

# Git Introduction

Holger Obermaier | 24. May 2022



# Table of Contents

1. Motivation
2. Git: Data structures
3. Simple Workflow
4. Working with Tags
5. Working with Branches
6. Working with Remote Servers
7. References

# Motivation

## Why Use Version Control?

- Collaboration
  - Code sharing / distribution
  - Resolving code conflicts
- Version History
  - Track changes over time
  - Meta data
    - Who made the change?: Author
    - When was it made?: Date
    - Why was it made?: Message
  - Which commit broke something , e.g. a unit test?
- Backup: Restoring previous versions
- Data Integrity

# Motivation (continued)

## Git Pros

- Distributed system (no central repository) / enables easy collaborative work
- Work offline (Entire repository is stored locally)
- Fast (All operations can be performed locally)
- Powerful (extensive functionality)
- Strong ecosystem (GitLab, GitHub), tools (IDE, GUIs, CLI)

## Git Cons

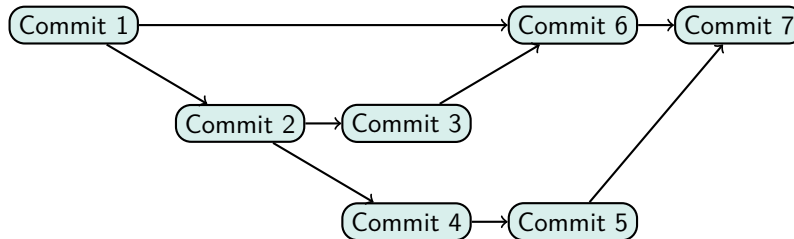
- No central repository (Question: What is the latest state of development)
- Problems with large repositories, as stored locally  
(Workarounds: LFS, Shallow Copy, Sparse Checkout)
- No tracking of copies and renaming of files: Detection with similarity heuristics
- Complex: Steep learning curve

# Git: Data structures

- Git manages files and directories
- **Repository** a git managed directory with all its files and directories
- **Commit** A specific temporal state of a repository (snapshot)
- **History** A successive sequence of commits
- **Directed Acyclic Graph**
  - Graph  $\rightarrow$  Vertices and Edges
  - Directed edges
  - Will never form a closed loop

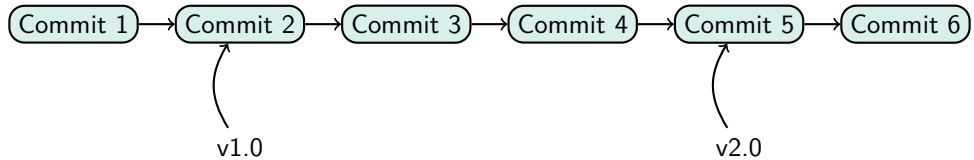
# Git: Data structures (continued)

- Directed Acyclic Graph



# Git: Data structures (continued)

- **Tags**: Human readable references to commits

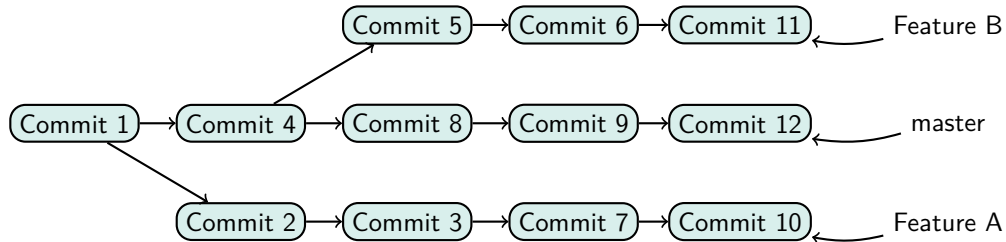


# Git: Data structures (continued)

- Branches

- master

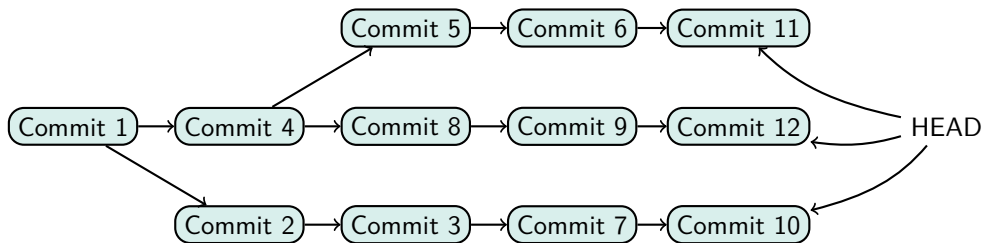
- Name of default branch
    - Configurable
    - Common alternative names: main , trunk , development





# Git: Data structures (continued)

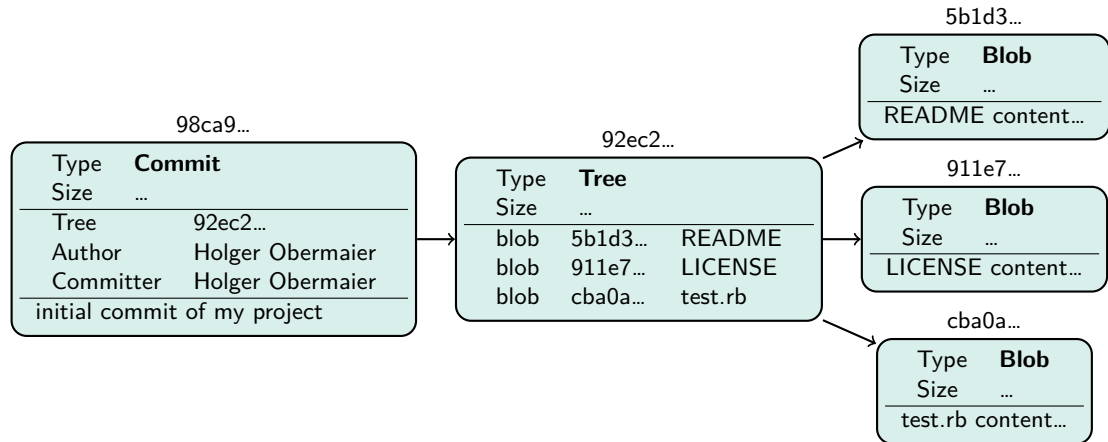
- **HEAD** Reference to the last commit on the current branch
- **detached HEAD** Reference to the currently checked out commit



