

KONF - Exercise Sheet 2

by Niklas Brandtner 6.03.2024

1. Branching

Working in the master/main branch is considered a faux pas in the community. For this reason, create a development branch and check it out. Which 2 command sequences can you use for it? What options do these commands offer?

- ```
git branch development
```

 # Create a development branch  

```
git checkout development
```

 # Switch to the development branch
- You could also use `git checkout -b development` to create and check out the new branch

## 2. Documentation

Note in your presentation what specifically happened. What does the history look like now? What happened with HEAD?

- ```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ git log-all
* 625d5ca (HEAD -> development, main) Update test.txt again
* 0acce1b Updated files
* b6ff320 Add another_test.txt
* 1406498 (origin/main) Add test.txt
```
- `HEAD` now points to the "development" branch, indicating that this is the currently checked-out branch.

3. Changes

In the first exercise sheet, you have already added a few files. Now change a file in the Development Branch and add these changes to the Staging Area. What does the status look like now? Has the history changed?

- ```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ nano test.txt

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ git add test.txt

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ git status
On branch development
Changes to be committed:
 (use "git restore --staged <file>..." to unstage)
 modified: test.txt
```
- Status tells us `On branch development`
- ```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ git log-all
* 625d5ca (HEAD -> development, main) Update test.txt again
* 0acce1b Updated files
* b6ff320 Add another_test.txt
* 1406498 (origin/main) Add test.txt
```
- History hasn't changed

4. Diff

Run the `git diff` command - what does the output look like and how do you interpret it?

```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ git diff .
diff --git a/test.txt b/test.txt
index 13069c5..a4a405e 100644
--- a/test.txt
+++ b/test.txt
@@ -1,4 +1,4 @@
  Hello world!
  again!
  and again!
  -damn it doesnt stop!
  +damn it won't stop!
```

- here we see a possible output by `git diff` we can see what files the command has found changes in. It also shows the exact changes.

5. Commit

Commit your changes. What does the history look like now and how do you interpret the result?

```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ git log-all
* 3f8dd6d (HEAD -> development) 2.5 Commit Changes:
* 625d5ca (main) Update test.txt again
* 0acce1b Updated files
* b6ff320 Add another_test.txt
* 1406498 (origin/main) Add test.txt
```

- the new commit gets added to the log...

6. Accidentally Committed?

If you commit once by mistake there is a way to undo the commit. Describe the command. The history should look like before the commit! Which options does the command offer (especially `--soft` and `--hard`)?

```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ git reset --soft HEAD~1

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ git log-all
* 625d5ca (HEAD -> development, main) Update test.txt again
* 0acce1b Updated files
* b6ff320 Add another_test.txt
* 1406498 (origin/main) Add test.txt
```

- `git reset` moves `HEAD` pointer back by one commit
- `git reset --soft HEAD~1` undoes the last commit but leaves modified files in staging area
- `git reset --hard HEAD~1` undoes the last commit, unstages the changes, and reverts working directory back to previous commit

7. Changes 2

Now make changes in the development branch again and commit them.

- Since I've used `--soft` in my example i dont need to do changes again because the previous changes are back to the staging area:

```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ git commit -m "2.5 Commit Changes (again):" .
[development d6a8fd6] 2.5 Commit Changes (again):
 1 file changed, 1 insertion(+)

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ git log-all
* d6a8fd6 (HEAD -> development) 2.5 Commit Changes (again):
* 625d5ca (main) Update test.txt again
* 0acce1b Updated files
* b6ff320 Add another_test.txt
* 1406498 (origin/main) Add test.txt
```

8. .gitignore

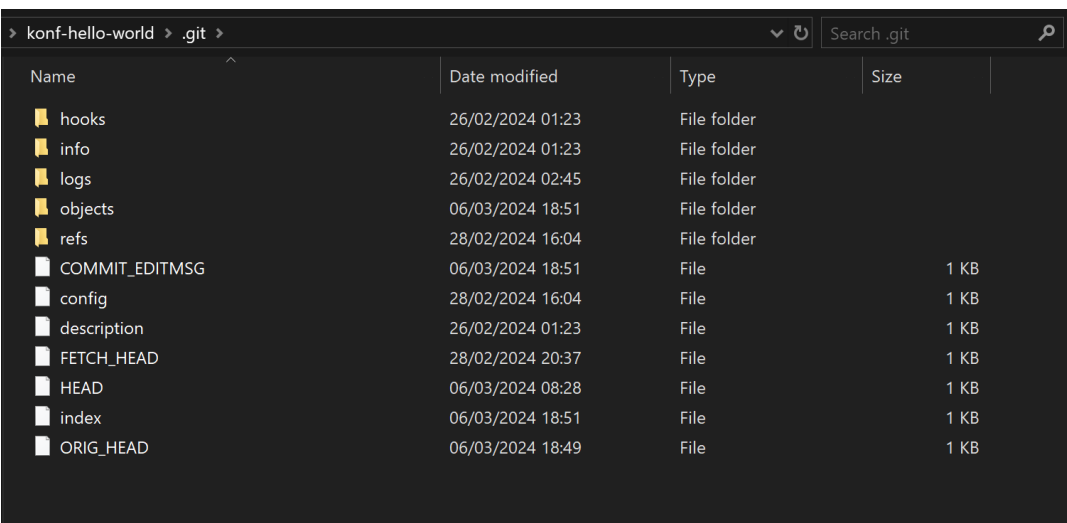
Add a `.gitignore` file to the root directory of your workspace and populate it so that files with the extension `.o` are not included in the repository. (Note: `.o` files, i.e. object files, are created by the compiler and can be created again at any time and are therefore not necessarily

