Ministry of Education and Science of Ukraine

National Technical University of Ukraine

"Igor Sikorsky Kyiv Polytechnic Institute"

LABORATORY WORK №1

Git: working with branches and conflicts

The purpose of the work: is to learn how to work with branches and resolve conflicts during git merge.

KULUBECİOGLU MEHMET

KYİV-2024

Input Data

- Installed Git on your PC
- GitHub account

Output Data

- The final version of the repository
- The report

Task

1. Learn how to work with Git.
2. Learn to resolve merge conflicts.

Experiment Program

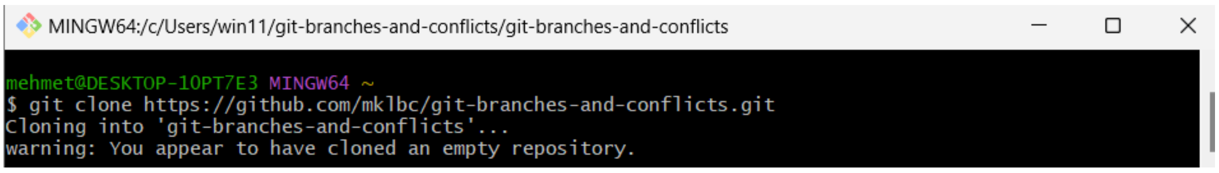
Step 0: Choose a Programming Language

I chose Python for this project.

Step 1: Create a New Repository on GitHub and Clone It

1. Created a new repository on GitHub named git-branches-and-conflicts.
2. Cloned the repository to my local machine:

sh


MINGW64:/c/Users/win11/git-branches-and-conflicts/git-branches-and-conflicts

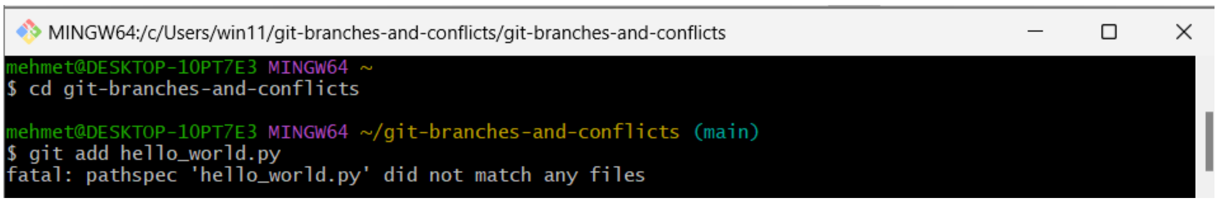
```
mehmet@DESKTOP-10PT7E3 MINGW64 ~
$ git clone https://github.com/mklbc/git-branches-and-conflicts.git
Cloning into 'git-branches-and-conflicts'...
warning: You appear to have cloned an empty repository.
```

Step 2: Create `hello_world.py` in the Main Branch

Created a file named `hello_world.py` with the following content:

```
print("Hello, world!")
```

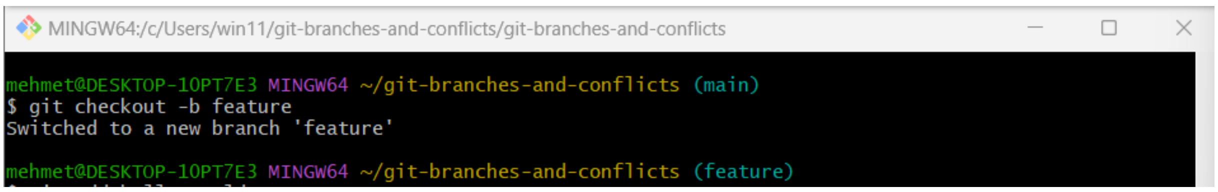
Added and committed the file to the main branch


MINGW64:/c/Users/win11/git-branches-and-conflicts/git-branches-and-conflicts

```
mehmet@DESKTOP-10PT7E3 MINGW64 ~
$ cd git-branches-and-conflicts

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git add hello_world.py
fatal: pathspec 'hello_world.py' did not match any files
```

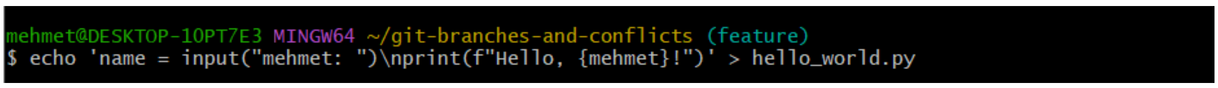
Step 3: Create a New Branch `feature` and Modify the Program

Created a new branch named `feature`:


MINGW64:/c/Users/win11/git-branches-and-conflicts/git-branches-and-conflicts

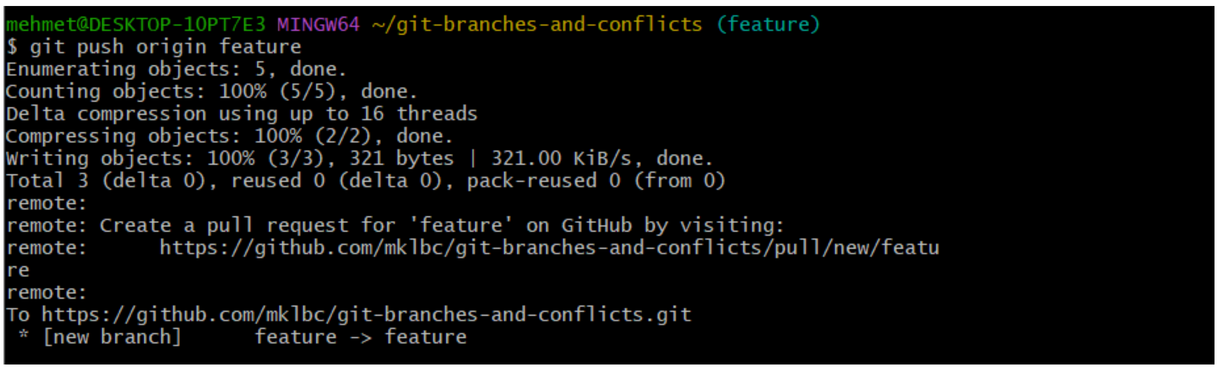
```
mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git checkout -b feature
Switched to a new branch 'feature'

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (feature)
```

Modified `hello_world.py` to ask for the user's name and greet them



```
mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (feature)
$ echo 'name = input("mehmet: ")\nprint(f"Hello, {mehmet}!")' > hello_world.py
```

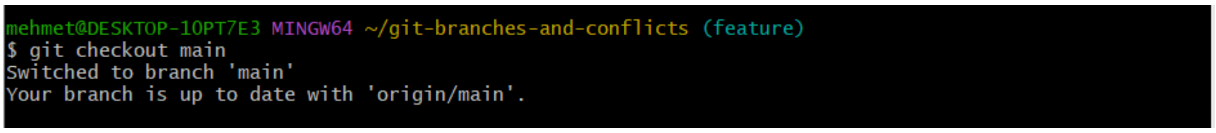
Added and committed the changes:



```
mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (feature)
$ git push origin feature
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 321 bytes | 321.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'feature' on GitHub by visiting:
remote:      https://github.com/mklbc/git-branches-and-conflicts/pull/new/featu
re
remote:
To https://github.com/mklbc/git-branches-and-conflicts.git
 * [new branch]      feature -> feature
```

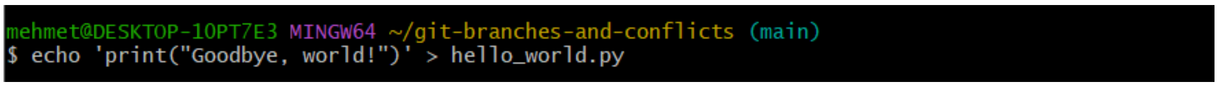
Step 4: Switch Back to the Main Branch and Modify the Program

Switched back to the main branch



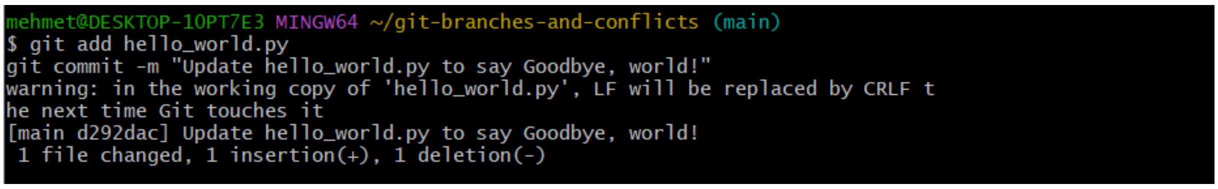
```
mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (feature)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

Modified `hello_world.py` to print "Goodbye, world!"