# Git: Data structures (continued)

- All data and metadata is stored in objects
- Three types of objects: Commit , Tree , Blob
- All objects are identified by hash codes (not simple numbers)
- Hash function maps from object contents to (hex) number e.g.:
  - Text 1  $\mapsto$  e7d3fcbbd2c090f63c3b88585b9400932e0926b1
  - Text 2  $\mapsto$  abdf644dbb8d2e6cac179cb17f3e5c2e0257e2d5
- Objects are considered equal if their hash code matches
- Danger of collisions  $\Rightarrow$  Migration from sha1 (160 bit) to stronger hash function

# Git: Data structures (continued)



# Simple Workflow

- Git requires a first-time setup
- Non permanent configuration through environment variables

```
export GIT_COMMITTER_NAME="Holger Obermaier"  
export GIT_COMMITTER_EMAIL="Holger.Obermaier@kit.edu"  
export GIT_AUTHOR_NAME="Holger Obermaier"  
export GIT_AUTHOR_EMAIL="Holger.Obermaier@kit.edu"
```

- Permanent configuration through configuration files (`${HOME}/.gitconfig`)

```
git config --global user.name "Holger Obermaier"  
git config --global user.email holger.obermaier@kit.edu
```

# Simple Workflow

- Initialize an empty Git repository

```
mkdir myFirstRepository  
cd    myFirstRepository  
git  init
```

```
...  
Initialized empty Git repository in .../myFirstRepository/.git/
```

- Creates hidden directory `.git`
- Git objects and references are stored in `.git`

## Simple Workflow (continued)

- Review status of the repository

```
git status
```

```
On branch master
```

```
No commits yet
```

```
nothing to commit (create/copy files and use "git add" to track)
```

## Simple Workflow (continued)

- Create file `helloWorld.c` with the editor of your choice

```
int main(int* argc, char *argv[]) {  
    printf("Hello World\n");  
}
```

## Simple Workflow (continued)

- Review status of the repository

```
git status
```

```
On branch master
```

```
No commits yet
```

```
Untracked files:
```

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

```
  helloWorld.c
```

```
nothing added to commit but untracked files present (use "git add" to track)
```



## Simple Workflow (continued)

- Commits are prepared in the staging area (also called cache or index)
- Multiple changes can be staged before a commit
- Only staged changes are part of a commit. Unstaged changes are ignored
- Add file `helloWorld.c` to the staging area

```
git add helloWorld.c
```

## Simple Workflow (continued)

- Review status of the repository

```
git status
```

```
On branch master
```

```
No commits yet
```

```
Changes to be committed:
```

```
  (use "git rm --cached <file>..." to unstage)
```

```
    new file:   helloWorld.c
```

## Simple Workflow (continued)

- Commit changes to repository

```
git commit -m "Add hello world"
```

```
[master (Root-Commit) 31231f5] Add hello world  
1 file changed, 3 insertions(+)  
create mode 100644 helloWorld.c
```

- Choose appropriate and short commit message!  
This helps to understand the changes made.

## Simple Workflow (continued)

- Review status of the repository

```
git status
```

```
On branch master  
nothing to commit, working tree clean
```

## Simple Workflow (continued)

- Review history on the current branch

```
git log
```

```
commit aebed842803f58b9b08c7e9755bb1a562f2abb8e
Author: Holger Obermaier <holger.obermaier@kit.edu>
Date:   Tue Oct 19 10:55:34 2021 +0200

    Add hello world
```

- `git log` shows flattened history ignoring graph structure of repository
- `git log --all --decorate --graph` shows history with graph structure

## Simple Workflow (continued)

- Modify file `helloWorld.c` with the editor of choice. Add missing include directive

```
#include <stdio.h>

int main(int* argc, char *argv[]) {
    printf("Hello World\n");
}
```

## Simple Workflow (continued)

- Review status of the repository

```
git status
```

```
On branch master
```

```
Changes not staged for commit:
```

```
  (use "git add <file>..." to update what will be committed)
```

```
  (use "git restore <file>..." to discard changes in working directory)
```

```
    modified:   helloWorld.c
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

## Simple Workflow (continued)

- Review changes of modified files

```
git diff
```

```
diff --git a/helloWorld.c b/helloWorld.c
index f1ad8d5..f319c4a 100644
--- a/helloWorld.c
+++ b/helloWorld.c
@@ -1,3 +1,5 @@
+#include <stdio.h>

int main(int* argc, char *argv[]) {
    printf("Hello World\n");
}
```



## Simple Workflow (continued)

- Directly commit changes to repository, without using staging area

```
git commit -m "Added missing include directive" helloWorld.c
```

```
[master db8e376] Added missing include directive  
1 file changed, 2 insertions(+)
```

## Simple Workflow (continued)

- Review history on the current branch

```
git log
```

```
commit db8e376d5c52d3605e516022eeefd476cc813369 (HEAD -> master)
```

```
Author: Holger Obermaier <holger.obermaier@kit.edu>
```

```
Date: Tue Oct 19 15:38:27 2021 +0200
```

```
    Added missing include directive
```

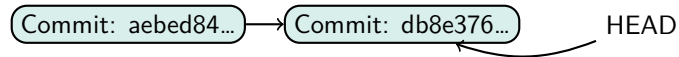
```
commit aebed842803f58b9b08c7e9755bb1a562f2abb8e
```

