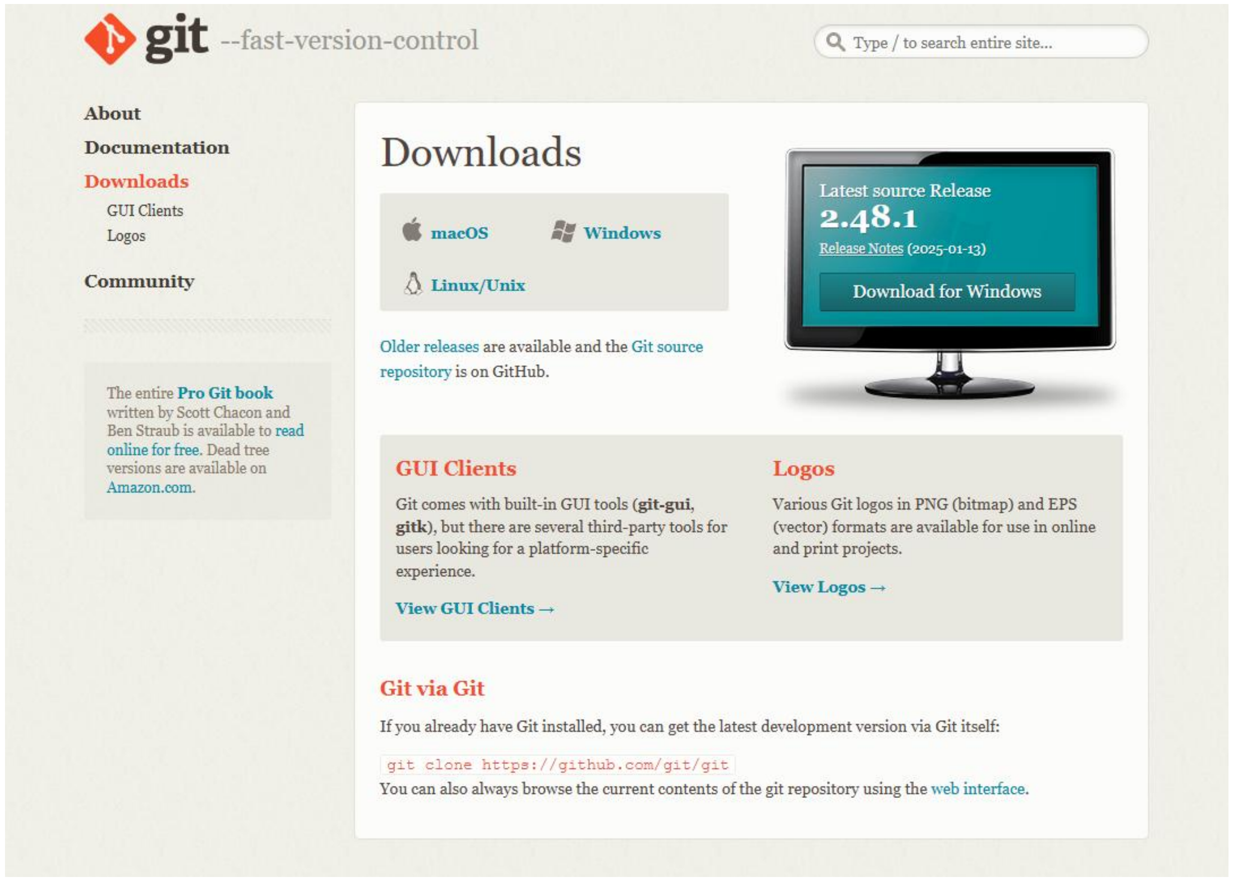
Exercise 1

BIF2

Install the Git Bash



Downloads

macOS Windows
Linux/Unix

Latest source Release
2.48.1
Release Notes (2025-01-13)
Download for Windows

Older releases are available and the Git source repository is on GitHub.

The entire Pro Git book written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

GUI Clients

Git comes with built-in GUI tools (**git-gui**, **gitk**), but there are several third-party tools for users looking for a platform-specific experience.

View GUI Clients →

Logos

Various Git logos in PNG (bitmap) and EPS (vector) formats are available for use in online and print projects.

View Logos →

Git via Git

If you already have Git installed, you can get the latest development version via Git itself:

```
git clone https://github.com/git/git
```

You can also always browse the current contents of the git repository using the web interface.

Link

https://git-scm.com/downloads

Operating System

Windows

The first repository

Create the first repository in a folder of your choice.

- What is the command for it?
- What are the options?
- What exactly happens in the process?

The first repository

COMMAND

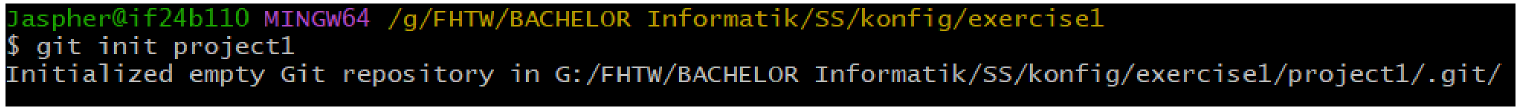
$ git init [directory]

- Creates new folder and initializes Git-Repository

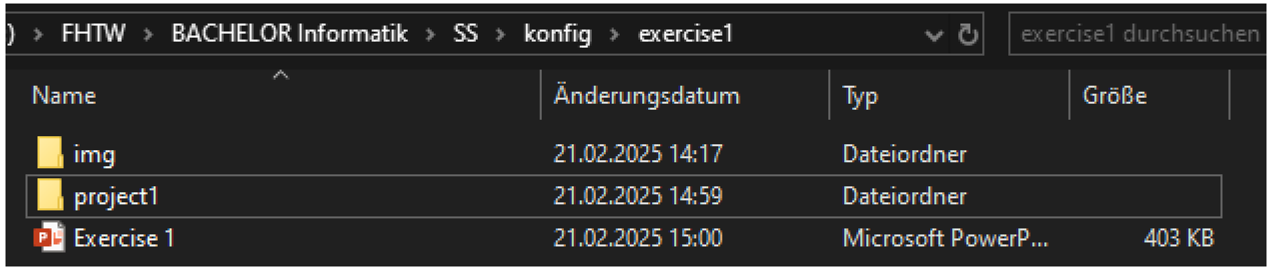
Other option

$ git init

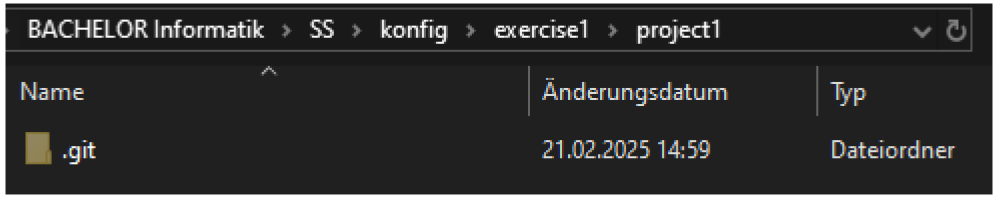
- *Current folder will be initialized as Git-Repository*



