

# Labor-Protokoll

## SEW

Name	<b>Karun Sandhu</b>
4-stellige Login-Nummer	<b>1195</b>
Klasse	<b>4CN</b>
Datum der Übung	16.09.2024
Datum der Abgabe	27.09.2024
Übungsnummer	00
Auftraggeber	ZAI
Thema der Übung	<b>git</b>

## Inhaltsverzeichnis

1 Git-Katas.....	2
1.1 Basic Commits.....	2
1.1.1 Result.....	3
1.2 Basic Staging.....	4
1.2.1 Result.....	6
1.3 Basic branching.....	7
1.3.1 Results.....	8
1.4 Fast-Forward merge.....	9
1.4.1 Results.....	10
2 Git Branching.....	11

# 1 Git-Katas

## 1.1 Basic Commits

1. Use git status to see which branch you are on.

```
exercise on ȷ master
fsh > git s
```

2. What does git log look like?

```
exercise on ȷ master
fsh > git l
fatal: your current branch 'master' does not have any commits yet
```

3. Create a file

```
exercise on ȷ master
fsh > touch nix
```

4. What does the output from git status look like now?

```
exercise on ȷ master [?]
fsh > git s
?? nix
```

5. add the file to the staging area

```
exercise on ȷ master [?]
fsh > git a nix
```

6. How does git status look now?

```
exercise on ȷ master [+]
fsh > git s
A nix
```

7. commit the file to the repository

```
exercise on ȷ master [+]
fsh > git cm "add nix"
[master (root-commit) 48a3bfe] add nix
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 nix
```

8. How does git status look now?

```
exercise on ȷ master took 5s
fsh > git s
```

9. Change the content of the file you created earlier

```
exercise on ȷ master
fsh > echo "NixOS is great!" > nix
```

10. What does git status look like now?

```
exercise on ȷ master [!]
fsh > git s
M nix
```

11. add the file change

```
exercise on ȷ master [!]
fsh > git a nix
```

12. What does git status look like now?

```
exercise on ȷ master [+]
fsh > git s
M nix
```

13. Change the file again

```
exercise on ȷ master [+]
fsh > echo "NixOS is the best!" > nix
```

14. Make a commit

```
exercise on ʘ master [!+]
fsh > git cm "nix: add content"
[master f459968] nix: add content
1 file changed, 1 insertion(+)
```

15. What does the status look like now? The log?

```
exercise on ʘ master [!]
fsh > git s
M nix

exercise on ʘ master [!]
fsh > git l --oneline
f459968 (HEAD → master) nix: add content
48a3bfe add nix
```

16. Add and commit the newest change

```
exercise on ʘ master [!]
fsh > git a nix

exercise on ʘ master [+]
fsh > git cm "nix: NixOS is the best not only great :P"
[master e2af14d] nix: NixOS is the best not only great :P
1 file changed, 1 insertion(+), 1 deletion(-)
```

### 1.1.1 Result

```
exercise on ʘ master
fsh > git s

exercise on ʘ master
fsh > git graph
* e2af14d (HEAD → master) nix: NixOS is the best not only great :P
* f459968 nix: add content
* 48a3bfe add nix
```

## 1.2 Basic Staging

1. What's the content of file.txt?

```
exercise on ʘ master
fsh > cat file.txt
```

	File: file.txt
1	1

2. Overwrite the content in file.txt: echo 2 > file.txt to change the state of your file in the working directory (or sc file.txt '2' in PowerShell)

```
exercise on ʘ master
fsh > echo 2 > file.txt
```

3. What does git diff tell you? That I edited 1 to 2.

```
exercise on ʘ master [!]
fsh > git d

file.txt

1:
1
2
```

4. What does git diff --staged tell you? why is this blank? Because we didn't stage the file yet.

```
exercise on ʘ master [!]
fsh > git ds
```

5. Run git add file.txt to stage your changes from the working directory.

```
exercise on ʘ master [!]
fsh > git a file.txt
```

6. What does git diff tell you? Nothing because all the changes are staged.

```
exercise on ʘ master [+]
fsh > git d
```

7. What does git diff --staged tell you? It shows the changes from before.

```
exercise on ʘ master [+]
fsh > git ds

file.txt

1:
1
2
```

8. Overwrite the content in file.txt: echo 3 > file.txt to change the state of your file in the working directory (or sc file.txt '3' in PowerShell).

```
exercise on ʘ master [+]
fsh > echo 3 > file.txt
```

9. What does git diff tell you? That I changed the staged state.

```

exercise on ʘ master [!+]
fsh > git d

file.txt
-----
1:
2
3

```

10. What does git diff --staged tell you?

```

exercise on ʘ master [!+]
fsh > git ds

file.txt
-----
1:
1
2

```

11. Explain what is happening

Git diff compares the your directory to the staging area or to the index. Git diff staged compares the changes from the staging area to the index.