```
Author: Holger Obermaier <holger.obermaier@kit.edu>
```

```
Date: Tue Oct 19 10:55:34 2021 +0200
```

```
    Add hello world
```

## Simple Workflow (continued)



# Simple Workflow (continued)

## Various useful Git commands

- `git show <Commit>`  
Show changes made in the commit
- `git checkout <Commit>`  
Return to a previous commit
- `git rm <File>`  
Remove files, directories or links in the repository
- `git restore <File>`  
Restore a modified file
- `git mv <Source> <Destination>`  
Move / rename a file, directory or link in the repository
- `git grep <Pattern>`  
Search in the repository

# Simple Workflow (continued)

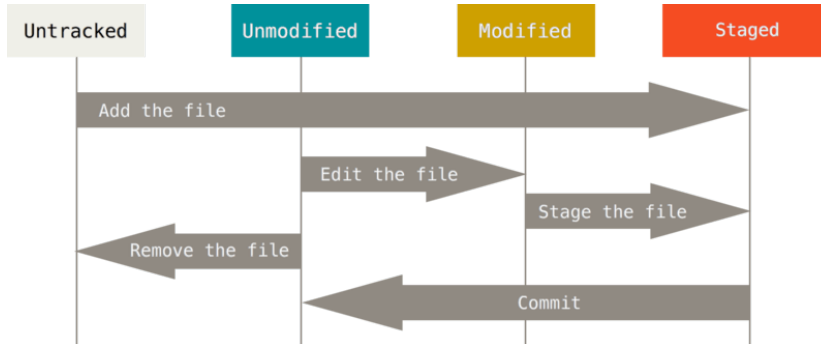


Figure: The life cycle of the status of your files (from [Pro-Git])

# Working with Tags

- Create an empty repository

```
git init WorkingWithTags  
cd WorkingWithTags
```

- Create an executable shell script HelloWorld.sh

```
#!/usr/bin/bash  
  
echo "Hello World"
```

- First commit

```
git add HelloWorld.sh  
git commit -m "Hello World"
```

## Working with Tags (continued)

- Create an annotated tag

```
git tag -a v1.0 -m "Final version of Hello World"
```

- List tags

```
git tag
```

```
v1.0
```

# Working with Tags (continued)

- Review history

```
git log
```

```
commit db4257601ac325d47bf35f9998ab439fd3d78f41 (HEAD -> master, tag: v1.0)
Author: Holger Obermaier <Holger.Obermaier@kit.edu>
Date: Tue Nov 9 16:15:30 2021 +0100
```

```
Hello World
```



## Working with Tags (continued)

- Tags can be used as references to commits (e.g. in show command)

```
git show v1.0
```

```
tag v1.0
Tagger: Holger Obermaier <Holger.Obermaier@kit.edu>
Date:   Tue Nov 9 16:20:37 2021 +0100

Final version of Hello World

commit db4257601ac325d47bf35f9998ab439fd3d78f41 (HEAD -> master, tag: v1.0)
Author: Holger Obermaier <Holger.Obermaier@kit.edu>
Date:   Tue Nov 9 16:15:30 2021 +0100

Hello World
```

## Working with Tags (continued)

- Delete a tag

```
git tag --delete v1.0
```

```
Deleted tag 'v1.0' (was 45bf7a)
```

- Caution: Tags are not transferred to remote servers by default

# Working with Branches

## Git Branching "Philosophy"

- Create branches for each feature  $\Rightarrow$  parallel development branches
- Avoid clashes caused by making changes on the master branch
- Commit often to track small changes
- Merge changes as one feature update (Squash commits)

# Working with Branches (continued)

- Create an empty repository

```
git init WorkingWithBranches
cd WorkingWithBranches
```

- Create an executable shell script HelloWorld.sh

```
#!/usr/bin/bash

echo "Hello World"
```

- First commit on master branch

```
git add HelloWorld.sh
git commit -m "Hello World"
```

## Working with Branches (continued)

- Create new branch HelloWorldForCats

```
git branch HelloWorldForCats  
git branch
```

```
HelloWorldForCats  
* master
```

- Switch to branch HelloWorldForCats

```
git switch HelloWorldForCats  
git branch
```

```
* HelloWorldForCats  
master
```

- `git switch` / restore separate functionality of the overloaded `git checkout` command

## Working with Branches (continued)

- Add feature helloWorldForCats to script HelloWorld.sh

```
#!/usr/bin/bash

case "$1" in
  cats) echo "Hello World for Cats" ;;
  *)   echo "Hello World"           ;;
esac
```

- Commit changes to branch helloWorldForCats

```
git commit -m "Hello World for Cats" HelloWorld.sh
```

## Working with Branches (continued)

- Return to master branch

```
git switch master
```

- Create and switch to new branch HelloWorldForDogs

```
git switch -c HelloWorldForDogs  
git branch
```

```
HelloWorldForCats  
* HelloWorldForDogs  
master
```

## Working with Branches (continued)

- Add feature helloWorldForDogs to script HelloWorld.sh

```
#!/usr/bin/bash

case "$1" in
  dogs) echo "Hello World for Dogs" ;;
  *) echo "Hello World" ;
esac
```

- Commit changes to branch helloWorldForDogs

```
git commit -m "Hello World for Dogs" HelloWorld.sh
```



## Working with Branches (continued)

- Return to master branch

```
git switch master
```

- Integrate feature developed on branch HelloWorldForCats

```
git merge HelloWorldForCats
```

```
Updating 148a364..01de454
Fast-forward
 HelloWorld.sh | 5 ++++-
 1 file changed, 4 insertions(+), 1 deletion(-)
```

## Working with Branches (continued)

- Integrate feature developed on branch HelloWorldForDogs

```
git merge HelloWorldForDogs
```

```
Auto-merging HelloWorld.sh  
CONFLICT (content): Merge conflict in HelloWorld.sh  
Automatic merge failed; fix conflicts and then commit the result.
```

## Working with Branches (continued)

- Review repository status

```
git status
```

```
On branch master
```

```
You have unmerged paths.
```

```
  (fix conflicts and run "git commit")
```

```
  (use "git merge --abort" to abort the merge)
```

```
Unmerged paths:
```

```
  (use "git add <file>..." to mark resolution)
```

```
    both modified:   HelloWorld.sh
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

## Working with Branches (continued)

- Conflict markers in the script HelloWorld.sh help to resolve the conflicts

```
#!/usr/bin/bash

case "$1" in
<<<<<<< HEAD
  cats) echo "Hello World for Cats" ;;
=====
  dogs) echo "Hello World for Dogs" ;;
>>>>>>> HelloWorldForDogs
  *) echo "Hello World" ;;
esac
```

# Working with Branches (continued)

- Review changes

```
git diff
```

```
diff --cc HelloWorld.sh
index 7fd1fa7,ea975f8..0000000
--- a/HelloWorld.sh
+++ b/HelloWorld.sh
@@@ -1,6 -1,6 +1,7 @@@
  #!/usr/bin/bash

  case "$1" in
+ cats) echo "Hello World for Cats" ;;
+ dogs) echo "Hello World for Dogs" ;;
    *) echo "Hello World"      ;;
  esac
```