```
mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ echo 'print("Goodbye, world!")' > hello_world.py
```

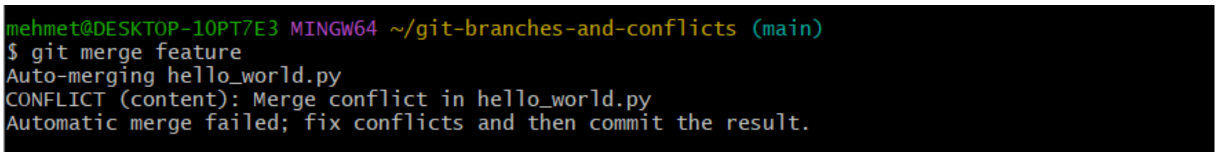
Added and committed the changes



```
mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git add hello_world.py
git commit -m "Update hello_world.py to say Goodbye, world!"
warning: in the working copy of 'hello_world.py', LF will be replaced by CRLF t
he next time Git touches it
[main d292dac] Update hello_world.py to say Goodbye, world!
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Step 5: Merge `feature` Branch into Main and Resolve Conflicts

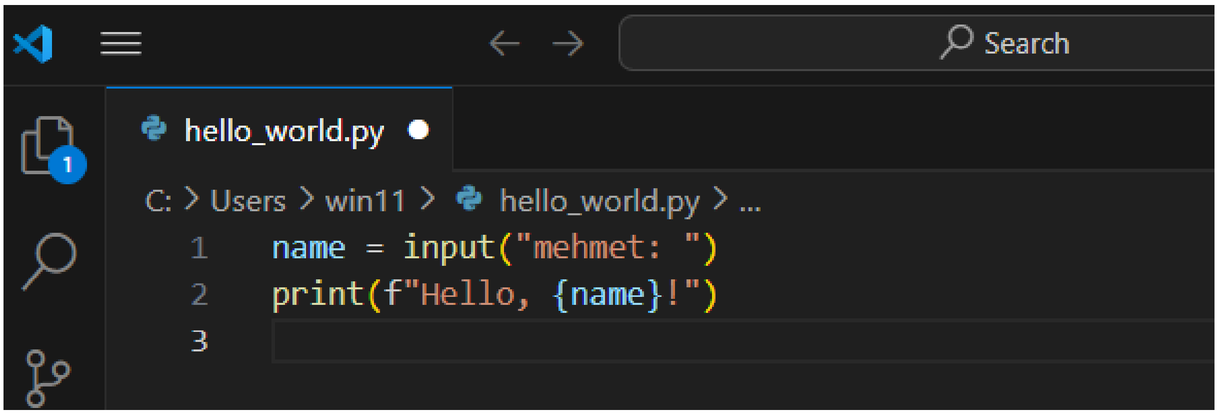
Merged the `feature` branch into the main branch:



```
mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git merge feature
Auto-merging hello_world.py
CONFLICT (content): Merge conflict in hello_world.py
Automatic merge failed; fix conflicts and then commit the result.
```

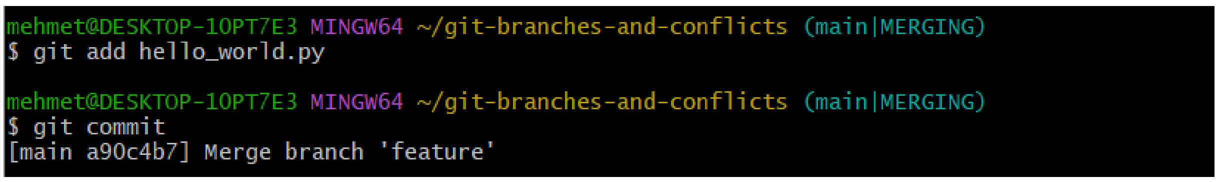
This caused a conflict in `hello_world.py`.

Resolved the conflict by making the file print a personalized greeting:



```
name = input("mehmet: ")
print(f"Hello, {name}!")
```

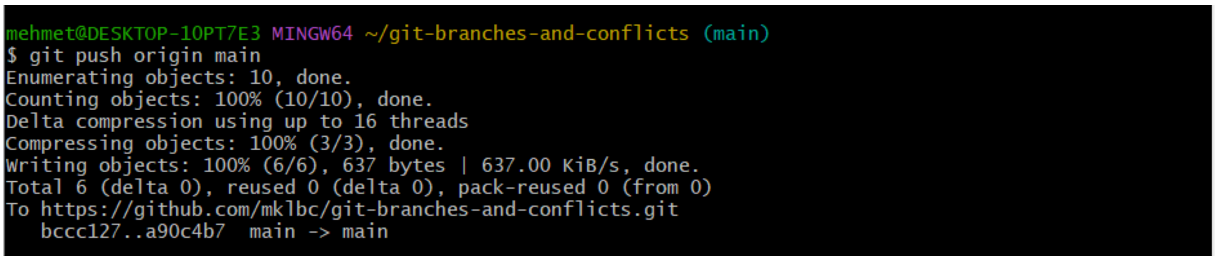
Added and committed the resolved changes:



```
mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main|MERGING)
$ git add hello_world.py

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main|MERGING)
$ git commit
[main a90c4b7] Merge branch 'feature'
```