12. Run git status and observe that file.txt are present twice in the output.

```

exercise on ʘ master [!+]
fsh > git s
MM file.txt

```

13. Run git restore --staged file.txt to unstage the change

```

exercise on ʘ master [!+]
fsh > git restore --staged file.txt

```

14. What does git status tell you now?

```

exercise on ʘ master [!]
fsh > git s
M file.txt

```

15. Stage the change and make a commit

```

exercise on ʘ master [+]
fsh > git cm "file.txt: change 1 to 3"
[master e9f1c9f] file.txt: change 1 to 3
1 file changed, 1 insertion(+), 1 deletion(-)

```

16. What does the log look like?

```

exercise on ʘ master
fsh > git graph
* e9f1c9f (HEAD -> master) file.txt: change 1 to 3
* 8cc5d2c 1

```

17. Overwrite the content in file.txt: echo 4 > file.txt (or sc file.txt '4' in PowerShell)

```

exercise on ʘ master [!]
fsh > echo 4 > file.txt

```

18. What is the content of file.txt?

```
exercise on ȳ master [!]  
fsh > cat file.txt
```

	File: file.txt
1 ~	4

19. What does git status tell us?

```
exercise on ȳ master [!]  
fsh > git s  
M file.txt
```

20. Run git restore file.txt

```
exercise on ȳ master [!]  
fsh > git restore file.txt
```

21. What is the content of file.txt?

```
exercise on ȳ master  
fsh > cat file.txt
```

	File: file.txt
1	3

22. What does git status tell us? Nothing because we are already on the newest change.

```
exercise on ȳ master  
fsh > git s
```

### 1.2.1 Result

```
exercise on ȳ master  
fsh > git s  
  
exercise on ȳ master  
fsh > git graph  
* e9f1c9f (HEAD → master) file.txt: change 1 to 3  
* 8cc5d2c 1
```

## 1.3 Basic branching

1. Use git branch to see the two branches that are relevant for this exercise

```
exercise on ʘ master
fsh > git b
* master
second-branch
```

2. What branch are you on? **master**
3. Use git branch mybranch to create a new branch called *mybranch*
4. Use git branch again to see the new branch created.

```
exercise on ʘ master
fsh > git b
* master
mybranch
second-branch
```

5. Use git switch mybranch to switch to your new branch.
6. How does the output from git status change when you switch between the *master* and the new branch that you have created? **It doesn't really except for the first line.**

```
exercise on ʘ master
fsh > git ss
On branch master
nothing to commit, working tree clean

exercise on ʘ master
fsh > git co mybranch
Switched to branch 'mybranch'

exercise on ʘ mybranch
fsh > git ss
On branch mybranch
nothing to commit, working tree clean
```

7. How does the workspace change when you change between the two branches? **It doesn't.**

```
exercise on ʘ master
fsh > cat dummy.txt
```

	File: dummy.txt
1	dummy

```
exercise on ʘ master
fsh > git co mybranch
Switched to branch 'mybranch'

exercise on ʘ mybranch
fsh > cat dummy.txt
```

	File: dummy.txt
1	dummy

8. Make sure you are on your *mybranch* branch before you continue.
9. Create a file called file1.txt with your name.
10. Add the file and commit with this change.
11. Use git log --oneline --graph to see your branch pointing to the new commit.

```
exercise on ʘ mybranch
fsh > git graph
* 4a85391 (HEAD → mybranch) add file1.txt
* c571de3 (second-branch, master) dummy commit
```

12. Switch back to the branch called *master*.
13. Use git log --oneline --graph and notice how the commit you made on the *mybranch* branch is missing on the *master* branch.

```
exercise on ʘ master
fsh > git log --oneline --graph
* c571de3 (HEAD → master, second-branch) dummy commit
```