## Working with Branches (continued)

- Continue integrating feature from branch HelloWorldForDogs

```
git add HelloWorld.sh  
git merge --continue
```

```
[master e8dbfdc] Merge branch 'HelloWorldForDogs'
```

- Merge can be aborted, when unable to resolve conflicts

```
git merge --abort
```

# Working with Branches (continued)

- Review history

```
git log --all --graph --oneline
```

```
* e8dbfdc (HEAD -> master) Merge branch 'HelloWorldForDogs'  
|\n| * 4e3387b (HelloWorldForDogs) Hello World for Dogs  
* | 01de454 (HelloWorldForCats) Hello World for Cats  
|/  
* 148a364 Hello World
```

# Working with Branches (continued)

- Remove unneeded branches

```
git branch --delete HelloWorldForCats HelloWorldForDogs  
git branch
```

```
* master
```

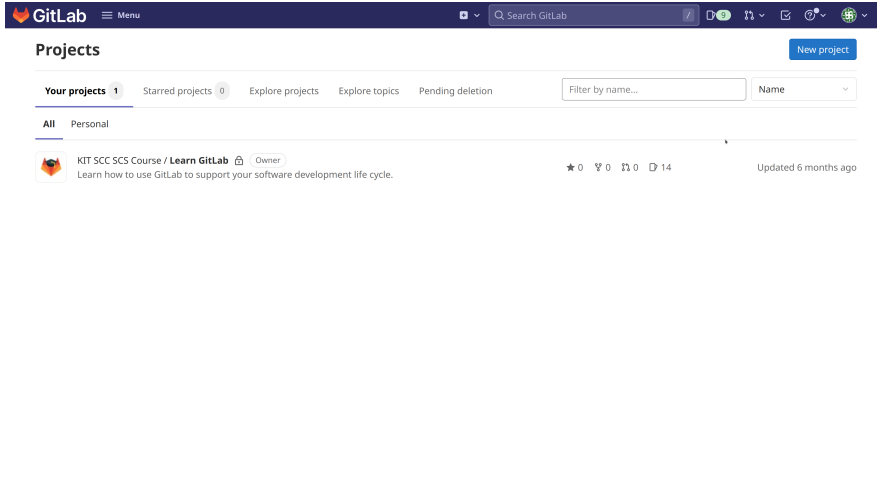
- Caution: Branches are not transferred to remote servers by default



# Working with Remote Servers

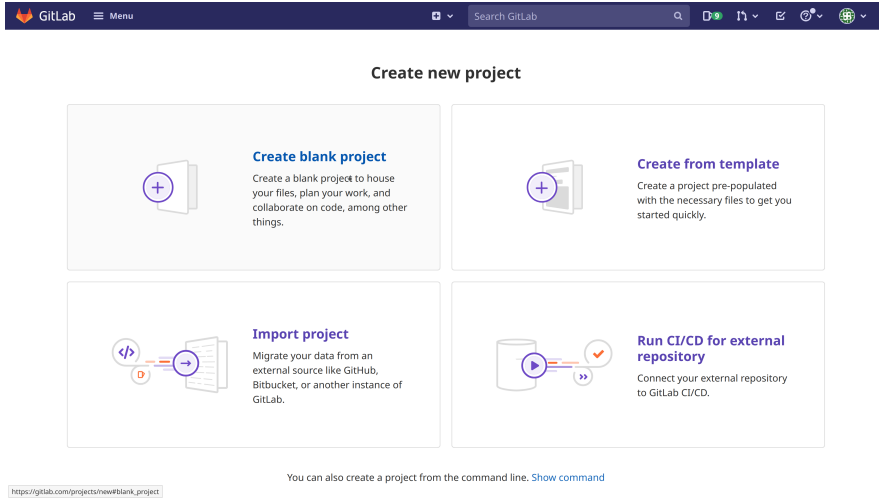
- Collaboration
- Code sharing / distribution
- All repositories are equal: No distinction between local and remote repository
- Git provides commands to keep local and remote repositories in sync
- Hosting services: Bitbucket [↗](#), GitHub [↗](#), GitLab [↗](#), ...
- Private and public repositories
- Transport protocol
  - `ssh://git@` , authentication via ssh-key, all users use the same remote username git
  - `https://` , authentication via username / password, token or without
  - `git@<host>:<path>` scp syntax for ssh-protocol

# Working with Remote Servers (GitLab: Projects)



The screenshot shows the GitLab web interface. At the top is a dark blue navigation bar with the GitLab logo, a menu icon, a search bar, and several utility icons. Below the navigation bar is the 'Projects' section. It features a 'New project' button in the top right corner. Underneath, there are tabs for 'Your projects' (1), 'Starred projects' (0), 'Explore projects', 'Explore topics', and 'Pending deletion'. A search box labeled 'Filter by name...' and a dropdown menu labeled 'Name' are also present. Below these are two tabs: 'All' (selected) and 'Personal'. The main content area displays a single project card for 'KIT SCC SCS Course / Learn GitLab'. The card includes a small GitLab icon, the project name, an 'Owner' role indicator, a description 'Learn how to use GitLab to support your software development life cycle.', and statistics: 0 stars, 0 forks, 0 issues, and 14 commits. The project was last updated 6 months ago.

