Lab: Merging Changes Together

In this lab, we will work on resolving merge conflicts.

Creating a merge conflict

Here, we will show you a simulation of how merge conflicts appear.

```
cd ~/work
mkdir test-dir
cd test-dir
git init .
echo "some content" > example.txt
git add example.txt
git commit -am "initial commit"

[master (root-commit) a45c22d] initial commit
1 file changed, 1 insertion(+)
create mode 100524 example.txt
```

In the given example, we create a **test-dir** new directory. Next, we create **example.txt** text file with some content and add it to the repository and commit it. As a result, we have a new repository with one master branch and **example.txt** file. The next step is creating another branch to use as a conflicting merge.

```
git checkout -b branch_to_merge
echo "completely different content to merge later" > example.txt
git commit -am "edit the content of example.txt to make a conflict"
[branch_to_merge 4221135] edit the content of example.txt to make a conflict
1 file changed, 1 insertion(+), 1 deletion(-)
```

In the above example, we create and check out **branch_to_merge** branch. After creating, we overwrite the content in **example.txt** file and commit the new content. After doing all this, the commit overrides the content of **example.txt:**

```
git checkout master
Switched to branch 'master'
echo "content to add" >> example.txt
git commit -am "added content to example.txt"
[master 11ab34b] added content to example.txt
1 file changed, 1 insertion(+)
```

This bunch of commands checks out the master branch attaching the content to **example.txt** and committing it. So, our repository is put to the state where we have one commit in the master branch and one in the **branch_to_merge** branch. The final step is to execute the [git merge]{.kbd .highlighted} command after which conflict will occur:

```
git merge branch_to_merge
Auto-merging example.txt
CONFLICT (content): Merge conflict in example.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Identifying merge conflicts

As we have already seen, Git displays output which indicates that a conflict has appeared. Execute the `git status` command to see the unmerged paths:

```
git status
On branch master
You have unmerged paths.
(fix conflicts and run "git commit")
(use "git merge --abort" to abort the merge)
Unmerged paths:
(use "git add <file>..." to mark resolution)
both modified:   example.txt
```

The example.txt file appears in a modified state. Execute **cat** command to put out the contents of the example.txt file. We can see these visual marks:

```
<<<<<<< HEAD
=======
>>>>>>> branch_to_merge
```

The **=======** marks is the center of the conflict. The content between the center and the HEAD line is the content existing in the current branch master that the HEAD reference is pointing to. Read more about visual marks on the [git merge]{.kbd .highlighted} page.

Resolving merge conflicts

To resolve a merge conflict you should edit the conflicted file. Open the **example.txt** file in the editor and remove all the marks. The changed file has the following look:

```
some content to mess with
content to add
completely different content to merge later
```

Execute the `git add` command to stage the new merge content. Next, create a new commit to complete the merge:

```
git add .
git commit -m "the conflict in example.txt is merged and resolved"
```