Exercise 3

Goals

- Checkout files at some point in the Git history
- Merge two branches (without merge conflicts)
- Delete unused branches

Structure

In this exercise we will continue working on flyers (flyer_A and flyer_B) for two parties (Party A and Party B). First we need to revert the flyers to a specific commit. Then we modify the same file, but in different branches. Finally we merge all our changes into branch master.

Again the exercise consists of short descriptions about a specific git command (less detailed than in previous exercises), followed by a hands-on part for you to execute appropiate git commands.

In order to allow a smooth exercise, there are some functions written by C2SM in the file *helpers.sh* that are **NOT** part of Git. For this exercise we us the following functions from that file:

- **init_exercise:** It will create the *work* directory and navigate into it
- **reset:** It will delete the *work* folder and enable you a clean restart of the exercise in case you completely mess it up
- **init_broken_repo:** setup a Git repository as in exercise 2, but with commits, that we need to revert first.

Reminder: *all text enclosed with* `<>` *denotes a placeholder to be replaced by a specific string appropriate in your context.*

Initialization

In [ ]:
```
# check current directory with "pwd"
pwd
# go to folder of this exercise using "cd"
```

In [ ]:
```
# execute this code at the very beginning to initialize the exercise proper
source ../helpers.sh
init_exercise
```

Optional: clear notebook and restart

In case you mess up your notebook completely,

execute *reset* in the following cell. This will restore a clean environment!

In [ ]:
```
## only execute in case of (serious) trouble ##
## it will delete your entire work directory ##
reset
```

Exercise

Checkout files at some point in the Git history

In [ ]:
```
# this line will setup a simple Git repository for you
init_broken_repo
```

To see the flyers text, follow the instructions below:

- Go to folder *work* and enter *party_planning*
- Open *flyer_A* and *flyer_B*

As you can see, the music at both of our parties is *Classical Music*.

Initially we planned to play *Metal Music* on both of our parties (see output above), but had to cancel it because of our neighbors.

Our neighbors were loud too last night, so we decide to play *Metal Music*. To not do the work twice, we want to reuse the flyers we designed in the first place.

We can use `git checkout` to get any version of a file along its Git history.

Simply execute `git checkout <specific_commit_hash> <your_flyer>`.

Let's try it out for flyer_A first!

In [ ]:
```
# checkout the version of flyer_A at commit: Metal Music added
# "git checkout <commit_hash> <file_to_checkout>"
```

Refresh the jupyter notebook page showing flyer_A and have a look at it. You see, we now play Heavy Metal again.

Do the same for flyer_B as well.

In [ ]:
```
# checkout flyer_B at commit Metal Music added
```

In [ ]:
```
# check if git tracked our changes

# commit our updated flyers (the git add was done automatically as part of
```

In [ ]:
```
# git log to see the Git-history
```

Every *commit hash* is unique, so you can checkout files back in time **and** across different

branches.

All of a sudden we decide to not play music at party B. Therefore checkout flyer_B at commit "add happy-hour"

In [ ]:
```
# checkout flyer_B at right commit

# commit flyer_B
```

In [ ]:
```
# execute git log --oneline for short summary
```

Your output should look similar to:

```
6f0670f (HEAD -> master) remove music from flyer_B
0309b26 revert both flyers back to Metal Music
db7f2a6 Classical Music added
a37f6b4 Metal Music added
7a221e6 add happy-hour
762d054 add opening time
d7555f7 add flyer_B
ff3e7bb add flyer_A
```

Merge two branches (without merge conflicts)

In this part of the excercise we continue writing on our *flyer_A*. There are two sections in flyer_A to modify in a seperate branch:

- dresscode
- VIP-guests

In [ ]:
```
# checkout a new branch for the dresscode
```

To edit the flyers text, follow the instructions below:

- Go to folder *work* and enter *party_planning*
- Open *flyer_A*
- Edit text in the `dresscode` section

Do not forget to save flyer_A before coming back here

In [ ]:
```
# make commit
```

In [ ]:
```
# go back to branch master using git checkout
```

In [ ]:
```
# checkout a new branch for the VIP-guests
```

To edit the flyers, follow the instructions above!

In [ ]:
```
# make commit
```

In [ ]:
```
# go back to branch master
```

`git branch` should output something like that:

```
  VIP
  dresscode
* master
```

Let's put the pieces of the flyer together. For that we use the *git merge* functionality.
It allows us to merge files with different text from different branches.

To merge all modification from branch *VIP* into branch *master* we type:

`git merge VIP`

In [ ]:
```
# merge VIP into master
```

Git just performed a so called *Fast-forward merge*. This means, that there is a linear path between the two merged branches. See the slides for more detailed information about it.

Most important:
Git does **NOT** create an additional commit to perform the merge. It only appends the commit from the branch *VIP* to the HEAD.

In [ ]:
```
# git log to see the added commit
```

In [ ]:
```
# display content of flyer_A using "cat"
```

As you see, we succesfully took over our changes from branch VIP

Let's do the same, but for the modifications in branch *dresscode*

In [ ]:
```
# merge dresscode into master
```

For this merge Git performs a so-called *3-way merge*, because the path between the two branches is not linear anymore due to the merge of branch *VIP*.
Therefore Git creates a *merge-commit* to bring the two histories together.

In [ ]:
```
# display content of flyer_A using "cat"
```

In [ ]:
```
# git log to see the added commit
```

Your git log looks the following:

```
82414b3 (HEAD -> master) merge dresscode
aa563f9 (dresscode) add dresscode
```

```
8d56ebc (VIP) add VIP
a74572e Classical Music added
56fab47 Metal Music added
f6ecd48 add happy-hour
51448ea add opening time
a39a076 add flyer_B
a0ffdf8 add flyer_A
```

Delete unused branches

After merging it is good practice to delete merged branches.

The command `git branch -d <branch_to_delete>` can do this.

In [ ]:
```
# delete branch VIP
```

In [ ]:
```
# delete branch dresscode
```

Congrats, your Git skills are getting better and better!