# Working with Remote Servers (GitLab: New Project)

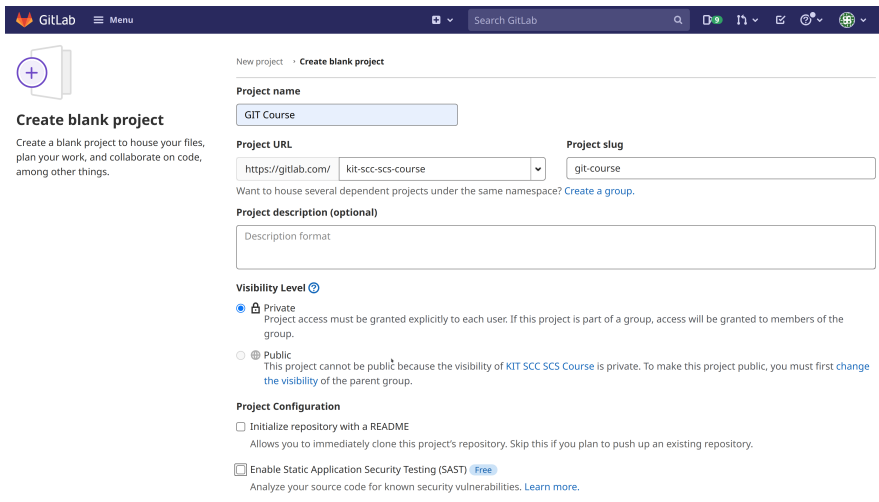


The screenshot shows the GitLab web interface for creating a new project. The header includes the GitLab logo, a menu icon, and a search bar. The main content area is titled 'Create new project' and contains four options:

- Create blank project**: Create a blank project to house your files, plan your work, and collaborate on code, among other things.
- Create from template**: Create a project pre-populated with the necessary files to get you started quickly.
- Import project**: Migrate your data from an external source like GitHub, Bitbucket, or another instance of GitLab.
- Run CI/CD for external repository**: Connect your external repository to GitLab CI/CD.

At the bottom, there is a note: 'You can also create a project from the command line. [Show command](#)' and a URL: [https://gitlab.com/projects/new#blank\\_project](https://gitlab.com/projects/new#blank_project)

# Working with Remote Servers (GitLab: Blank Project)



The screenshot shows the GitLab web interface for creating a new blank project. The top navigation bar includes the GitLab logo, a menu icon, a search bar, and several utility icons. On the left, there is a sidebar with a '+ Create blank project' button and a descriptive paragraph. The main content area is titled 'New project > Create blank project' and contains several form fields: 'Project name' (filled with 'GIT Course'), 'Project URL' (filled with 'https://gitlab.com/'), 'Project slug' (filled with 'git-course'), and 'Project description (optional)' (with a 'Description format' placeholder). Below these is the 'Visibility Level' section, where 'Private' is selected. At the bottom, the 'Project Configuration' section includes checkboxes for 'Initialize repository with a README' and 'Enable Static Application Security Testing (SAST)'. The SAST option is checked and has a 'Free' label.

GitLab Menu Search GitLab

New project > Create blank project

**Create blank project**

Create a blank project to house your files, plan your work, and collaborate on code, among other things.

**Project name**  
GIT Course

**Project URL** **Project slug**  
https://gitlab.com/ kit-scc-scs-course git-course

Want to house several dependent projects under the same namespace? [Create a group.](#)

**Project description (optional)**  
Description format

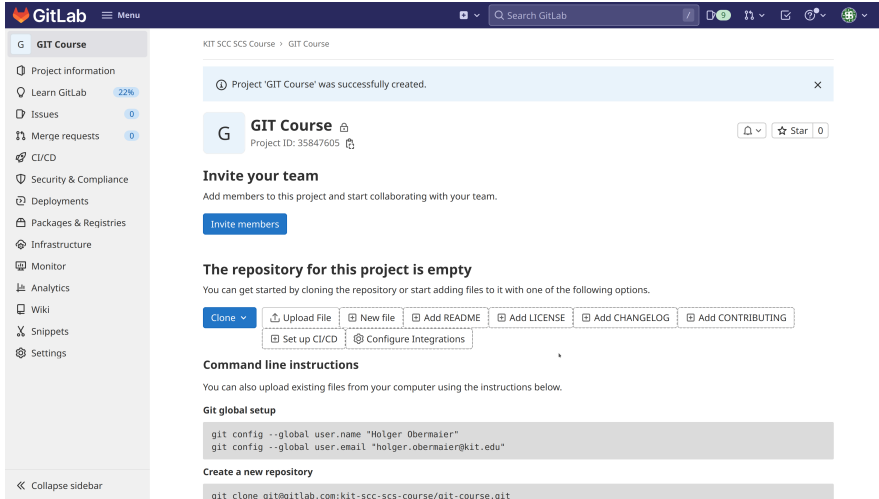
**Visibility Level**

- Private  
Project access must be granted explicitly to each user. If this project is part of a group, access will be granted to members of the group.
- Public  
This project cannot be public because the visibility of KIT SCC SCS Course is private. To make this project public, you must first [change the visibility](#) of the parent group.

**Project Configuration**

- Initialize repository with a README  
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.
- Enable Static Application Security Testing (SAST) **Free**  
Analyze your source code for known security vulnerabilities. [Learn more.](#)

# Working with Remote Servers (GitLab: Project created)



The screenshot shows the GitLab web interface for a newly created project named "GIT Course". The left sidebar contains navigation options such as "Project information", "Learn GitLab", "Issues", "Merge requests", "CI/CD", "Security & Compliance", "Deployments", "Packages & Registries", "Infrastructure", "Monitor", "Analytics", "Wiki", "Snippets", and "Settings". The main content area displays a success message: "Project 'GIT Course' was successfully created." Below this, the project name "GIT Course" is shown with its ID "35847605" and a lock icon. There are buttons for "Invite members", "Star", and "0" stars. The section "The repository for this project is empty" provides instructions on how to get started, including cloning the repository or adding files. A "Clone" dropdown menu is visible, along with buttons for "Upload File", "New file", "Add README", "Add LICENSE", "Add CHANGELOG", "Add CONTRIBUTING", "Set up CI/CD", and "Configure Integrations". The "Command line instructions" section provides the following commands:

```
git config --global user.name "Holger Obermaier"
git config --global user.email "holger.obermaier@kit.edu"
```

Under "Create a new repository", the following command is shown:

```
git clone git@gitlab.com:kit-scc-scs-course/git-course.git
```

## Working with Remote Servers (continued)

- Clone a remote GitLab repository via ssh or https protocol

```
git clone git@gitlab.com:kit-scc-scs-course/git-course.git
```

```
git clone https://gitlab.com/kit-scc-scs-course/git-course.git
```

```
Cloning into 'git-course'...
Username for 'https://gitlab.com': holger.obermaier
Password for 'https://holger.obermaier@gitlab.com':
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

## Working with Remote Servers (continued)

- Remote servers have a name and a corresponding URL
- Default name: `origin`
- Multiple remote servers can be configured. All git commands only use one at a time.
- Show name and URL of the remote server.

