

Exercise: A Journey Through Git

Goal

Learn the fundamentals of Git by working on a Python project that computes prime numbers and generates Fibonacci sequences, while navigating through various Git commands.

Step 1: Initialize a Git Repository

1. Navigate to desired directory

Make sure to make it in a folder that is not synchronized ie. no OneDrive, Google Drive or similar. One safe spot is the user folder which can be navigated to in a **powershell** or a **linux terminal** simply with:

```
cd ~
```

2. Create a new directory

```
mkdir git_exercise  
cd git_exercise
```

3. Initialize a Git repository:

```
git init
```

4. Check the status of your repository:

```
git status
```

At this point, there should be nothing tracked.

Step 2: Write a Python Program to Compute Prime Numbers

1. Create a Python file named `prime_numbers.py` and write code that checks if a number is prime [COPY PASTE]:

```
def is_prime(num):  
    if num < 2:  
        return False  
    for i in range(2, int(num ** 0.5) + 1):
```

```
        if num % i == 0:
            return False
        return True

def get_primes(n):
    primes = []
    for i in range(2, n+1):
        if is_prime(i):
            primes.append(i)
    return primes

if __name__ == "__main__":
    n = int(input("Find primes up to: "))
    print(get_primes(n))
```

2. Check that the code runs as expected.

Step 3: Stage and Commit the Changes

1. Add the file to the staging area:

```
git add prime_numbers.py
```

2. Commit your changes:

```
git commit -m "Add prime number computation"
```

3. Check the status:

```
git status
```

4. Now is a good time to **mark** the commit with a **tag** to indicate that up until this commit the stuff is working.

```
git tag prime_number
```

Step 4: Create a Remote Repository (on GitHub/GitLab)

1. **Create a remote repository** on GitHub/GitLab.
2. **Link your local repository** to the remote repository:

```
git remote add origin https://github.com/username/git_exercise.git
```

3. **Push your code** to the remote repository:

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

Step 5: Clone the Repository to Another Folder

1. **Navigate out** of the current folder and clone the repository to a new location:

```
cd ..  
git clone https://github.com/username/git_repo.git clone-folder
```

2. **Move into the cloned repository:**

```
cd clone-folder
```

Step 6: Create a Branch Named "fibonacci"

1. **Create a new branch** named `fibonacci`:

```
git switch -c fibonacci
```

The command `switch` allows you to switch between branches and different commits. When the `-c` flag is used in combination, a new branch will be **created** with a designated name as the last parameter, in this case `fibonacci`.

Step 7: Implement a Fibonacci Sequence Generator

1. **Create a new file** `fibonacci_numbers.py` to include a function for generating Fibonacci numbers **[COPY PASTE]**:

```
def fibonacci(n):  
    fib_sequence = [0, 1]  
    while len(fib_sequence) < n:  
        fib_sequence.append(fib_sequence[-1] + fib_sequence[-2])  
    return fib_sequence
```

```
if __name__ == "__main__":  
    fib_n = int(input("Generate how many Fibonacci numbers? "))  
    print(fibonacci(fib_n))
```

2. Add and commit the changes:

Step 8: Merge the "fibonacci" Branch with Main

1. Merge the **fibonacci** branch into the main branch:

```
git merge fibonacci
```

2. **Run the code again** to ensure both prime number and Fibonacci functions are present.

Step 9: Make a Change in the Cloned Repository and Push It

1. Modify **prime_numbers.py** to compute primes between two input numbers **[COPY PASTE]**:

```
def get_primes(m, n):  
    primes = []  
    for i in range(m, n+1):  
        if is_prime(i):  
            primes.append(i)  
    return primes  
  
if __name__ == "__main__":  
    m = int(input("Start prime search from: "))  
    n = int(input("Find primes up to: "))  
    print(get_primes(m, n))
```

2. **Add, commit, and push** the changes:

Step 11: Make Another Change in the Original Repository

1. **Return to the original folder:**

```
cd ../git_exercise
```

2. Modify **prime_numbers.py** to print the number of primes found **[COPY PASTE]**:

```
print(f"Number of primes found: {len(get_primes(m, n))}")
```

3. Add and commit the changes to make a merge conflict:

```
git add prime_numbers.py
git commit -m "Add print statement for number of primes"
```

4. Push the changes:

```
git push
```

Step 12: Resolve the Merge Conflict

Merge conflicts are very common when working on projects and files which are frequently edited. Therefore it is important to know what a **merge conflict** entails and how you can quickly be on with your day. A merge conflict happens in Git when two branches have conflicting changes to the same part of a file, and git doesn't know how to automatically merge them. This requires manual intervention to resolve.

1. Go back to the cloned folder and pull the latest changes to start the merge conflict:

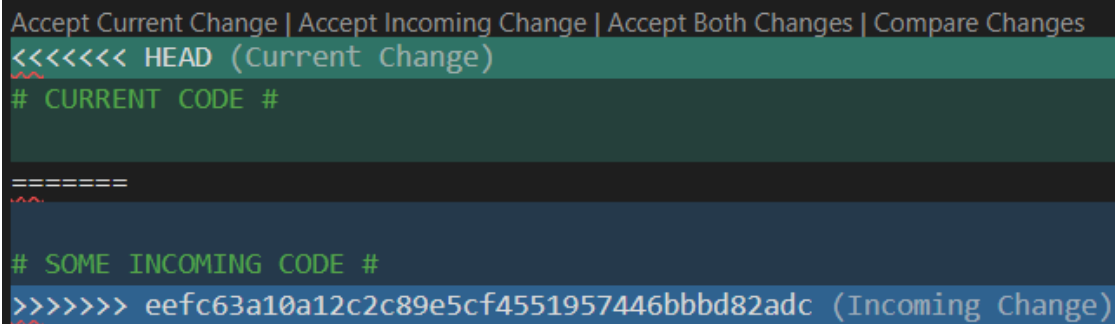
```
cd ../clone-folder
git pull
```

2. Understanding the merge conflict.

A merge conflict is not dangerous fear not! Lets dissect it together.

- A merge conflict will begin with a <<<<<< HEAD statement to indicate the changes of the current branch.
- Following this statement will be some # CURRENT CODE # of the current branch.
- Then some divider signs ===== to separate the **current** and **incoming** code.
- Then # SOME INCOMING CODE #.
- Finally the end of the merge conflict will be shown with some arrows and the commit id >>>>>> eefc6...2adc. This is convenient if you should want to investigate the commit that caused the merge conflict further.

If you are using VSCode the merge conflict will look like this:



```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
# CURRENT CODE #

=====
# SOME INCOMING CODE #
>>>>>> ee6c63a10a12c2c89e5cf4551957446bbbd82adc (Incoming Change)
```

It can be resolved either by only keeping the code you want and deleting the rest or simply clicking one of the buttons at the top.

2. **Resolve the merge conflict** in `prime_numbers.py`. Combine the changes **[COPY PASTE]**:

```
def get_primes(m, n):
    primes = []
    for i in range(m, n+1):
        if is_prime(i):
            primes.append(i)
    print(f"Number of primes found: {len(primes)}")
    return primes
```

3. **Stage and commit** the resolved file.
4. **Push** the resolved changes.

Final Thoughts

This exercise covers the complete workflow of using Git in a project: initializing a repository, working with remotes, cloning, resolving conflicts, branching, and merging.

A common work/ project situation

Often when working on projects in a team merge conflicts, changing files, deleted files and name changes are common. We should be able to handle all of these situations in git.