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1
$ git init project1
Initialized empty Git repository in G:/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1/.git/
```



Name	Änderungsdatum	Typ	Größe
img	21.02.2025 14:17	Dateiordner	
project1	21.02.2025 14:59	Dateiordner	
Exercise 1	21.02.2025 15:00	Microsoft PowerP...	403 KB



Name	Änderungsdatum	Typ
.git	21.02.2025 14:59	Dateiordner

Global config

To set up your name and email address one, configure it via git config.

- What is the full command?
- What options are there?
- Where are global setting stored in Git?

Global config

COMMAND

$ git config --global user.name "[name]"

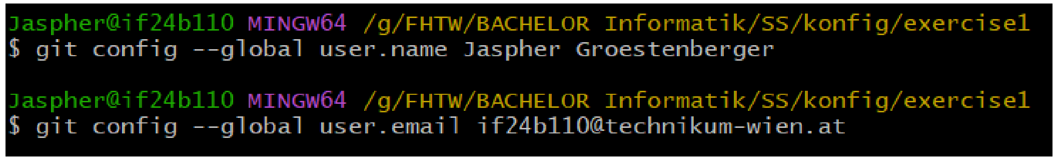
- user.name = Jaspher

$ git config --global user.name "[email adress]"

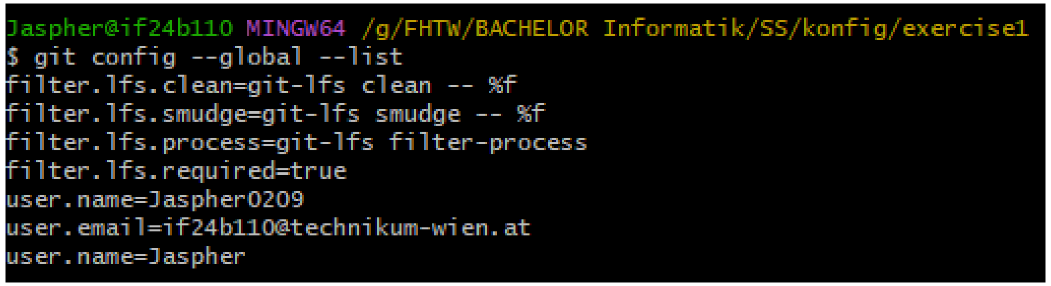
- user.name = if24b110@technikum-wien.at

$ git config --global --list

- Shows entire list of global configurations



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1
$ git config --global user.name Jaspher Groestenberger

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1
$ git config --global user.email if24b110@technikum-wien.at
```



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1
$ git config --global --list
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
user.name=Jaspher0209
user.email=if24b110@technikum-wien.at
user.name=Jaspher
```

Where are global setting stored in Git?
Inside ~/.gitconfig

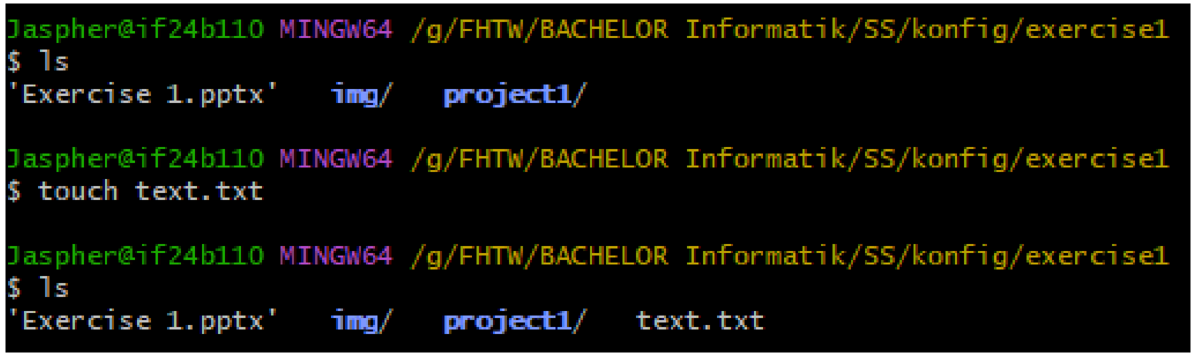
('~' is short for home-directory)

Text file

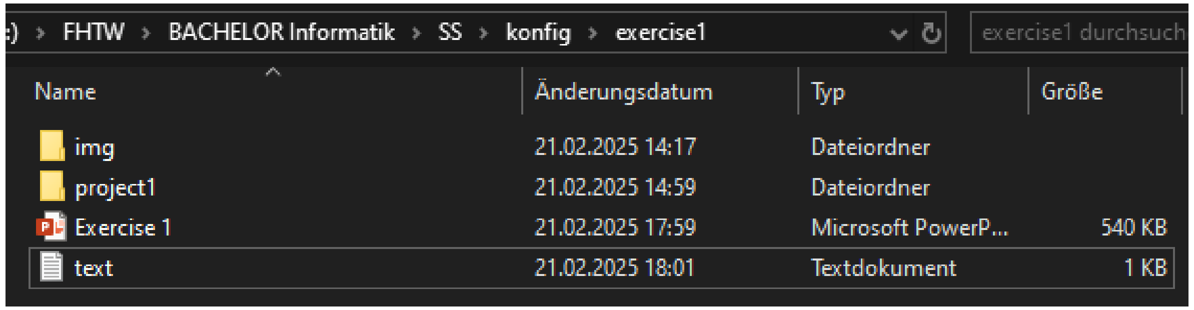
Create a text file and write into it.

Text file

Creating text file



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1
$ ls
'Exercise 1.pptx'   img/   project1/

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1
$ touch text.txt

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1
$ ls
'Exercise 1.pptx'   img/   project1/   text.txt
```

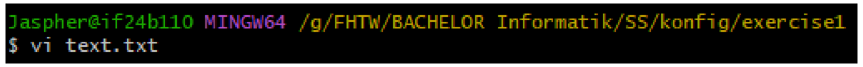


:) › FHTW › BACHELOR Informatik › SS › konfig › exercise1

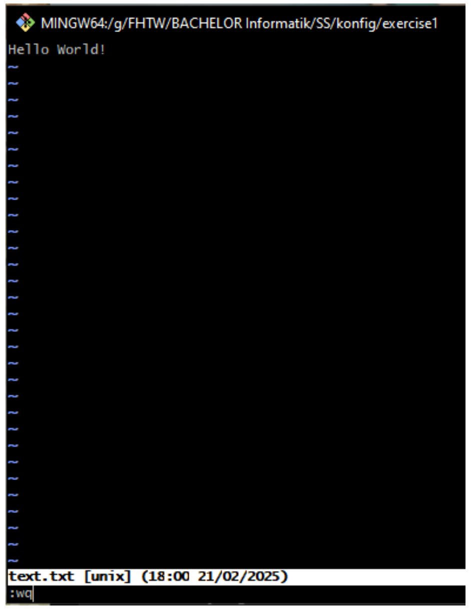
exercise1 durchsuch

Name	Änderungsdatum	Typ	Größe
img	21.02.2025 14:17	Dateiordner	
project1	21.02.2025 14:59	Dateiordner	
Exercise 1	21.02.2025 17:59	Microsoft PowerP...	540 KB
text	21.02.2025 18:01	Textdokument	1 KB

vi (Visual Editor)

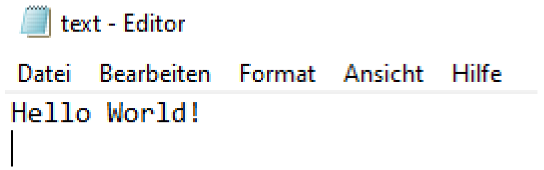


```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1
$ vi text.txt
```



MINGW64:/g/FHTW/BACHELOR Informatik/SS/konfig/exercise1

```
Hello World!
~
~
~
~
text.txt [unix] (18:00 21/02/2025)
:wq
```



text - Editor

Datei   Bearbeiten   Format   Ansicht   Hilfe

```
Hello World!
```

Status

On the Git Bash, check the status.

- What is the command for this?
- What result do you get and how do you interpret it?