Step 6: Push the Final Changes to GitHub

Pushed the final changes to the main branch on GitHub:



```
mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git push origin main
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 637 bytes | 637.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/mklbc/git-branches-and-conflicts.git
   bccc127..a90c4b7  main -> main
```

Conclusion

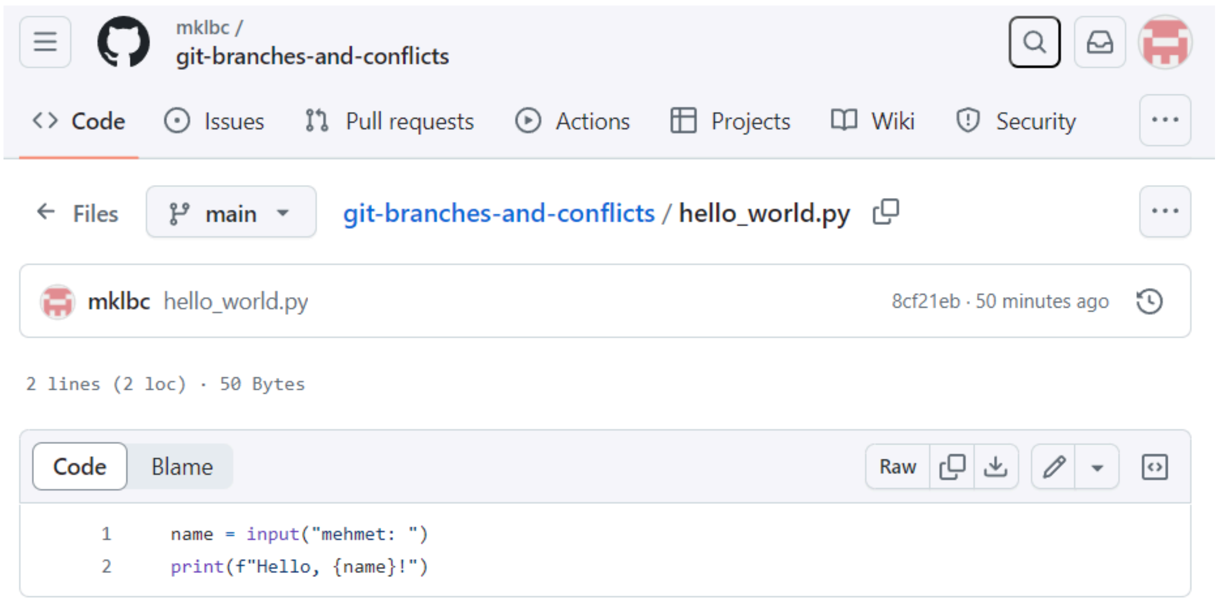
This laboratory work successfully demonstrated the process of working with Git branches and resolving merge conflicts. The key steps included creating and switching between branches, making changes, and resolving conflicts that arise during merges.

Repository Structure

- `hello_world.py`: A Python script that asks for the user's name and greets them.