```
cd git-course  
git remote -v
```

```
origin git@gitlab.com:kit-scc-scs-course/git-course.git (fetch)  
origin git@gitlab.com:kit-scc-scs-course/git-course.git (push)
```

## Working with Remote Servers (continued)

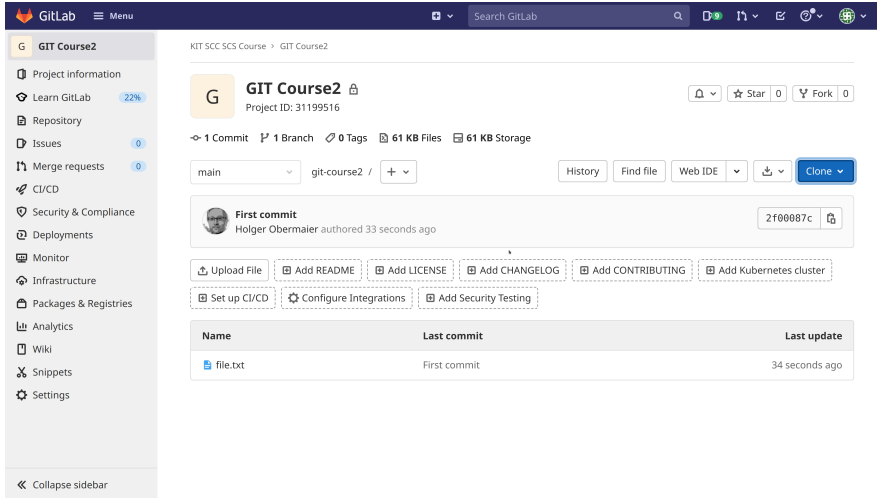
- Create a local commit and send changes to the remote server

```
echo "First commit" > file.txt
git add file.txt
git commit -m "First commit"
git push
```

```
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 290 bytes | 290.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To gitlab.com:kit-scc-scs-course/git-course.git
 29366ff..d1652b2  main -> main
```



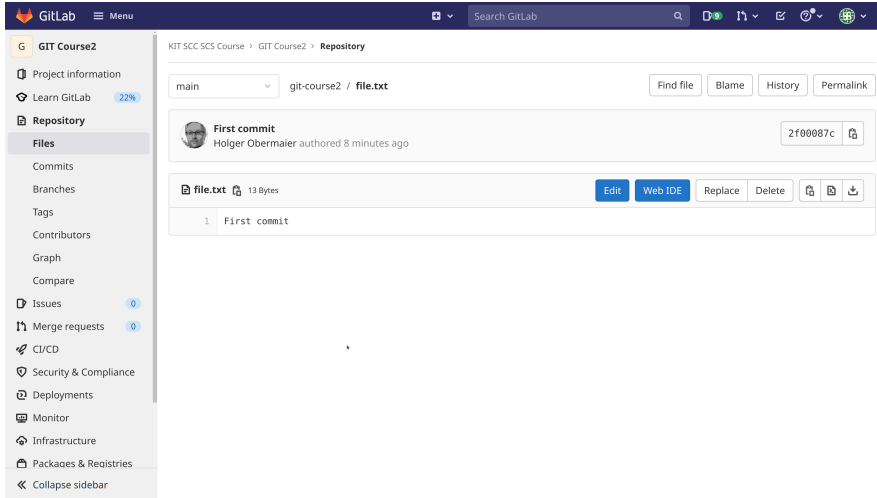
# Working with Remote Servers (GitLab: Project Page)



The screenshot shows the GitLab interface for a project named 'GIT Course2'. The left sidebar contains navigation options such as Project information, Learn GitLab (22%), Repository, Issues (0), Merge requests (0), CI/CD, Security & Compliance, Deployments, Monitor, Infrastructure, Packages & Registries, Analytics, Wiki, Snippets, and Settings. The main content area displays the project name, ID (31199516), and statistics (1 Commit, 1 Branch, 0 Tags, 61 KB Files, 61 KB Storage). It shows the current branch 'main' and the file path 'git-course2 /'. A 'Clone' button is visible. Below this, a 'First commit' by Holger Obermaier is shown, with a commit hash '2f00087c'. A list of actions is provided, including 'Upload File', 'Add README', 'Add LICENSE', 'Add CHANGELOG', 'Add CONTRIBUTING', 'Add Kubernetes cluster', 'Set up CI/CD', 'Configure Integrations', and 'Add Security Testing'. At the bottom, a table lists the files in the repository.

Name	Last commit	Last update
file.txt	First commit	34 seconds ago


# Working with Remote Servers (GitLab: file.txt)






The screenshot displays the GitLab web interface for a repository. The top navigation bar includes the GitLab logo, a menu icon, and a search bar. The left sidebar shows the project structure for 'GIT Course2', with 'Repository' and 'Files' selected. The main content area shows the 'file.txt' file view for the 'main' branch. It includes a commit history table with one entry: 'First commit' by Holger Obermaier, authored 8 minutes ago, with a commit hash of 2f00087c. The file content area shows the text '1 First commit'.

KIT SCC SCS Course > GIT Course2 > Repository

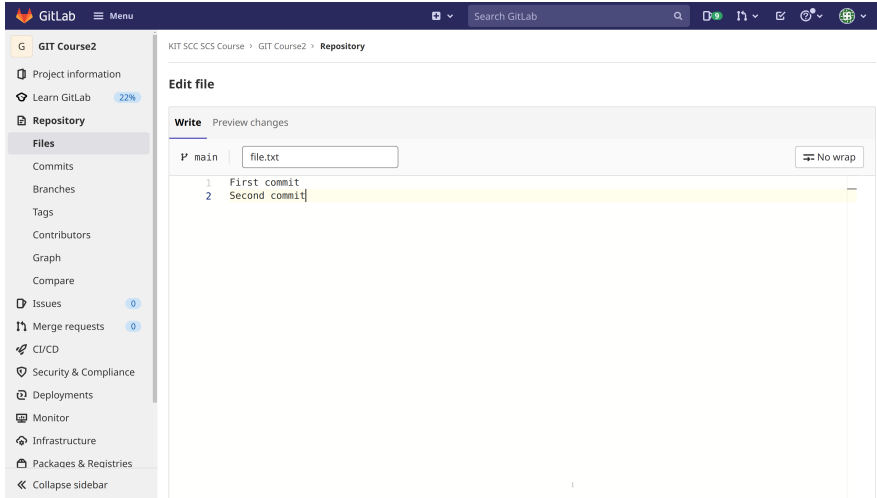
main git-course2 / file.txt Find file Blame History Permalink

 **First commit** 2f00087c  
Holger Obermaier authored 8 minutes ago

file.txt 13 Bytes Edit Web IDE Replace Delete   

1	First commit
---	--------------

# Working with Remote Servers (GitLab: Edit File)



GitLab Menu

KIT SCC SCS Course > GIT Course2 > Repository

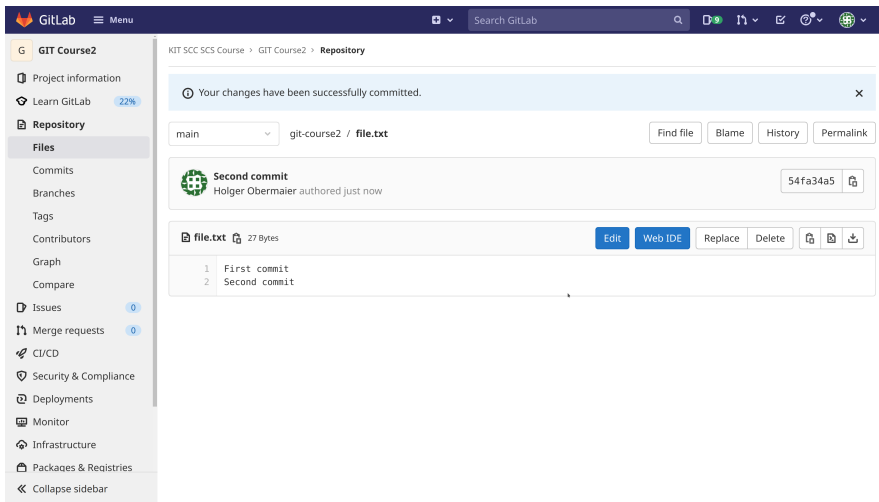
### Edit file

Write Preview changes

main file.txt No wrap