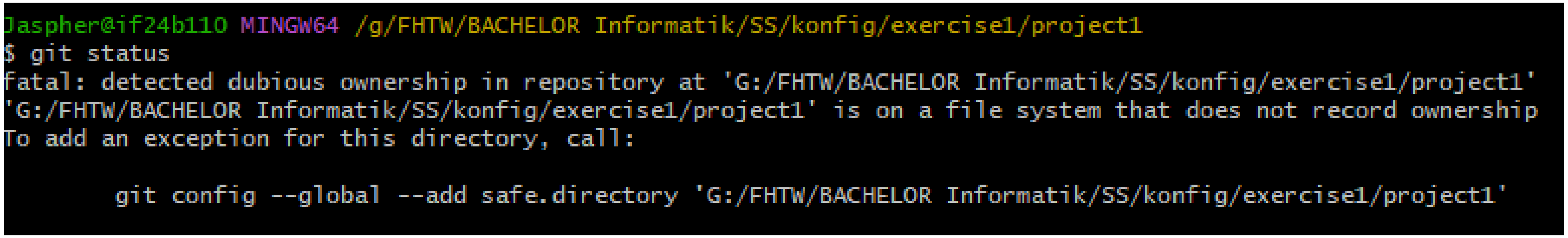
Global config

COMMAND

$ git status

Interpretation of result

Detect dubious ownership: Due to repository location in external drive



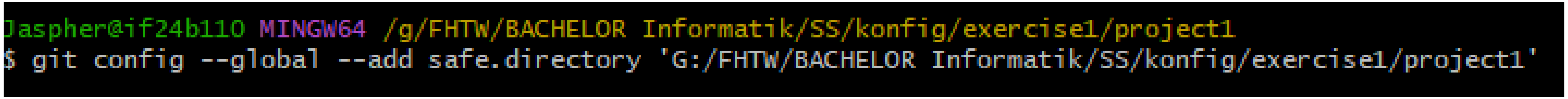
```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1
$ git status
fatal: detected dubious ownership in repository at 'G:/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1'
'G:/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1' is on a file system that does not record ownership
To add an exception for this directory, call:

        git config --global --add safe.directory 'G:/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1'
```

Global config

Solution

Command:

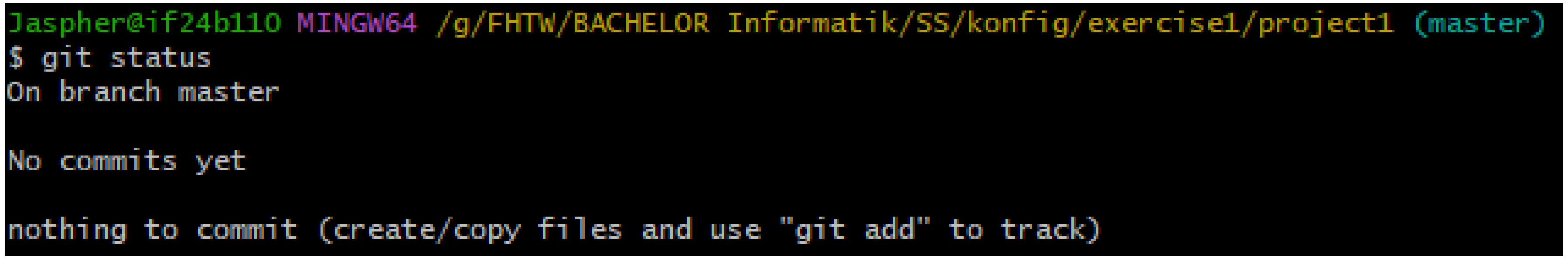
$ git config git config --global --add safe.directory 'G:/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1'



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1
$ git config --global --add safe.directory 'G:/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1'
```

Global config

Repeating action



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git status
On branch master

No commits yet

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

Interpretation of result

- (master) appears next to directory-path
- Git-repository is located on the branch “master” (Standard-Branch) > *‘On branch master’*
- Shows that there are no commits yet > *‘No commits yet’*
- No existing changes to commit > *‘nothing to commit’*

Put file in the repository

Put file in the repository.

- Now What are the commands for doing this?
- What steps are necessary and what results do you get?
- How do you interpret the output?
- Where does Git store the files?

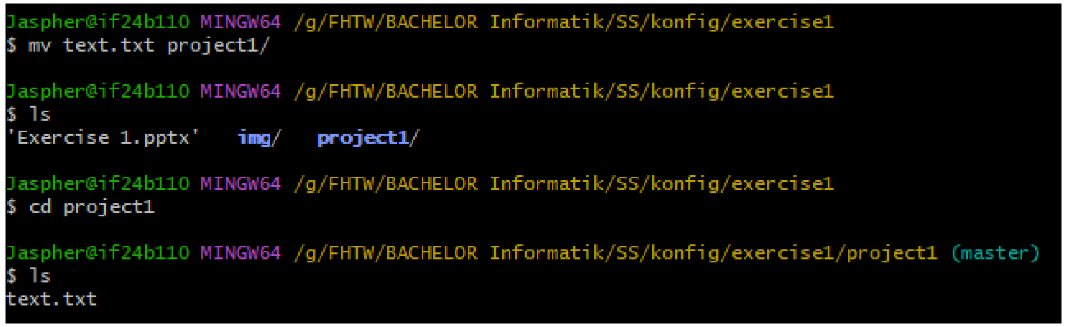
Put file in the repository

COMMAND

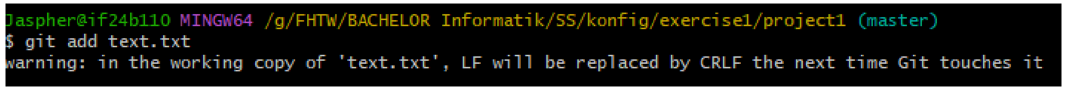
$ git add [filename]

Steps adding file to repository

- File should be inside repository



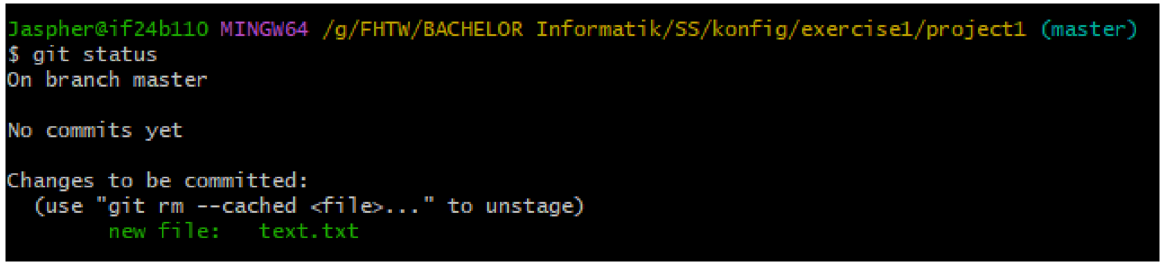
```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1
$ mv text.txt project1/

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1
$ ls
'Exercise 1.pptx'   img/   project1/

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1
$ cd project1

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ ls
text.txt
```



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git add text.txt
warning: in the working copy of 'text.txt', LF will be replaced by CRLF the next time Git touches it
```

Result

- File is now in Staging-Area – *Ready to commit*
- It was untracked first – Files that Git does not know about
- Status check with ***$ git status***



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   text.txt
```

SHA1

You have now created Git Objects.

- Which ones (according to the Data Model) and where are they located?
- How can you find the SHA1 key of the blob object?
- What is the command for this?

SHA1

COMMAND

$ git ls-files --stage [filename]

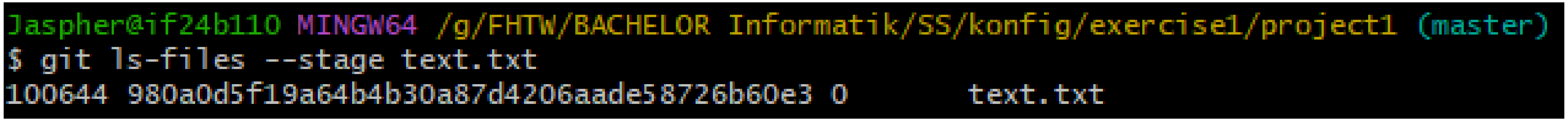
Lists all files Git is currently tracking

--stage is to outputs staging information plus SHA1 hash of object

Objects

- Blob (Binary Large Object)
- Tree
- Commit
- Tag

Location: .git/objects/



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git ls-files --stage text.txt
100644 980a0d5f19a64b4b30a87d4206aade58726b60e3 0       text.txt
```

Search and output via SHA1

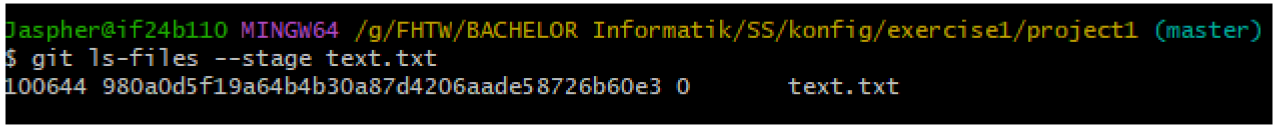
- How can you output the contents of the file only by specifying the SHA1 key?
- What is the command and what options does it provide?