subject to version control) Now create an .o file and try to add it to the repository.

- ```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ touch .gitignore
```
- ```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ echo "*.o" > .gitignore
```
- ```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ touch test.o
```
- ```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ ls
another_test.txt  forgotten_test.txt  test.o  test.txt
```
- ```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (development)
$ git add test.o
The following paths are ignored by one of your .gitignore files:
test.o
hint: Use -f if you really want to add them.
hint: Turn this message off by running
hint: "git config advice.addIgnoredFile false"
```

## 9. .git directory

Venture into the .git directory and browse a bit. What do you notice?



| Name           | Date modified    | Type        | Size |
|----------------|------------------|-------------|------|
| hooks          | 26/02/2024 01:23 | File folder |      |
| info           | 26/02/2024 01:23 | File folder |      |
| logs           | 26/02/2024 02:45 | File folder |      |
| objects        | 06/03/2024 18:51 | File folder |      |
| refs           | 28/02/2024 16:04 | File folder |      |
| COMMIT_EDITMSG | 06/03/2024 18:51 | File        | 1 KB |
| config         | 28/02/2024 16:04 | File        | 1 KB |
| description    | 26/02/2024 01:23 | File        | 1 KB |
| FETCH_HEAD     | 28/02/2024 20:37 | File        | 1 KB |
| HEAD           | 06/03/2024 08:28 | File        | 1 KB |
| index          | 06/03/2024 18:51 | File        | 1 KB |
| ORIG_HEAD      | 06/03/2024 18:49 | File        | 1 KB |

- .gitignore is hidden by default
- HEAD contains a reference to the branch or commit that is currently being checked out
- config holds your configurations for this repo
- index stores staging area
- the ref folder stores references to the branches, remotes and tags of the repo
- objects stores all kinds of git objects

## 10. Feature Branches

Imagine Customer A calls and urgently needs a feature A. You get right to work and create a feature branch "featureA", develop a file FeatureA and populate it. Commit the changes.

What does the history look like now?

- ```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (featureA)
$ touch featureA.txt
```
- ```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (featureA)
$ echo "I am Feature A" > f
featureA.txt forgotten_test.txt
```
- ```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (featureA)
$ echo "I am Feature A" > featureA.txt
```

```
nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (featureA)
$ git add .
warning: LF will be replaced by CRLF in .gitignore.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in featureA.txt.
The file will have its original line endings in your working directory

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (featureA)
$ git commit -m "add FeatureA" .
warning: LF will be replaced by CRLF in .gitignore.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in featureA.txt.
The file will have its original line endings in your working directory
[featureA a14643d] add FeatureA
 2 files changed, 2 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 featureA.txt

nbran@DESKTOP-051PSJ2 MINGW64 ~/Documents/GitHub/konf-hello-world (featureA)
$ git log -all
* a14643d (HEAD -> featureA) add FeatureA
* d6a8fd6 (development) 2.5 Commit Changes (again):
* 625d5ca (main) Update test.txt again
* 0acce1b Updated files
* b6ff320 Add another_test.txt
* 1406498 (origin/main) Add test.txt
```

- We can see the difference in branches and the new commit

11. Hotfix

Now customer B calls, complaining about an error. He cannot continue working. However, you are not done with FeatureA yet. Is this a problem? How do you go about delivering a hotfix quickly? How does this affect your developments in FeatureA? Do you deliver a hotfix to customer B? Remember that only master/main commits are delivered, or commits in their own deploy branches. Document your command sequence and history.

- If FeatureA is not as urgent as the hotfix then I'd stash the changes made to Feature A and begin working on the Hotfix for Customer B on their Branch.
- `git stash` to stash away unfinished changes without committing them
- `git checkout featureB`
- `<develop hotfix>`
- `git add .`
- `git commit -m "deliver hotfix customer b" .`
- `git checkout main`
- `git merge featureB`
- `git checkout featureA`
- `git stash apply` to get changes back to Working Dir

12. Finish Feature A

We now simulate that you complete featureA, i.e. you modify the featureA file again and check it in. Now merge the hotfix from 11 and featureA in such a way that you can deliver featureA including hotfix to customer B. Document the command sequence.

- After everything in step 11 you only have to finish developing featureA and then use:
- `git checkout main`
- `git merge featureA` to merge the featureA branch with the main branch

13. Branch deletion

After completion and merging with the master branch, delete the featureA branch. What is the command to do this? What options does it provide?

- `git branch -d featureA` or `git branch -D featureA`
- -d for merged branches
- -D for unmerged branches