```
1 First commit
2 Second commit|
```

# Working with Remote Servers (GitLab: Committed)



The screenshot shows the GitLab web interface for a repository named 'git-course2'. The breadcrumb path is 'KIT SCC SCS Course > GIT Course2 > Repository'. A blue notification banner at the top states 'Your changes have been successfully committed.' Below this, the current branch is 'main' and the file path is 'git-course2 / file.txt'. Buttons for 'Find file', 'Blame', 'History', and 'Permalink' are visible. A commit summary for a 'Second commit' by 'Holger Obermaier' is shown with the commit hash '54fa34a5'. Below the commit, the file 'file.txt' (27 Bytes) is listed with buttons for 'Edit', 'Web IDE', 'Replace', and 'Delete'. The file content is displayed as a list of lines: '1 First commit' and '2 Second commit'.

## Working with Remote Servers (continued)

- Preparation: Insert one line into file `file.txt` on GitLab
- All git commands work local. Changes on the server are invisible
- Fetch changes from the remote repository

```
git fetch
```

```
remote: Enumerating objects: 5, done.  
remote: Counting objects: 100% (5/5), done.  
remote: Compressing objects: 100% (2/2), done.  
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0  
Unpacking objects: 100% (3/3), 281 bytes | 281.00 KiB/s, done.  
From gitlab.com:kit-scc-scs-course/git-course  
    d1652b2..61ddece  main      -> origin/main
```

## Working with Remote Servers (continued)

- Review status of the repository

```
git status
```

```
On branch main
```

```
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
```

```
(use "git pull" to update your local branch)
```

```
nothing to commit, working tree clean
```

# Working with Remote Servers (continued)

- Review changes made on the remote server

```
git diff HEAD @{upstream}
```

```
diff --git a/file.txt b/file.txt
index 6eaf244..6a3adff 100644
--- a/file.txt
+++ b/file.txt
@@ -1,2 @@
  First commit
+Second commit
```

## Working with Remote Servers (continued)

- Integrate changes made on the remote server

```
git pull
```

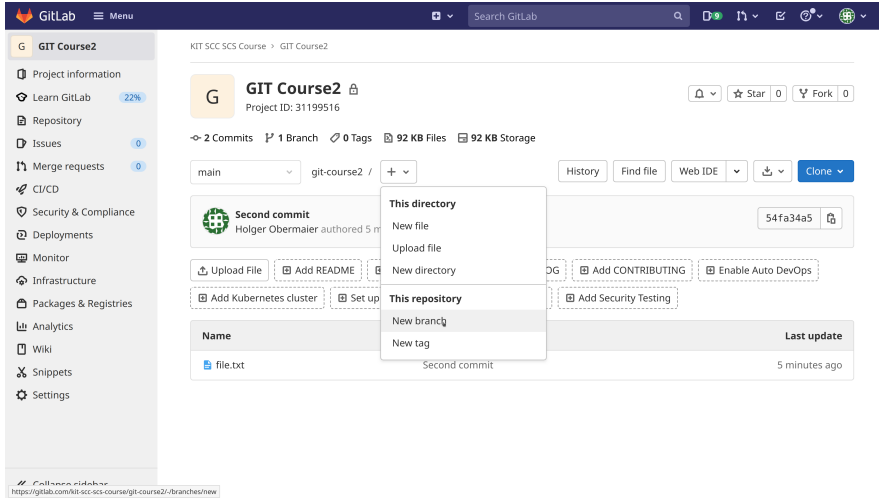
```
Updating d1652b2..61ddece  
Fast-forward  
 file.txt | 1  
 1 file changed, 1 insertion(+)
```