SHA1

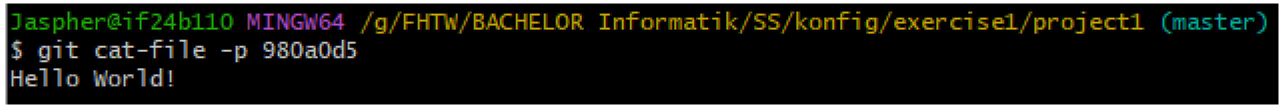
COMMAND

$ git cat-file -p <SHA1>

- Outputs content of file by specifying SHA1
- cat-file: outputs content
- -p: outputs readable content of specified object

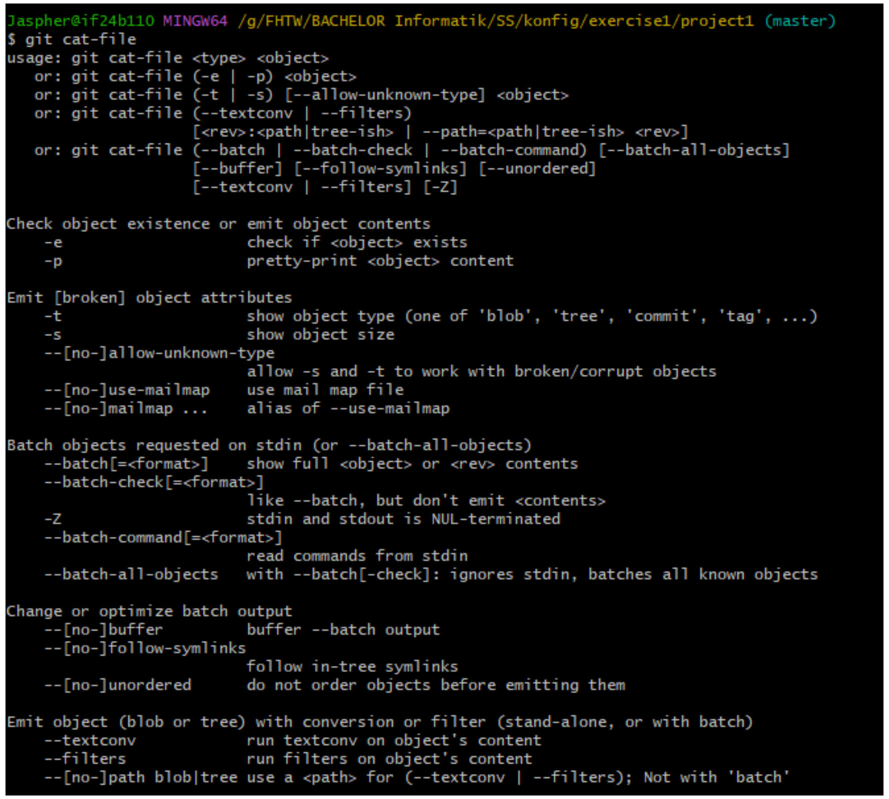


```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git ls-files --stage text.txt
100644 980a0d5f19a64b4b30a87d4206aade58726b60e3 0       text.txt
```



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git cat-file -p 980a0d5
Hello World!
```

Options



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git cat-file
usage: git cat-file <type> <object>
   or: git cat-file (-e | -p) <object>
   or: git cat-file (-t | -s) [--allow-unknown-type] <object>
   or: git cat-file (--textconv | --filters)
                    [<rev>:<path|tree-ish> | --path=<path|tree-ish> <rev>]
   or: git cat-file (--batch | --batch-check | --batch-command) [--batch-all-objects]
                    [--buffer] [--follow-symlinks] [--unordered]
                    [--textconv | --filters] [-Z]

Check object existence or emit object contents
    -e                    check if <object> exists
    -p                    pretty-print <object> content

Emit [broken] object attributes
    -t                    show object type (one of 'blob', 'tree', 'commit', 'tag', ...)
    -s                    show object size
    --[no-]allow-unknown-type
                          allow -s and -t to work with broken/corrupt objects
    --[no-]use-mailmap    use mail map file
    --[no-]mailmap ...    alias of --use-mailmap

Batch objects requested on stdin (or --batch-all-objects)
    --batch[=<format>]    show full <object> or <rev> contents
    --batch-check[=<format>]
                          like --batch, but don't emit <contents>
    -Z                    stdin and stdout is NUL-terminated
    --batch-command[=<format>]
                          read commands from stdin
    --batch-all-objects   with --batch[-check]: ignores stdin, batches all known objects

Change or optimize batch output
    --[no-]buffer         buffer --batch output
    --[no-]follow-symlinks
                          follow in-tree symlinks
    --[no-]unordered      do not order objects before emitting them

Emit object (blob or tree) with conversion or filter (stand-alone, or with batch)
    --textconv            run textconv on object's content
    --filters             run filters on object's content
    --[no-]path blob|tree use a <path> for (--textconv | --filters); Not with 'batch'
```

SHA1 Hello World

- What is the SHA1 key of the text “Hello World!”?
- Do not create a separate file for this!
- What is the command for this?

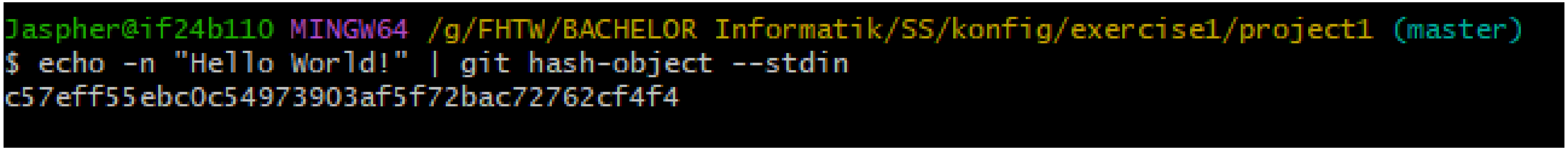
SHA1 Hello World

COMMAND

$ git hash-object [option] <filename>

Command to hash text content

- Echoes text to console
- -git hash-object --stdin hashes the standard input of console



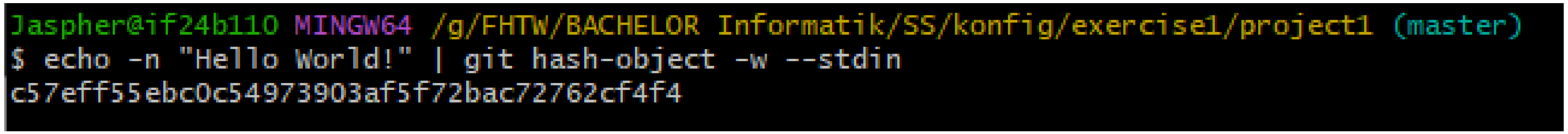
```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ echo -n "Hello World!" | git hash-object --stdin
c57eff55ebc0c54973903af5f72bac72762cf4f4
```