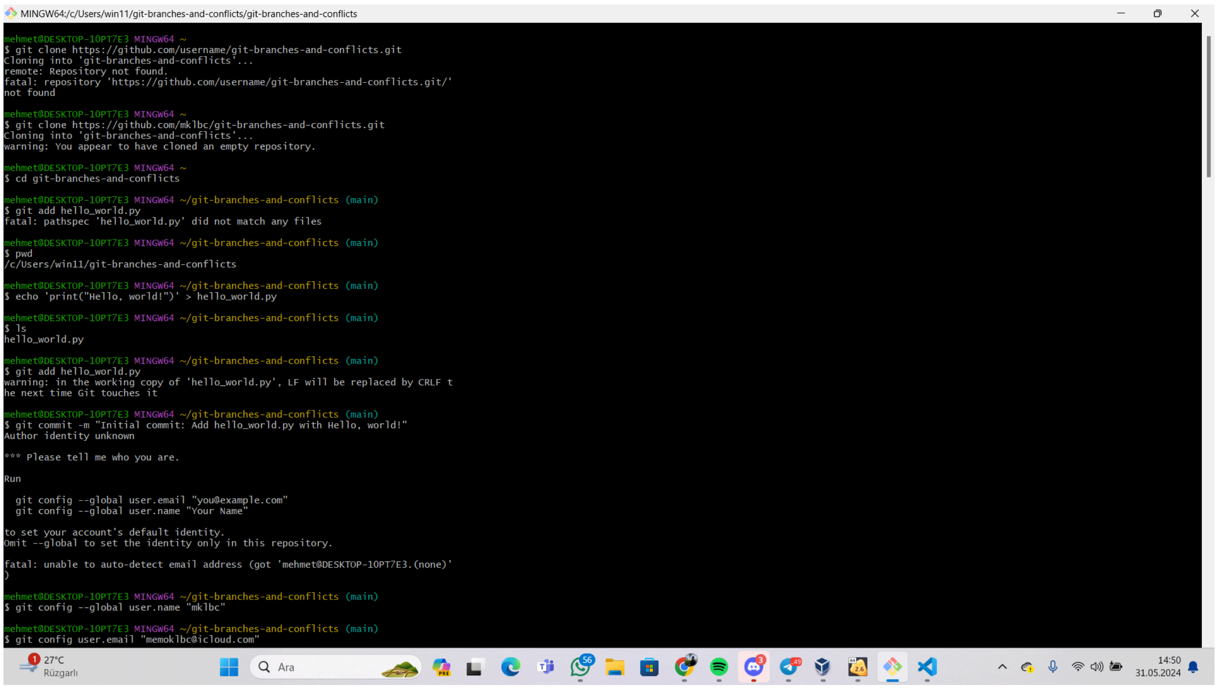
Final `hello_world.py` Content

Final `hello_world.py` Content

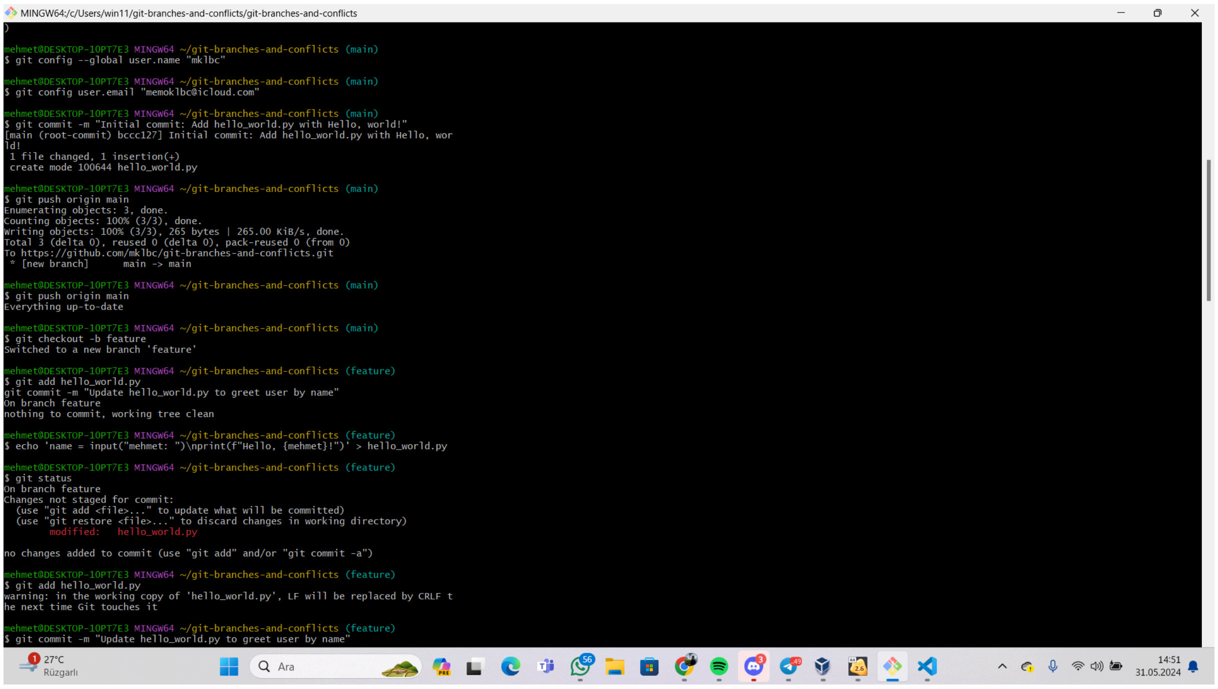


```python
name = input("mehmet: ")
print(f"Hello, {name}!")
```

