SA4 - Using Git with VSCode and Github, Basic Merging

Activity 1: Lecture Comprehension Check

Based on the provided notes, answer the following short-answer comprehension questions:

1. **How can you globally change the default initial branch name for new repositories to `main` in Git?**
2. **What command is used to rename an existing `master` branch to `main` in a local repository?**
3. **Describe the significance of the color-coded icons next to files in VSCode's Explorer panel.**
4. **What does a grey icon or color next to a file or directory in VSCode indicate?**
5. **Explain the difference between a fast-forward merge and a three-way merge in Git.**
6. **What steps should be taken to resolve a merge conflict in Git?**
7. **Why is it important to configure the `.gitignore` file, and how does VSCode represent ignored files?**
8. **How do local and remote repository structures in Git compare, and why is this important for collaboration?**
9. **What steps are involved in linking a local repository to a GitHub repository and pushing changes?**

What to submit:

- docx or pdf file. Please include both the text of the question and your answer.

Rubric:

- H: All questions answered thoroughly, correctly, and in your own words.
- L: All questions answered, but some incorrect or incomplete answers.
- F: No submission, unanswered questions, suspected plagiarism, or unsanctioned use of generative-AI

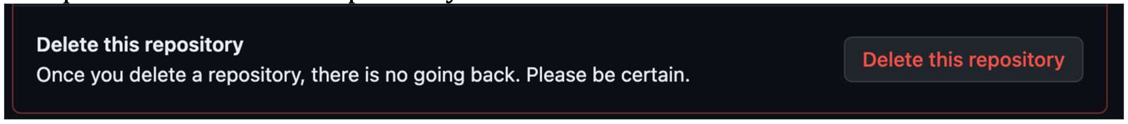
Activity 2: Using Git and Github for a Python "Palindrome Checker" Project (Guided)

To begin this activity, first, make sure your current working directory is `workspace-3081`. Then, create a folder inside of `workspace-3081` called `sa4-act2` to contain the files for this activity. `cd` into the `sa4-act2` folder, and then run `code .` to open the folder in VSCode.

Also, run the following to update your `.gitconfig` file to use `main` as the initial branch:

```
git config --global init.defaultBranch main
```

If you need to restart the activity, you can go back to the `workspace-3081` folder and `rm -rf sa4-act2`. You won't need to update your `./gitconfig` again. To delete your GitHub repo, go to the GitHub page for the repo and click "Settings". At the bottom of the page, there is an option to delete the repository.


Delete this repository

Once you delete a repository, there is no going back. Please be certain.

1. Initialize a git repo in the `sa4-act2` folder.

```
git init
```

2. Create a Python file, `check_palindrome.py`, with the following starter code:

```
text = "did"

if text == text[::-1]:
   print(f"{text} is a palindrome")
else:
   print(f"{text} is not a palindrome")
```

3. Save the file (`Ctrl-S`) and run the code with the following command to confirm that it works:

```
python3 check_palindrome.py
```

4. Stage the changes and make an initial commit with the starter code on `main` with the following commands:

```
git add -A
git commit -m "Initial commit, added basic palindrome check"
```

5. Create a public GitHub repo called `sa4-act2` under your account using the steps described in the notes. The repo must be public.

6. Add the remote for your new GitHub repo to the local repository with the command:

```
git remote add origin git@github.com:username/sa4-act2.git
```

Important! Make sure to replace `username` with your GitHub username.

7. Push your local `main` branch to to the `origin/main` branch and set up automatic tracking with the command:

```
git push -u origin main
```

8. On the `main` branch, refactor the starter code to use a function by changing the code to the following:

```
def check_palindrome(text):
   if text == text[::-1]:
      print(f"{text} is a palindrome")
   else:
      print(f"{text} is not a palindrome")

text = "did"
check_palindrome(text)
```

Save the file (`Ctrl-S`) and run the code with the earlier command to confirm that it works.

9. Stage the changes and make another commit on `main` with the following commands:

```
git add -A
git commit -m "Refactored palindrome check to use a function"
```

10. Push the `main` branch changes to GitHub with the following command:

```
git push
```

11. Create and switch to a local feature branch, `user-input` with the following command:

```
git switch -c user-input
```

12. On the `user-input` branch, modify the implementation to take a user input string to check by changing the code to the following:

```
def check_palindrome(text):
   if text == text[::-1]:
      print(f"{text} is a palindrome")
   else:
      print(f"{text} is not a palindrome")

text = input("Enter text to check for palindrome: ")
check_palindrome(text)
```

Save the file (`Ctrl-S`) and run the code with the earlier command to confirm that it works.

13. Stage the changes and make a commit on `user-input` with the following commands:

```
git add -A
git commit -m "Replaced hardcoded string with user-input text string"
```

14. Push the local `user-input` branch to a remote branch `origin/user-input` with the command:

```
git push -u origin user-input
```

15. Switch back to the local `main` branch with the following command:

```
git switch main
```

16. Create and switch to a local feature branch, `file-input` with the following command:

```
git switch -c file-input
```

17. On the `file-input` branch, modify the implementation to take a path to a file as input and check if its contents are palindromic by changing the code to the following:

```
def check_palindrome(text):
   if text == text[::-1]:
      print(f"{text} is a palindrome")
   else:
      print(f"{text} is not a palindrome")

try:
   file = input("Enter filepath for palindrome check: ")
   text = open(file, "r").read()
   check_palindrome(text)
except:
   print(f"Unable to process file at {path}")
```