Save Hello World

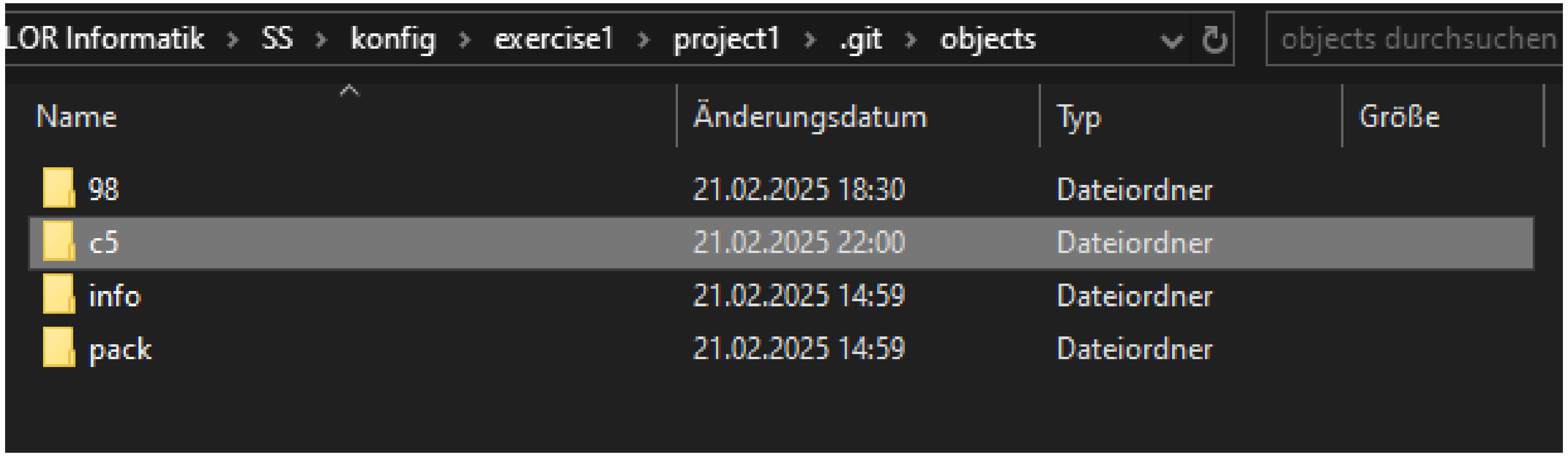
Save the text “Hello World!” in the repository without creating a file for it.

Save Hello World

“Hello World!” saved in repository



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ echo -n "Hello World!" | git hash-object -w --stdin
c57eff55ebc0c54973903af5f72bac72762cf4f4
```



LOR Informatik › SS › konfig › exercise1 › project1 › .git › objects

objects durchsuchen

Name	Änderungsdatum	Typ	Größe
98	21.02.2025 18:30	Dateiordner	
c5	21.02.2025 22:00	Dateiordner	
info	21.02.2025 14:59	Dateiordner	
pack	21.02.2025 14:59	Dateiordner	

Blobs

<u>You have now created the first 2 blob objects in the repository. Now create the first commit object.</u>

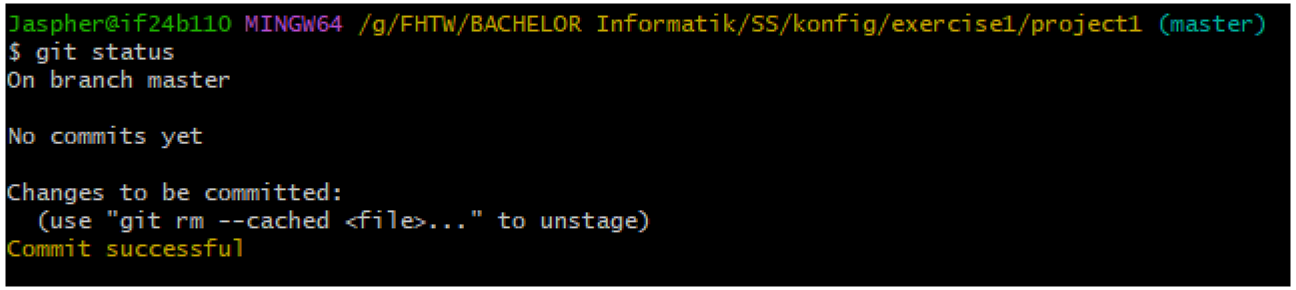
- Which command do you use it for it?
- What options does it offer?
- Where are commit objects stored?

Blobs

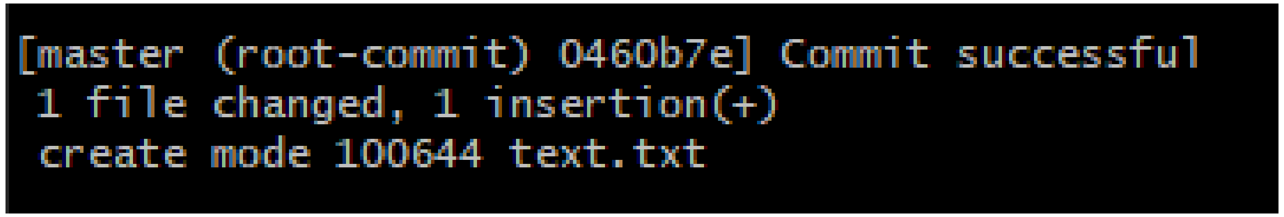
COMMAND

$ git commit –m "[descriptive message]"

- -m outputs descriptive message after commit

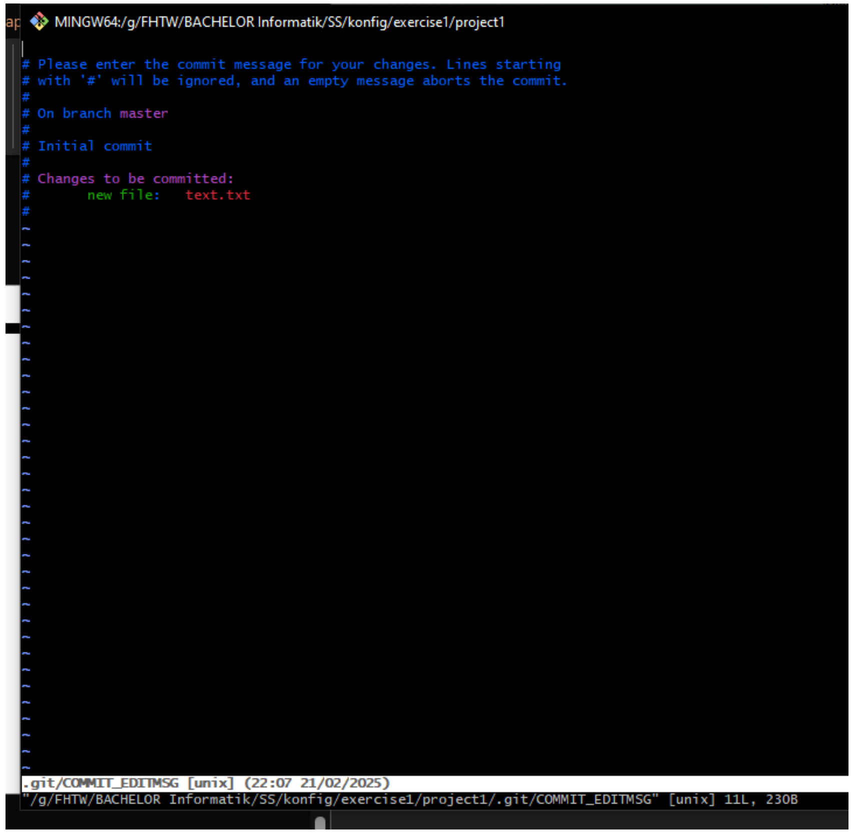


