Recipe for producing "merge conflicts"

pre-requisite

This recipe assumes that you've a code repository available on GitHub.

Recipe steps

Step 1:

Navigate to project root directory using `cd` command. Ensure using `pwd` command

Step 2:

Check which branch you're currently on using `git branch -a` command and ensure that you're on `master` branch

Step 3:

Checkout to `master` branch using `git checkout master` command

Step 4:

Run `git pull origin master` - to pull the latest changes from master branch on GitHub (server)

Step 5:

With all the steps above, we are ensuring 2 things -

```
- We are ensuring that we are on master branch on our machine.
- We are also ensuring local copy of "master" is up-to-date
  with remote copy of "master."
```

- Now is the good time to create a new "feature" branch to work on new feature development.
    - Use `git checkout -b feature1` command to create a "feature" branch.
    - You can name your branch anything you like.
    - This recipe assumes you have created branch with name `feature1`.
    - Typically, developers prefer name based on kind of feature they are working on. For example - `payment_feature`

Step 6:

Make changes to any file in your project. Let's assume your file is `foo.py`. And changes are on `line number 5`

Step 7:

Perform `git add foo.py` to add file to staging area.

Step 8:

Perform `git commit -m "commit-message"` to commit changes.

Step 9:

Push changes to GitHub using `git push origin feature1`

Step 10:

Now, create a "pull request" on GitHub using "create pull request" button.
Use base branch as `master` and compare branch as `feature1`
Now merge your changes in `master` branch using "merge pull request" button.

Step 10:

Now let's assume you've another developer (assume name `John`) in your team. "John" is working on his
"feature2" development. So he creates `feature2` branch and makes his own changes.
- John has changed something on `line number 5` and has pushed his changes to the

remote using similar process.
- John raises a "pull request" and he observes that there is "merge conflict" on GitHub

Step 11:

John has 2 options to resolve conflicts.

- Click on "web editor" (John can use this editor to resolve conflicts on server itself)
- Click on "command line" (John will get set of commands from GitHub that he needs to try on command line)
- John can use "web editor" and he can see "conflict dividers" something similar to following -

~~~
```
<<<<<<< HEAD
Adding some content to mess with it later
Append this text to initial commit
=======
Changing the contents of text file from `feature2`
>>>>>>> `feature2`

```
~~~

Step 12:

John needs to remove un-necessary code and dividers and should click on "mark as resolved"

Step 13:

Finally click on "merge pull request" to merge `feature2` into `master` branch.

Step 14:

Now since John has updated `master` copy on remote (GitHub), you will have to update your `master` copy on your machine. You can do so by running `git pull origin master` on your local machine (while you are on `master`)