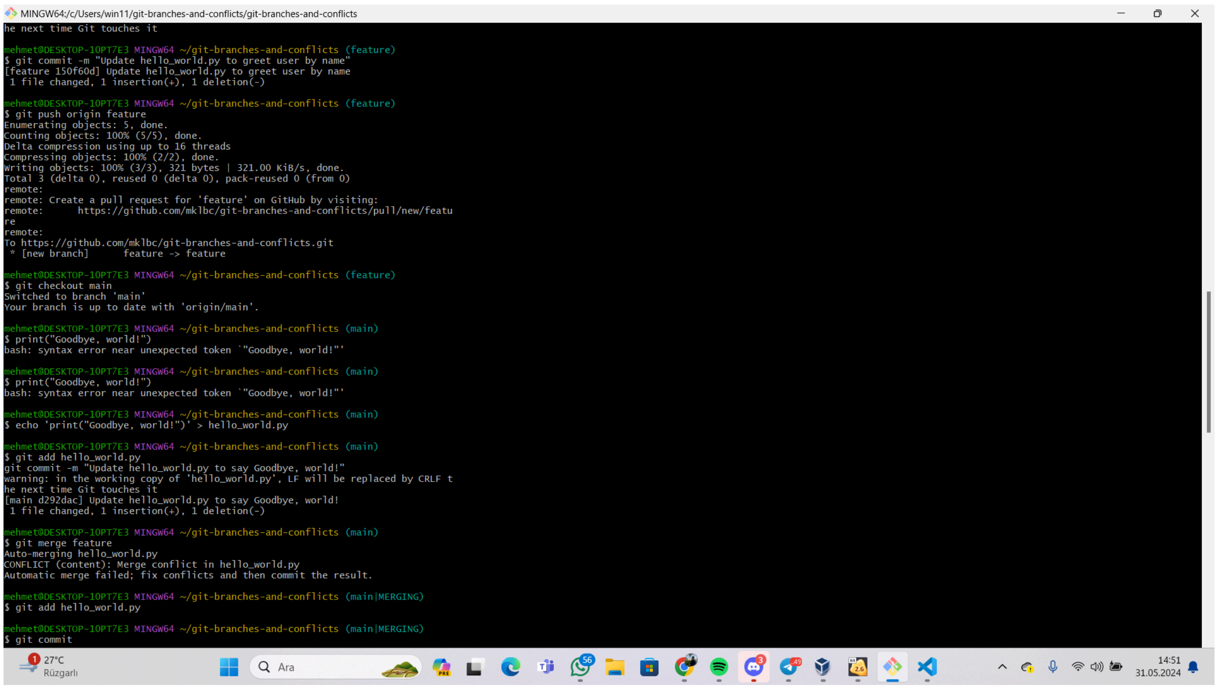
My codes:


MINGW64:/c/Users/win11/git-branches-and-conflicts/git-branches-and-conflicts
27°C
Rüzgarlı
Ara
14:50
31.05.2024

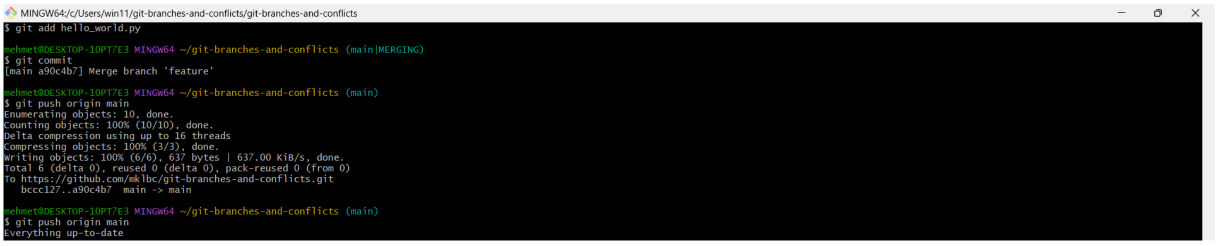
```
mehmet@DESKTOP-10PT7E3 MINGW64 ~
$ git clone https://github.com/username/git-branches-and-conflicts.git
Cloning into 'git-branches-and-conflicts'...
remote: Repository not found.
fatal: repository 'https://github.com/username/git-branches-and-conflicts.git/'
not found

mehmet@DESKTOP-10PT7E3 MINGW64 ~
$ git clone https://github.com/mklbc/git-branches-and-conflicts.git
Cloning into 'git-branches-and-conflicts'...
warning: You appear to have cloned an empty repository.

mehmet@DESKTOP-10PT7E3 MINGW64 ~
$ cd git-branches-and-conflicts

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git add hello_world.py
fatal: pathspec 'hello_world.py' did not match any files

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ pwd
/c/Users/win11/git-branches-and-conflicts

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ echo 'print("Hello, world!")' > hello_world.py

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ ls
hello_world.py

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git add hello_world.py
warning: in the working copy of 'hello_world.py', LF will be replaced by CRLF t
he next time Git touches it

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git commit -m "Initial commit: Add hello_world.py with Hello, world!"
Author identity unknown

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'mehmet@DESKTOP-10PT7E3.(none)'
)

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git config --global user.name "mklbc"

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git config user.email "memoklbc@icloud.com"
```