```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
Commit successful
```



```
[master (root-commit) 0460b7e] Commit successful
 1 file changed, 1 insertion(+)
 create mode 100644 text.txt
```

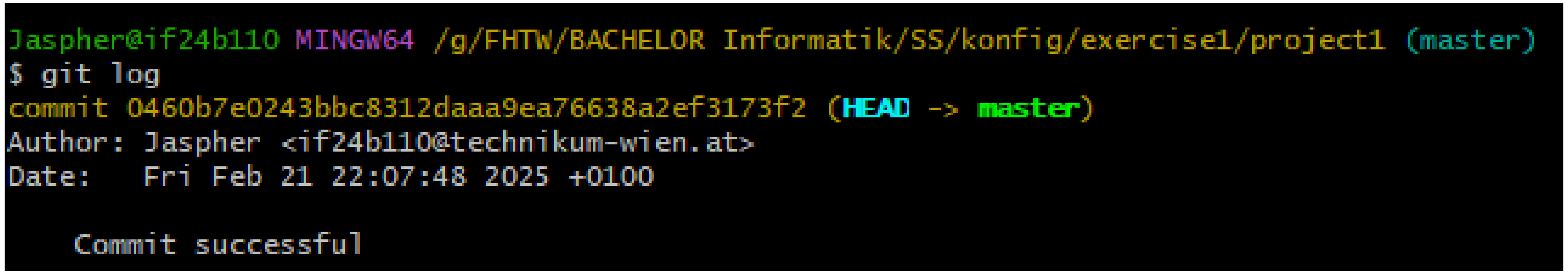
$git commit without -m



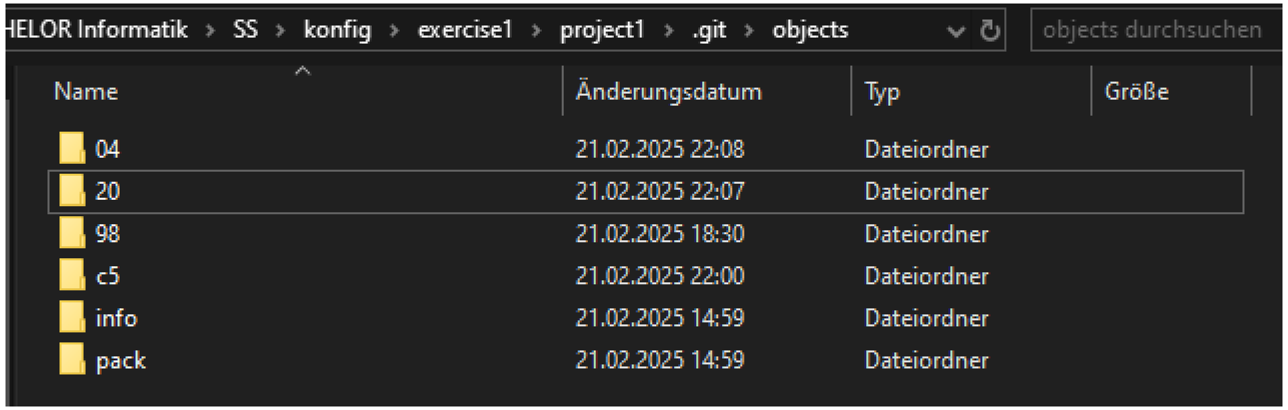
```
MINGW64:/g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   text.txt
#
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
.git/COMMIT_EDITMSG [unix] (22:07 21/02/2025)
"/g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1/.git/COMMIT_EDITMSG" [unix] 11L, 230B
```

Blobs

Commit stored in: ./git/objects



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git log
commit 0460b7e0243bbc8312daaa9ea76638a2ef3173f2 (HEAD -> master)
Author: Jaspher <if24b110@technikum-wien.at>
Date:   Fri Feb 21 22:07:48 2025 +0100

    Commit successful
```



HELOR Informatik › SS › konfig › exercise1 › project1 › .git › objects

objects durchsuchen

Name	Änderungsdatum	Typ	Größe
04	21.02.2025 22:08	Dateiordner	
20	21.02.2025 22:07	Dateiordner	
98	21.02.2025 18:30	Dateiordner	
c5	21.02.2025 22:00	Dateiordner	
info	21.02.2025 14:59	Dateiordner	
pack	21.02.2025 14:59	Dateiordner	

It’s the 04 directory

History output

In order to be able to display the history on the screen, you must use:

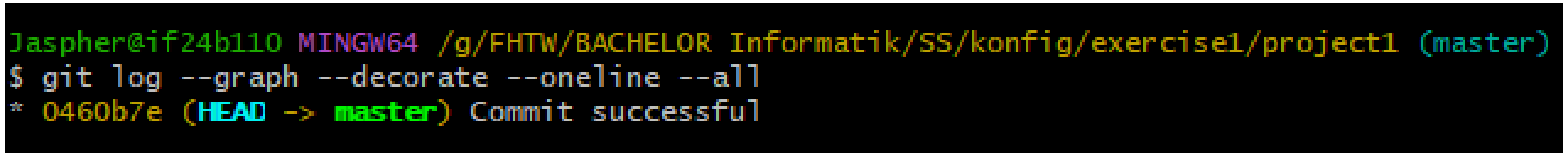
- Enter command: git log --graph --decorate --oneline --all
- We don’t type long command everytime -> new alias(git lol)
- What command do you enter for this?

History output

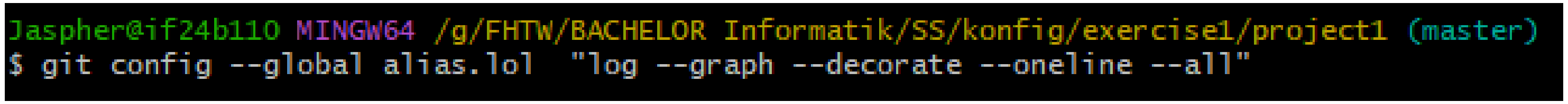
COMMAND

$ git config –global alias.<alias-name> “<git-command>

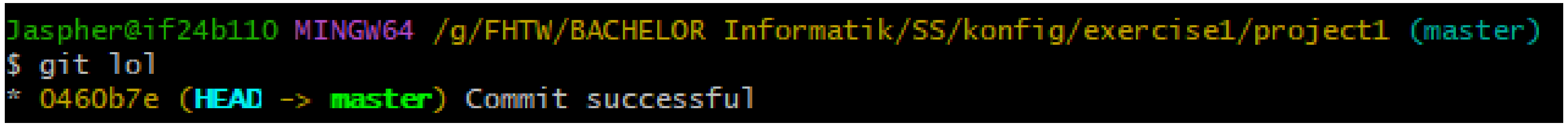
Configurates new alias for a git command



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git log --graph --decorate --oneline --all
* 0460b7e (HEAD -> master) Commit successful
```



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git config --global alias.lol  "log --graph --decorate --oneline --all"
```

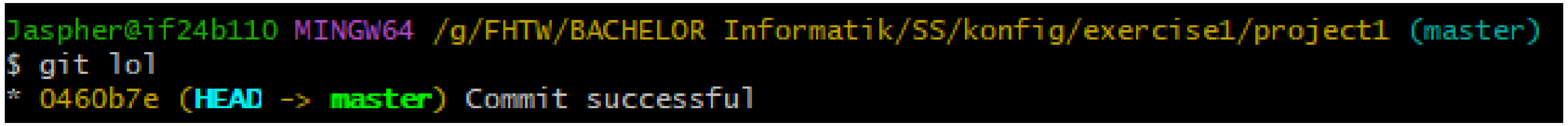


```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git lol
* 0460b7e (HEAD -> master) Commit successful
```

History

With git lol we can now output the history any time.

- What does the history look like?