14. Make a new file called file2.txt and commit that file.
15. Use git log --oneline --graph --all to see your branch pointing to the new commit, and that the two branches now have different commits on them.

```
exercise on ʘ master
fsh > git graph
* 9654e26 (HEAD → master) add file2.txt
| * 4a85391 (mybranch) add file1.txt
|/
* c571de3 (second-branch) dummy commit
```

16. Switch to your branch *mybranch*.
17. What happened to your working directory? Can you see your file2.txt? **No**

```
exercise on ʘ mybranch
fsh > ll
Permissions Size User Date Modified Git Name
.rw-r--r-- 6 karun 17 Sep 22:21 -- dummy.txt
.rw-r--r-- 0 karun 17 Sep 22:40 -- file1.txt
```

18. Use git diff mybranch master to see the difference between the two branches.

```
exercise on ʘ mybranch
fsh > git diff mybranch master

renamed: file1.txt → file2.txt
```

### 1.3.1 Results

```
exercise on ʘ mybranch
fsh > git s

exercise on ʘ mybranch
fsh > git graph
* 9654e26 (master) add file2.txt
| * 4a85391 (HEAD → mybranch) add file1.txt
|/
* c571de3 (second-branch) dummy commit
```



## 1.4 Fast-Forward merge

1. Create a (feature)branch called feature/uppercase (yes, feature/uppercase is a perfectly legal branch name, and a common convention).
2. Switch to this branch
3. What is the output of `git status`?

```
exercise on ȳ feature/uppercase
fsh > git s
```

4. Edit the `greeting.txt` to contain an uppercase greeting
5. Add `greeting.txt` files to staging area and commit
6. What is the output of `git branch`?

```
exercise on ȳ feature/uppercase
fsh > git b
* feature/uppercase
master
```

7. What is the output of `git log --oneline --graph -all`

```
exercise on ȳ feature/uppercase
fsh > git graph
* 8a9e7e3 (HEAD → feature/uppercase) greeting.txt: make uppercase
* eba84da (master) Add content to greeting.txt
* 8d46765 Add file greeting.txt
```

Remember: You want to update the master branch so it also has all the changes currently on the feature branch. The command '`git merge [branch name]`' takes one branch as argument from which it takes changes. The branch pointed to by HEAD (currently checked out branch) is then updated to also include these changes.

8. Switch to the master branch
9. Use `cat` to see the contents of the greetings

```
exercise on ȳ master
fsh > cat greeting.txt
```

	File: greeting.txt
1	hello

10. Diff the branches

```
exercise on ȳ master
fsh > git diff master feature/uppercase
```

```
greeting.txt
---
1:
hello
HELLO
```

11. Merge the branches

```

exercise on ȳ master
fsh > git merge feature/uppercase
Updating eba84da..8a9e7e3
Fast-forward
 greeting.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

```

12. Use `cat` to see the contents of the greetings

```

exercise on ȳ master
fsh > cat greeting.txt

```

	File: greeting.txt
1	HELLO

13. Delete the uppercase branch

```

exercise on ȳ master
fsh > git b -d feature/uppercase
Deleted branch feature/uppercase (was 8a9e7e3).

```

### 1.4.1 Results

```

exercise on ȳ master
fsh > git s

exercise on ȳ master
fsh > git graph
* 8a9e7e3 (HEAD → master) greeting.txt: make uppercase
* eba84da Add content to greeting.txt
* 8d46765 Add file greeting.txt

```

## 2 Git Branching

MainRemote

### Introduction Sequence

A nicely paced introduction to the majority of git commands

3: *Merging in Git*

### Ramping Up

The next serving of 100% git awesomes-ness. Hope you're hungry

1: *Detach yo' HEAD*

### Moving Work Around

"Git" comfortable with modifying the source tree :P

1: *Cherry-pick Intro*

1

2

### A Mixed Bag

A mixed bag of Git techniques, tricks, and tips

1: *Grabbing Just 1 Commit*

1

2

3

4

5

### Advanced Topics

For the truly brave!

1: *Rebasing over 9000 times*

1

2

3

MainRemote

### Push & Pull -- Git Remotes!

Time to share your 1's and 0's kids; coding just got social

1: *Clone Intro*

7

8

### To Origin And Beyond -- Advanced Git Remotes!

And you thought being a benevolent dictator would be fun...

1: *Push Main!*

1

2

3

4

5

6

7

8