MINGW64:/c/Users/win11/git-branches-and-conflicts/git-branches-and-conflicts

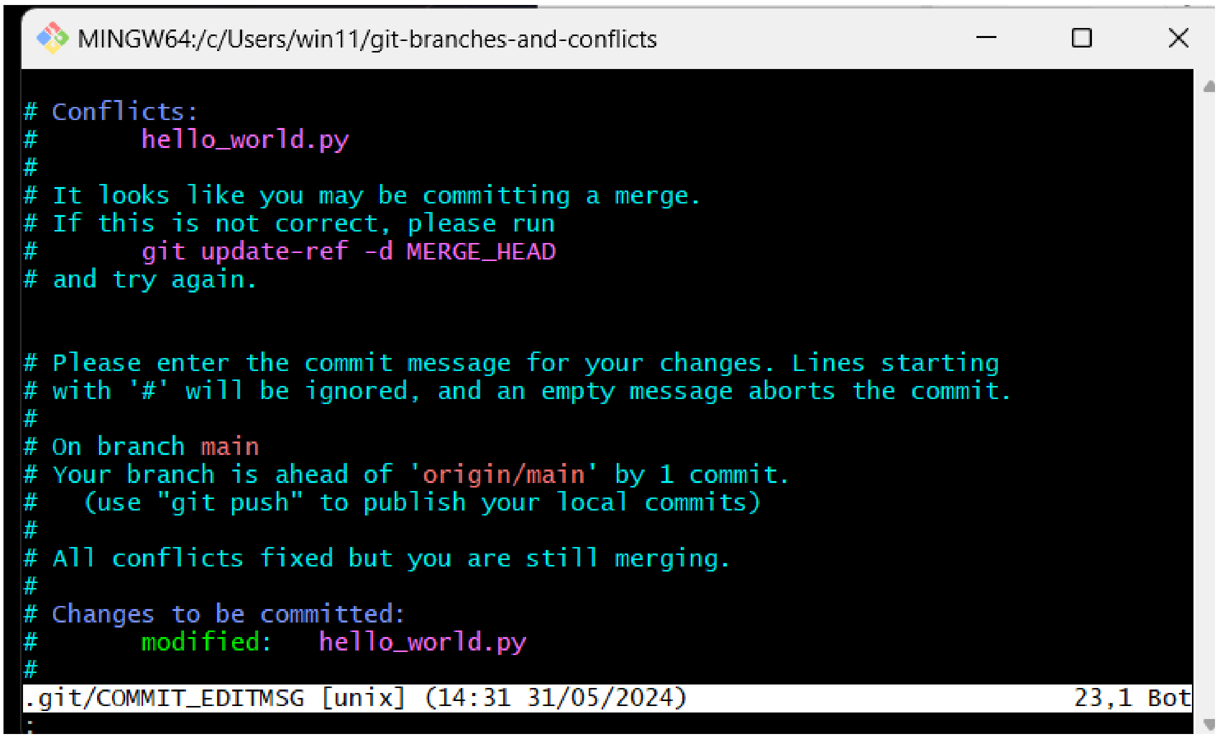
```
)

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git config --global user.name "mklbc"

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git config user.email "memoklbc@icloud.com"

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git commit -m "Initial commit: Add hello_world.py with Hello, world!"
[main (root-commit) bccc127] Initial commit: Add hello_world.py with Hello, wor
ld!
 1 file changed, 1 insertion(+)
 create mode 100644 hello_world.py

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git push origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 265 bytes | 265.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/mklbc/git-branches-and-conflicts.git
 * [new branch]      main -> main

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git push origin main
Everything up-to-date

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git checkout -b feature
Switched to a new branch 'feature'

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (feature)
$ git add hello_world.py
git commit -m "Update hello_world.py to greet user by name"
On branch feature
nothing to commit, working tree clean

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (feature)
$ echo 'name = input("mehmet: ")\nprint(f"Hello, {mehmet}!")' > hello_world.py

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (feature)
$ git status
On branch feature
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello_world.py

no changes added to commit (use "git add" and/or "git commit -a")

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (feature)
$ git add hello_world.py
warning: in the working copy of 'hello_world.py', LF will be replaced by CRLF t
he next time Git touches it

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (feature)
$ git commit -m "Update hello_world.py to greet user by name"
```