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git lol
* 0460b7e (HEAD -> master) Commit successful
```

0460b7e: hash-value of commit object

(HEAD -> master): HEAD points on master branch

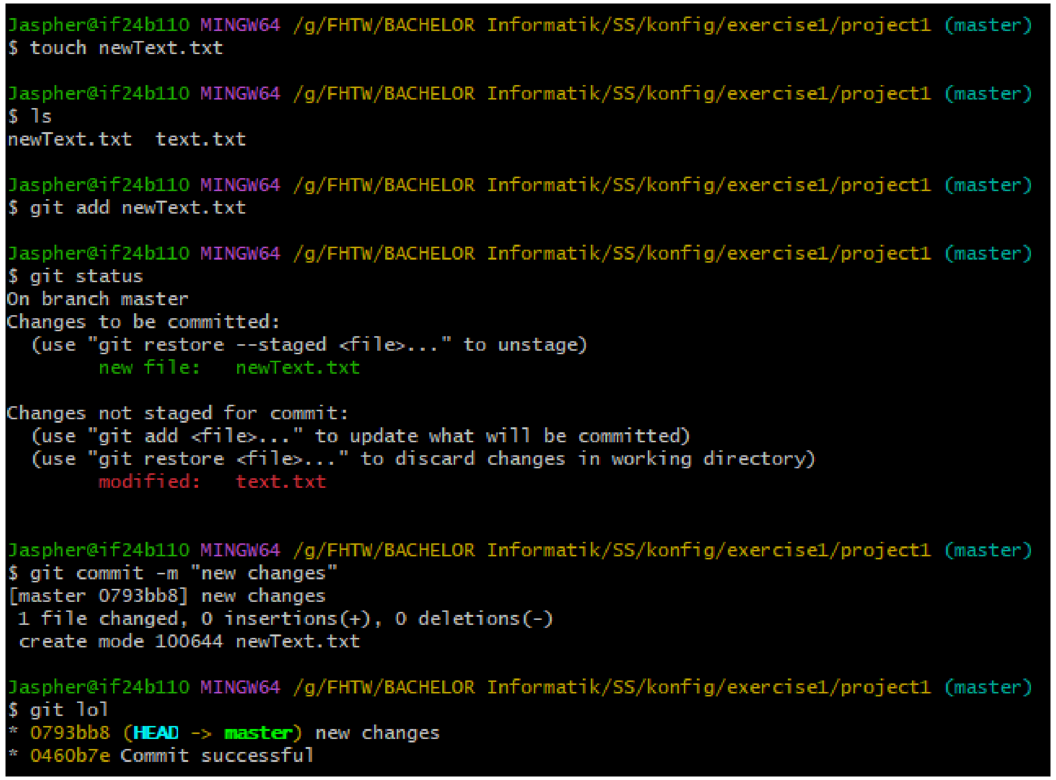
Commit successful: previous commit descriptive message

New file

In Create another file and add it to the repository. Also create new version of your repository.

New file

New file added to repository and creates new version of repository.



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ touch newText.txt

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ ls
newText.txt  text.txt

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git add newText.txt

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   newText.txt

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:   text.txt


Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git commit -m "new changes"
[master 0793bb8] new changes
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newText.txt

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git lol
* 0793bb8 (HEAD -> master) new changes
* 0460b7e Commit successful
```

Commit object

Oops, now you have forgotten a file.

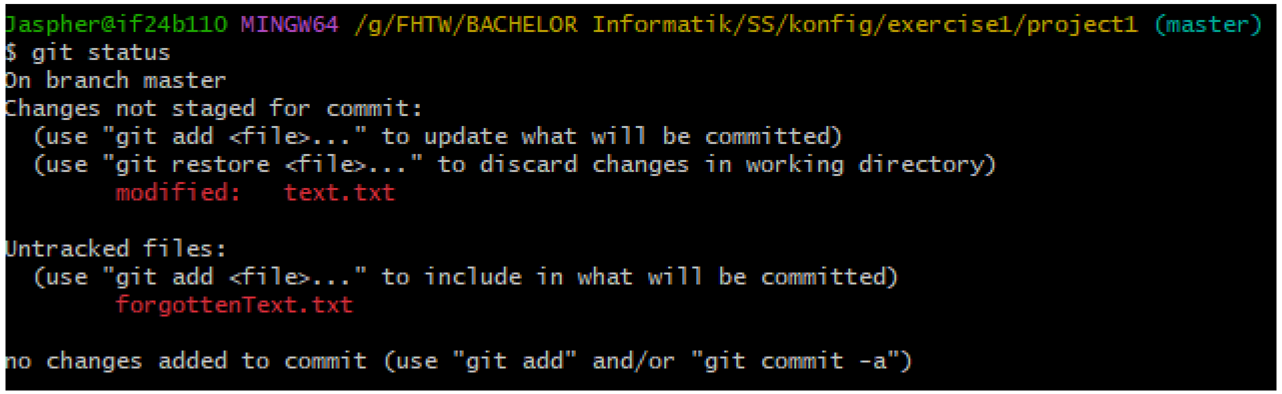
- Create a third text file and add this file to the repository.
- But make sure to that this file still goes into the same version of the history by using the amend command.
- Does git create a new commit object?
- If so, why?

Commit object

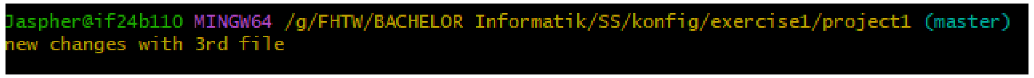
COMMAND

$ git commit --amend –m "[descriptive message]"

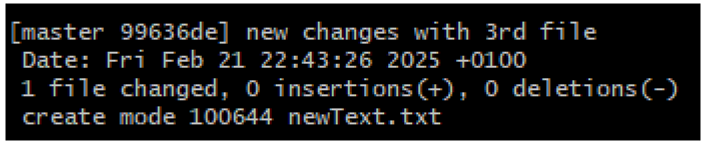
- A way to make quick corrections without creating new commits.
- Fixing commit message or adding new changes



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ 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:   text.txt

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

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



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
new changes with 3rd file
```



```
[master 99636de] new changes with 3rd file
 Date: Fri Feb 21 22:43:26 2025 +0100
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newText.txt
```

Does it create a new commit object?

Technically yes, it does but replaces the previous version in the history.

Delete

Delete a file and then restore it using commands.

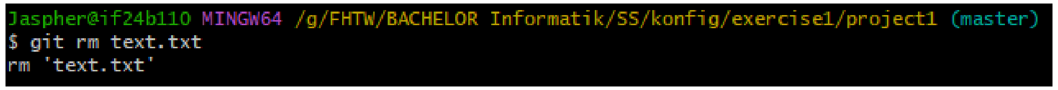
- What command do you use for this?
- From where is the file restored?

Delete

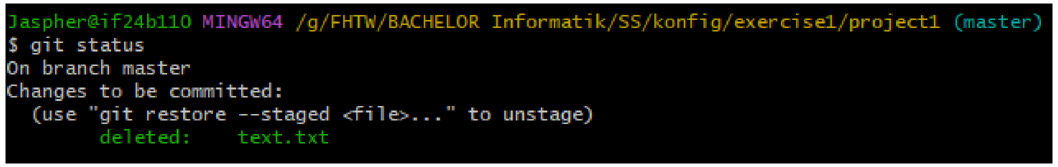
COMMAND

$git rm [filename]

- Removes file



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git rm text.txt
rm 'text.txt'
```

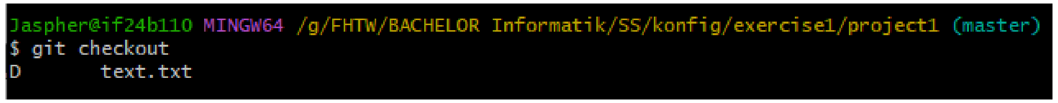


```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    text.txt
```

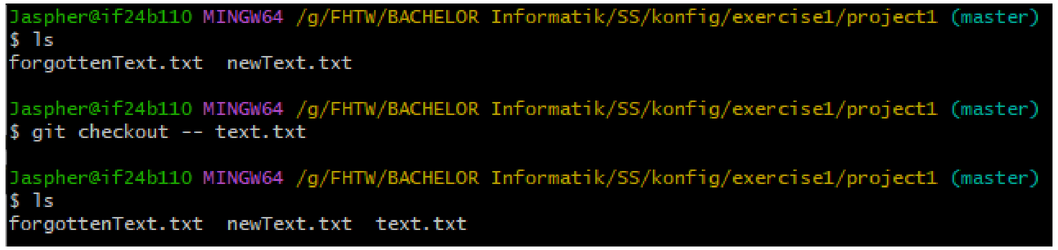
COMMAND

$git checkout -- <file-name>

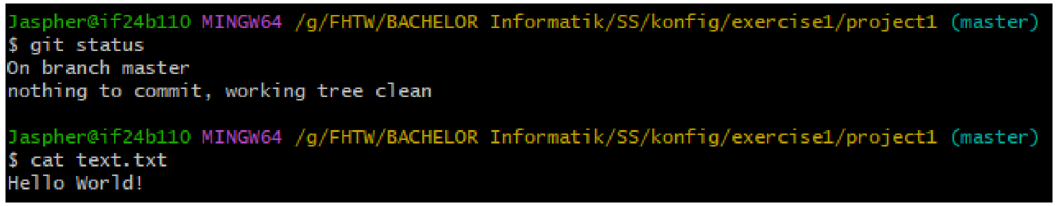
- Restores changes in working directory



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git checkout
D       text.txt
```



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ ls
forgottenText.txt  newText.txt

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git checkout -- text.txt

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ ls
forgottenText.txt  newText.txt  text.txt
```



