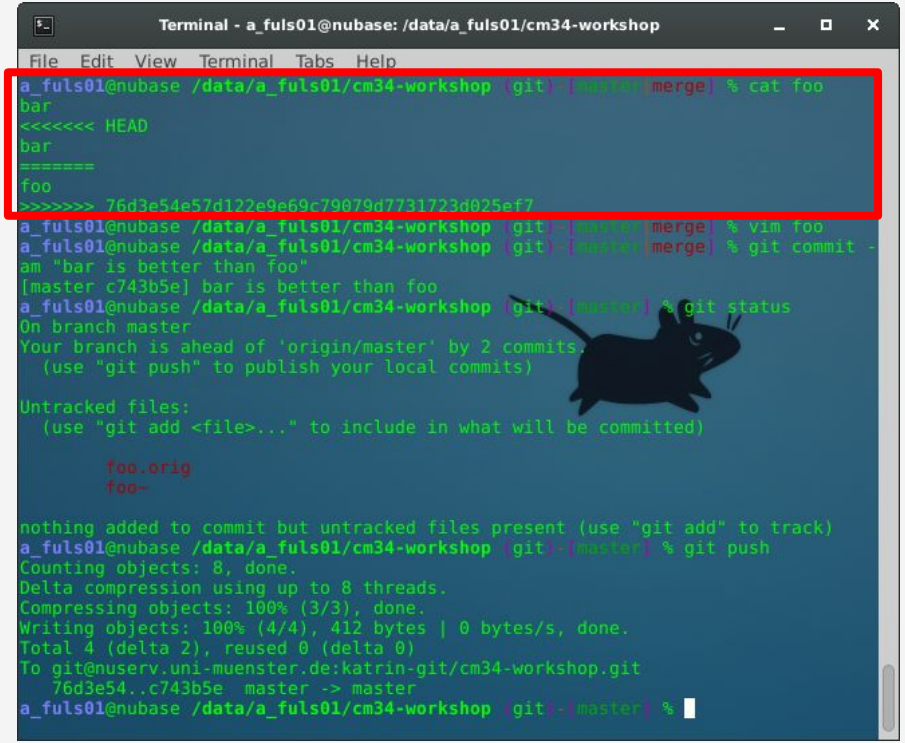


Synchronizing with the remote repository and fixing conflicts

- ➔ ***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

Synchronizing with the remote repository and fixing conflicts

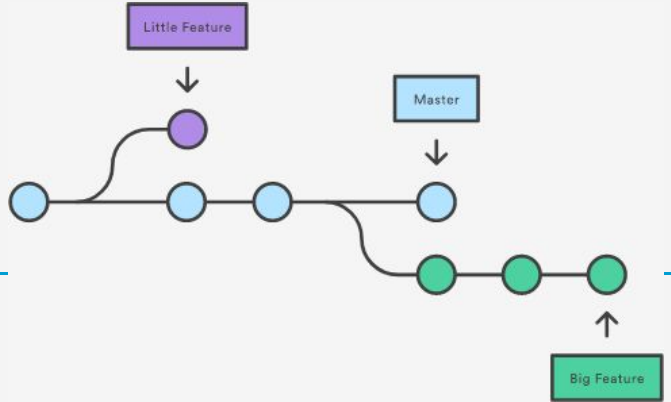
- ➔ ***git push*** publishes your local changes to the repository
- ➔ ***git pull*** applies changes in the repository to your local files
- ➔ If the remote repository has changes you do not yet have, you cannot push!
  - ◆ First use ***git pull***
- ➔ But what if someone made changes to the same file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



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



This project Search

KATRIN Software › cm34-workshop › Details

You pushed to cool_new_stuff about a minute ago

Create merge request

KATRIN Software › cm34-workshop › Merge Requests

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: Write Preview

Such cool updates

Markdown and quick actions are supported

Attach a file

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.

KATRIN Software › cm34-workshop › Merge Requests › !1

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

Discussion 0 Commits 3 Changes 2

Showing 2 changed files with 4 additions and 0 deletions

Edit View file @ 24a2e9e5

```
+ new stuff that is very cool and was never seen before:
+ foofoobar
+
```

foo

Edit View file @ 24a2e9e5

```
  bar
  bar
+ bar
```

Milestones Snippets This project Se

KATRIN Software › cm34-workshop › Merge Requests

Open Opened 2 minutes ago by Alexander Fulst Edit 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

```
HEAD//our changes                                   Use ours
1  bar
2  bar
3  bar

1  foo
2  foo
origin//their changes                               Use theirs
```

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) 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 shelf your changes and ***git stash apply*** to reapply them later

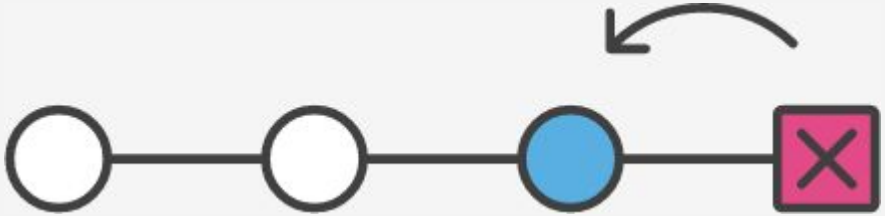
Terminal - alex@ideapad: ~/katrin/cm34-workshop

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, fix
ing 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 prefered 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

WWU MÜNSTER

Thank you for your attention