Advanced Git Skills

All the commands can either be executed on the command line or in the VS Code GUI.

The GUI is more user friendly and shows some nice overviews, but it is dependent on the program you use and when you switch IDE, you might have to learn it again. Note that the IDE executes the git commands in the background.

On the other hand, using git directly on the command line comes with very detailed control (with many parameters) and is stable over all environments (no matter the IDE and on servers or high-performance computing clusters).

Task	**Command Line**	**GUI (VS Code)**
1 Fork repository	Fork the repository on https://github.com/grundkurs-git/exercise-3 by clicking the “Fork” button in the top right corner	
2 Launch Codespace	- Click on the green "Code" button and from the dropdown, - Select "Codespaces" and click on “Create codespace on main”. This initiates a cloud-based development environment directly within GitHub, tailored for the repository.	
3 Run project	Open a terminal and run the project with `python3 main.py`	Click on “main.py”. Install any necessary extensions for Python. Click on the triangle in the top right to run the project.
4 Create new branch	Enter `git branch <lastname>-branch`	In the toolbar on the left switch to “Source Control”. Click on the three dots. Select “Branch”, “Create Branch”. Enter “<lastname>-branch” and click enter.

	to create a new branch. This command creates a new branch, but it does not change the current branch. To verify this: `git status` should display “`On branch main`”.	
5 Checkout new branch	Let's change to the branch with `git checkout <lastname>-branch`	The branch should be checked out automatically. You can see the name of the current branch in the bottom left corner next to the “Source Control” symbol.
6 Make changes	Open the main.py file and go to line 115. Enter the following line to complete the program: `result = '*** Congrats ! You won ! ***'` Save the modifications and run the program. It should work as expected now.	
7 Check Git Status	Enter `git status` We see an overview of the files in the git system and what we can do with them.	In the “Source Control” tab you can see the changes.
8 Create .credentials file	Create a file “.credentials” and add your top-secret password there. To protect our password, we don’t want to upload it to the remote repository. To exclude them, we can add the filename to the file “.gitignore” (the . at the beginning of a file means, that it is hidden).	
9 Add to .gitignore	Run the command `echo '.credentials' > .gitignore` to exclude the “.credentials” file. When you run	In the “Source Control” tab right-click on “.credentials” and click “Add to .gitignore”.

	`git status` You shouldn't see the file anymore. We want to exclude files from version control like editor config files or files containing passwords.	
10 Add changes to staging area	To add all files to the staging area, use the command `git add .` Alternatively, you can also add single files with `git add <path-to-file>`.	Highlight all files in the "Changes" section and right-click on it. Click on "Stage Changes".

11 Commit your changes	Re-enter
	`git status`
	The main.py file should be now in the `Changes to be committed` section. To commit the modifications, we use
	`git commit -m "<commit message>"`
	Then an editor opens. It is best practice to give a short description of the changes you have made in this commit.
		Click on the checkmark. Enter your commit message. It is best practice to give a short description of the changes you have made in this commit.

12 Switch back to main branch	Let's change back to the main branch by entering `git checkout main` If we scroll down to line 115 in main.py, we see that our inserted line is gone!	Click on the branch in the bottom left corner and then click on "main".

13 Create second branch	Let's create another branch. Make sure that you are on the main branch (verify it with `git status`) and then as before
	`git branch my_cool_new_feature`
	and
	`git checkout my_cool_new_feature`
	If you re-enter `git status` you should be on branch my_cool_new_feature.
		Like before in the tab “Source Control”. Click on the three dots. Select “Branch”, “Create Branch”. Enter “my_cool_new_feature” and click enter.

14 Make changes	Add this to line 115 (slightly different than in step 6!):
	`result = 'Congrats ! You won !'`
	And change line 118 to:
	`result = 'You lose !'`	

15 Add and commit changes	Save the file and add the main.py file to the staging area with `git add main.py` Commit the changes with git commit as before.	Like before, stage the file main.py by highlighting it, right-clicking it and clicking “Stage Changes”. Commit the changes like before.

16 Merge branch my_cool_new_feature in <lastname>-branch	Run
	`git checkout <lastname>-branch`
	to switch to the other branch. Use
	`git merge my_cool_new_feature`
	to merge your branch my_cool_new_feature into your branch <lastname>-branch.
		Switch to the “<lastname>-branch” branch like before.
		
		Then click on the three dots in the “Source Control” tab navigate to “Branch” and click on “Merge Branch…”. Select the “my_cool_new_feature” branch.

<table>
<tr><td>17 Resolve conflicts</td><td colspan="2">The idea of a merge is to combine the modifications of both branches. Git tries to do that, but sometimes it fails. This is called merge conflict. In our case, we modified in both branches the same line (line 115), therefore Git does not know which modification we want to keep. If a merge conflict happens, we must resolve the conflicts by hand and afterward, we can make a new commit (a so-called merge commit).
Go to line 115. Merge conflicts can be easily identified:
<pre><<<<<<< branch1
....
=======
...
>>>>>>> branch2</pre>Let’s pick one of the two options. Often you must combine them and pick some lines from the upper branch and some from the lower ones. Save the modifications and test the program by re-running it.
Notice that only one modification caused a merge conflict (line 115). The other modification (line 118) could be merged by Git automatically.</td></tr>
<tr><td>18 Commit modifications</td><td>Use
<pre>git add main.py</pre>and
<pre>git commit -m “<commit message>”</pre>to commit the resolved conflicts in steps 10 and 11.</td><td>Stage the changed main.py file and commit. Do not forget a suitable commit message (like “resolved merge conflicts”).</td></tr>
<tr><td>19 Look at your git history so far</td><td>To see what you have done so far, use
<pre>git log</pre></td><td>In the Explorer tab on the left side, open tab “Timeline” to look at the git history.</td></tr>
<tr><td>20 Push to the remote repository</td><td>To upload our branch to the remote repository, we can use</td><td>Click on “Publish Branch” to push the branch to the remote repository.</td></tr>
</table>

	`git push origin -u <lastname>-branch`
23 Have fun and get some beers ;)	Congratulations! You made it until the end of this tutorial.
24 Expert Skills	- Hands-on tutorial for learning expert skills such as cherry picking: https://learngitbranching.js.org/ - Learn how to integrate large data files into git repositories: https://dvc.org/, https://git-lfs.github.com/