18. Create a text file `text.txt` in the current directory to use as input. Add text content to file.

19. Save both files (`Ctrl-S`) and run the Python code with the earlier command to confirm that it works.

20. Stage the changes and make a commit on `file-input` with the following commands:

```
git add -A
git commit -m "Replaced hardcoded text string with filepath input and
read file content"
```

21. Push the local `file-input` branch to a remote branch `origin/file-input` with the command:

```
git push -u origin file-input
```

22. Switch back to the local `main` branch with the following command:

```
git switch main
```

23. Merge the `user-input` branch into `main` with the following command:

```
git merge user-input
```

24. Merge the `file-input` branch into `main` with the following command:

```
git merge file-input
```

25. Resolve the merge conflicts by accepting both changes and adding an extra conditional logic to determine whether to use file or text input. The completed file should look like:

```
def check_palindrome(text):
   if text == text[::-1]:
      print(f"{text} is a palindrome")
   else:
      print(f"{text} is not a palindrome")


choice = input("Enter 't' for text or 'f' for file: ")
if choice == "t":
   text = input("Enter text to check for palindrome: ")
   check_palindrome(text)
elif choice == "f":
   try:
      file = input("Enter filepath for palindrome check: ")
      text = open(file, "r").read()
      check_palindrome(text)
   except:
      print(f"Unable to process file at {path}")
else:
   print("Invalid choice")
```

26. Stage the changes and make a merge commit on `main` with the following commands:

```
git add -A
git commit -m "Merged file-input into main, added choice for file vs
text"
```

27. Push the changes to `main` to GitHub with the following command:

```
git push
```

What to submit:

- a link to your GitHub repo that you used for this activity. (Repo must be public)
- a screenshot of the final `gloga` for the local repo

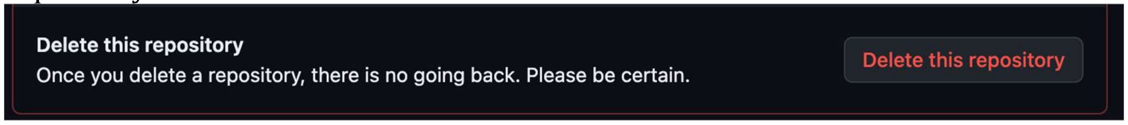
Rubric:

- H: All steps completed correctly, GitHub repo visible to graders and shows the correct branches and commit history
- L: Branching, merging, and GitHub link successful. Some mistakes with Python code.
- F: No submission, missing steps/features, repo not visible to graders

Activity 3: Using Git and Github for a Python "Guess the Number" Project

To begin this activity, first, make sure your current working directory is `workspace-3081`. Then, create a folder inside of `workspace-3081` called `sa4-act3` to contain the files for this activity. `cd` into the `sa4-act3` folder, and then run `code .` to open the folder in VSCode.

If you need to restart the activity, you can go back to the `workspace-3081` folder and `rm -rf sa4-act3` to delete your local repo. To delete your GitHub repo, go to the GitHub page for the repo and click "Settings". At the bottom of the page, there is an option to delete the repository.


Delete this repository
Once you delete a repository, there is no going back. Please be certain.

1. Initialize a git repo in the `sa4-act3` folder.

2. Create a Python file, `guess_number`, with the following starter code:

```
number = 10

print("I'm thinking of a number...")
guess = int(input("What number am I thinking of? "))

if guess == number:
   print("Congratulations! You guessed the right number.")
else:
   print(f"Sorry! The number was {number}.")
```

3. Run the code to confirm that it works and make an initial commit with the starter code on `main`. Use a descriptive commit message.

4. Create a public GitHub repo called sa4-act3 under your account and sync your local repo's `main` branch to it.

5. On the `main` branch, modify the starter code so that the program will allow the user to keep entering guesses until they guess the correct number or enter 'q' to quit. If

they enter 'q', the program should tell them what the number was. If they guess incorrectly, tell them to try again. Change the starter code as little as possible while still achieving the goal.

6. Run the code to confirm that it works and make a new commit on the `main` branch with a message that describes the change you made.
7. Push the `main` branch changes to GitHub.
8. Create and switch to a local feature branch, `guess-hints`, where you modify the implementation to tell the user if their guess was too high or too low if they guessed incorrectly.
9. Run the code to confirm that it works and make a new commit on the `guess-hints` branch with a message that describes the change you made.
10. Push the local `guess-hints` branch to a remote branch `origin/guess-hints`.
11. Switch back to the local `main` branch and don't merge yet.
12. Create and switch to a local feature branch, `limited-guesses`, where you modify the implementation to cap the total number of guesses the user can make before showing them the answer. For each wrong guess, you should tell them how many guesses they have left.
13. Run the code to confirm that it works and make a new commit on the `limited-guesses` branch with a message that describes the change you made.
14. Push the local `limited-guesses` branch to a remote branch `origin/limited-guesses`.
15. Switch back to the local `main` branch and merge the `limited-guesses` branch into `main`.
16. Merge the `guess-hints` branch into `main` and resolve any merge conflicts.
17. Push the changes to `main` to GitHub.

What to submit:

- a link to your GitHub repo that you used for this activity. (Repo must be public)
- a screenshot of the final `gloga` for the local repo

Rubric:

- H: All steps completed correctly, GitHub repo visible to graders and shows the correct branches and commit history
- L: Branching, merging, and GitHub link successful. Some mistakes with Python code.
- F: No submission, missing steps/features, repo not visible to graders

SA4 Submission and Grading

You will receive a numerical score for SA4 based on the following table:

Combined H/L/F	Score (%)
HHH	100
HHL	90
HLL	80
LLL	70
HHF or HLF or LLF or FFF	0

Scores will be calculated twice, after the first and second-chance submissions. After the second-chance grading is complete, the score will be final.