MINGW64:/c/Users/win11/git-branches-and-conflicts/git-branches-and-conflicts

```
he next time Git touches it

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (feature)
$ git commit -m "Update hello_world.py to greet user by name"
[feature 150f60d] Update hello_world.py to greet user by name
 1 file changed, 1 insertion(+), 1 deletion(-)

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (feature)
$ git push origin feature
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 321 bytes | 321.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'feature' on GitHub by visiting:
remote:      https://github.com/mklbc/git-branches-and-conflicts/pull/new/featu
re
remote:
To https://github.com/mklbc/git-branches-and-conflicts.git
 * [new branch]      feature -> feature

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (feature)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ print("Goodbye, world!")
bash: syntax error near unexpected token `"Goodbye, world!"'

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ print("Goodbye, world!")
bash: syntax error near unexpected token `"Goodbye, world!"'

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ echo 'print("Goodbye, world!")' > hello_world.py

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git add hello_world.py
git commit -m "Update hello_world.py to say Goodbye, world!"
warning: in the working copy of 'hello_world.py', LF will be replaced by CRLF t
he next time Git touches it
[main d292dac] Update hello_world.py to say Goodbye, world!
 1 file changed, 1 insertion(+), 1 deletion(-)

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git merge feature
Auto-merging hello_world.py
CONFLICT (content): Merge conflict in hello_world.py
Automatic merge failed; fix conflicts and then commit the result.

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main|MERGING)
$ git add hello_world.py

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main|MERGING)
$ git commit
```


MINGW64:/c/Users/win11/git-branches-and-conflicts/git-branches-and-conflicts

```
$ git add hello_world.py

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main|MERGING)
$ git commit
[main a90c4b7] Merge branch 'feature'

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git push origin main
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (6/6), 637 bytes | 637.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/mklbc/git-branches-and-conflicts.git
   bccc127..a90c4b7  main -> main

mehmet@DESKTOP-10PT7E3 MINGW64 ~/git-branches-and-conflicts (main)
$ git push origin main
Everything up-to-date
```



```
# Conflicts:
#       hello_world.py
#
# It looks like you may be committing a merge.
# If this is not correct, please run
#       git update-ref -d MERGE_HEAD
# and try again.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch main
# Your branch is ahead of 'origin/main' by 1 commit.
#   (use "git push" to publish your local commits)
#
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#       modified:   hello_world.py
#
.git/COMMIT_EDITMSG [unix] (14:31 31/05/2024)                   23,1 Bot
:
```

Learnings

- How to create and manage branches in Git.
- How to handle and resolve merge conflicts in Git.
- Best practices for committing and pushing changes to a remote repository.

Recommendations

- Always ensure to pull the latest changes from the remote repository before starting new work.
- Frequently commit changes to avoid large conflicts.
- Clearly comment on conflict resolutions to maintain project history integrity.

Control Questions

1. **What is a version control system and what is it used for? What are Git and GitHub?**

   A version control system (VCS) is a tool that helps manage changes to source code or other collections of files over time. It allows multiple people to collaborate on a project, track changes, revert to previous versions, and manage branches of development. Git is a distributed version control system that tracks changes to files and coordinates work on those files among multiple people. GitHub is a web-based platform that uses Git for version control and provides a collaborative interface for hosting and reviewing code, managing projects, and building software alongside a community.

2. **What is a commit and how is it used in Git?**

   A commit in Git is a snapshot of the project's history at a specific point in time. It represents a set of changes that are recorded in the repository. Each commit has a unique identifier (a hash) and includes a message describing the changes. Commits are used to save changes to the repository, allowing users to keep track of what has been done and revert to previous states if needed.

3. **What is the main difference between branches and the main branch in Git?**

   In Git, a branch is a separate line of development. Branches allow users to work on features, bug fixes, or experiments in isolation from the main codebase. The main branch (often called `main` or `master`) is the primary branch where the stable, production-ready code resides. Changes from other branches are typically merged into the main branch once they are reviewed and tested.

4. **What steps are required to resolve a merge conflict in Git?**

   To resolve a merge conflict in Git, follow these steps:

   - Identify the conflicting files: Git will indicate which files have conflicts.
   - Open the conflicting files: Locate the conflict markers (<<<<<<<, =======, >>>>>>>) that show the differing changes.
   - Resolve the conflicts: Edit the files to combine the changes as needed, ensuring the final version is correct.
   - Remove the conflict markers: Make sure to delete the conflict markers after resolving the conflicts.
   - Add the resolved files: Stage the resolved files using `git add`.
   - Commit the merge: Complete the merge by committing the resolved changes with `git commit`.