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git status
On branch master
nothing to commit, working tree clean

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ cat text.txt
Hello World!
```

Modify files

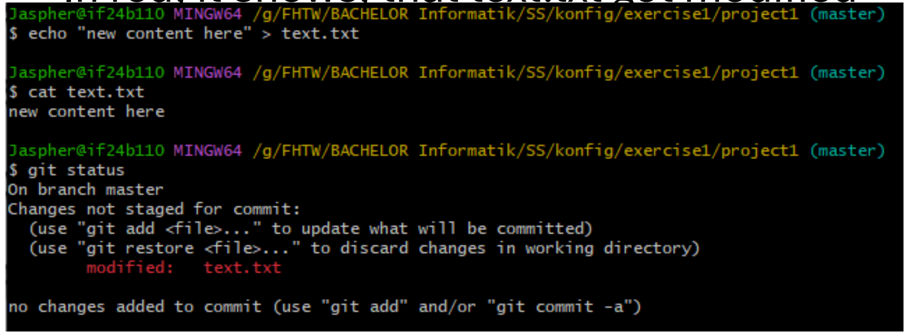
Modify the contents of the first file. Display the status in Git.

- What does the output look like and how do you interpret it?
- How do you commit the changes?
- What happens in the repository?

Modify files

Content of text.txt modified

- ***$ git status***
- In red, it shows, that text.txt got modified



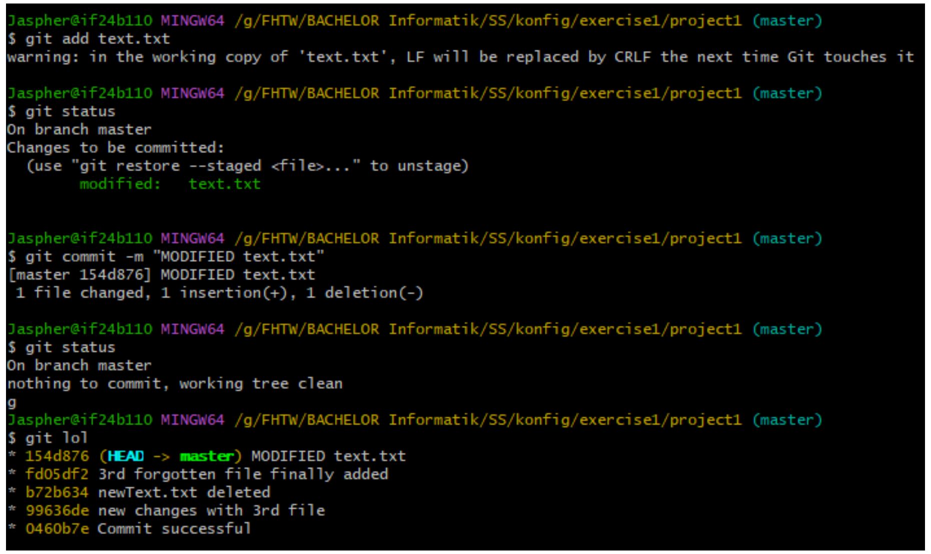
```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ echo "new content here" > text.txt

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ cat text.txt
new content here

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ 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:   text.txt

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

What happens to repository?

- Changes are saved in commit
- Commit is recorded in repository's history
- File's state is now part of the repository's commit history

How to commit the changes?

- After modifying content of file, add text.txt to repository again
- Commit the changes



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git add text.txt
warning: in the working copy of 'text.txt', LF will be replaced by CRLF the next time Git touches it

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   text.txt


Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git commit -m "MODIFIED text.txt"
[master 154d876] MODIFIED text.txt
 1 file changed, 1 insertion(+), 1 deletion(-)

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git status
On branch master
nothing to commit, working tree clean
g
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git lol
* 154d876 (HEAD -> master) MODIFIED text.txt
* fd05df2 3rd forgotten file finally added
* b72b634 newText.txt deleted
* 99636de new changes with 3rd file
* 0460b7e Commit successful
```

Checkout

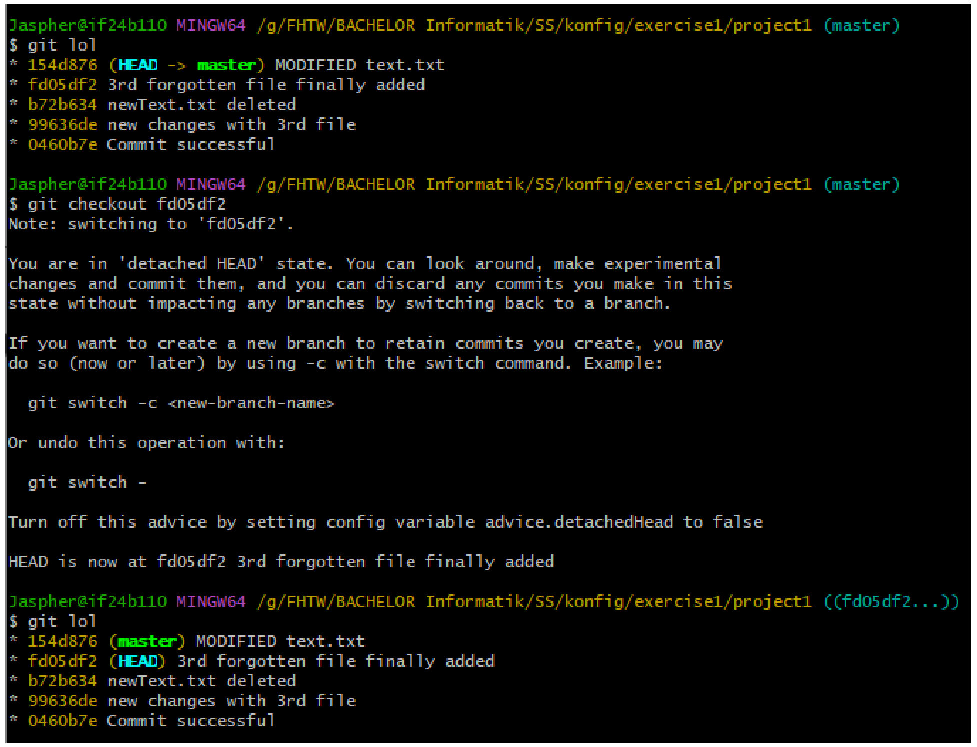
Look at the history andtry to go back to the penultimate version.

- Now look at the history again.
- What does HEAD mean?
- What is main/master?
- What other options does the command you just used offer?

Checkout

COMMAND

$ git checkout <SHA1-Key>



```
Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git lol
* 154d876 (HEAD -> master) MODIFIED text.txt
* fd05df2 3rd forgotten file finally added
* b72b634 newText.txt deleted
* 99636de new changes with 3rd file
* 0460b7e Commit successful

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 (master)
$ git checkout fd05df2
Note: switching to 'fd05df2'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at fd05df2 3rd forgotten file finally added

Jaspher@if24b110 MINGW64 /g/FHTW/BACHELOR Informatik/SS/konfig/exercise1/project1 ((fd05df2...))
$ git lol
* 154d876 (master) MODIFIED text.txt
* fd05df2 (HEAD) 3rd forgotten file finally added
* b72b634 newText.txt deleted
* 99636de new changes with 3rd file
* 0460b7e Commit successful
```

What happened?

- The HEAD just moved from the top commit to the one below.
- Next to path-directory, (master) changed to (SHA1 - key) of HEAD current position

What does HEAD mean?

It refers the current commit or current branch the user is working on.

What is main/master?

It is the default branch in a repository