```
cat file.txt
```

```
First commit  
Second commit
```



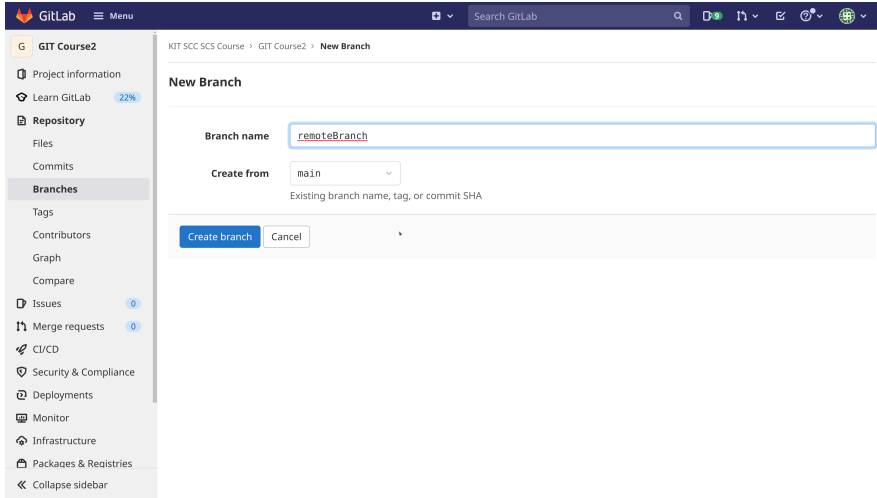
# Working with Remote Servers (GitLab: New Branch)



The screenshot shows the GitLab web interface for a project named "GIT Course2". The left sidebar contains navigation options like "Project information", "Learn GitLab", "Repository", "Issues", "Merge requests", "CI/CD", "Security & Compliance", "Deployments", "Monitor", "Infrastructure", "Packages & Registries", "Analytics", "Wiki", "Snippets", and "Settings". The main content area displays project statistics (2 Commits, 1 Branch, 0 Tags, 92 KB Files, 92 KB Storage) and a dropdown menu for branch selection. A "New branch" option is highlighted in the dropdown. Below the dropdown, a table lists the current branches, showing "file.txt" as the current branch and "Second commit" as a new branch created 5 minutes ago.

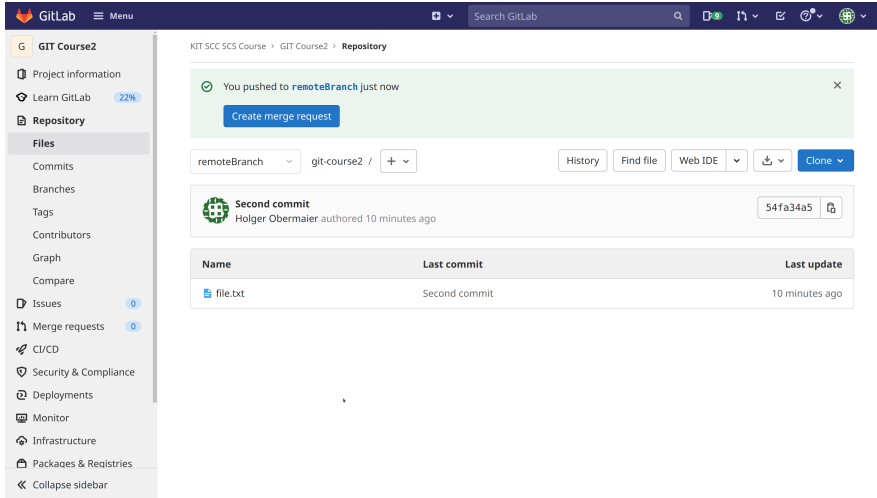
Name	Last update
file.txt	
Second commit	5 minutes ago

# Working with Remote Servers (GitLab: New Branch)



The screenshot shows the GitLab web interface for creating a new branch. The breadcrumb navigation at the top reads "KIT SCC SCS Course > GIT Course2 > New Branch". The main heading is "New Branch". There is a text input field for "Branch name" containing the text "remoteBranch". Below it is a dropdown menu for "Create from" with "main" selected. A note below the dropdown says "Existing branch name, tag, or commit SHA". At the bottom of the form are two buttons: "Create branch" (highlighted in blue) and "Cancel". The left sidebar contains a navigation menu with categories like "Repository", "Branches", "Issues", and "Merge requests".

# Working with Remote Servers (GitLab: New Branch)




KIT SCC SCS Course > GIT Course2 > Repository

✔ You pushed to `remoteBranch` just now


Create merge request

remoteBranch git-course2 / +

History Find file Web IDE Clone

 **Second commit** 54fa34a5

Holger Obermaier authored 10 minutes ago

Name	Last commit	Last update
 file.txt	Second commit	10 minutes ago

# Working with Remote Servers (remote branches)

- Preparation: Create a branch `remoteBranch` on GitLab
- Get remote branches

```
git pull
```

```
From gitlab.com:kit-scc-scs-course/git-course
* [new branch]      remoteBranch -> origin/remoteBranch
Already up to date.
```

- Track remote branch

```
git switch remoteBranch
```

```
Branch 'remoteBranch' set up to track remote branch 'remoteBranch' from
↪ 'origin'.
Switched to a new branch 'remoteBranch'
```

# Working with Remote Servers (local branches)

- Create a local branch and try sending it to the remote server

```
git switch -c localBranch  
git push
```

```
fatal: The current branch localBranch has no upstream branch.  
To push the current branch and set the remote as upstream, use
```

```
git push --set-upstream origin localBranch
```

# Working with Remote Servers (local branches)

- Send local branch to remote server

```
git push --set-upstream origin localBranch
```




```
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for localBranch, visit:
remote:
  ↪ https://gitlab.com/kit-scc-scs-course/git-course/-/merge_requests/new?merge_req
remote:
To gitlab.com:kit-scc-scs-course/git-course.git
 * [new branch]      localBranch -> localBranch
Branch 'localBranch' set up to track remote branch 'localBranch' from
  ↪ 'origin'.
```

# Questions?



<https://xkcd.com/1597/>

# References

- [1] S. Chacon, B. Straub, *Pro Git 2nd ed.*, 2014, Apress  
<https://git-scm.com/book/de/v2>
- [2] J. Abildskov, *Practical Git*, 2020, Apress
- [3] M. Tsitoara, *Beginning Git and GitHub*, 2020, Apress
- [4] GIT PURR! Git Commands Explained with Cats! 
- [5] How to explain git in simple words? 
- [6] Oh Shit, Git!?! 
- [7] The Missing Semester of Your CS Education: Lecture 1/22/20: Version Control (Git) 