Lesson | Weekend

Intermediate JavaScript (/intermediate-javascript) / Team Week (/intermediate-javascript/team-week) / Git with Collaborators: Workflow

Text

When we work on a development project as a collaborator, we have commit privileges to the main repository on GitHub. The development workflow for collaborators in a team takes advantage of some Git functionality that we don't often use on pair projects: **Branching** and **merging**. Below is a *general* outline of this workflow. Use this lesson as a reference when working on your group project this week:

Git Team Workflow

---

1. Setup Github Repo

Build a repo on Github and add all team members as collaborators (https://www.learnhowtoprogram.com/lessons/git-with-collaborators-setup). Clone the GitHub main repo to each pair's desktop with `$ git clone <repo-url>`. Navigate to the project with `$ cd <project-directory-name>`.

2. Create Branches

Each pair creates (and switches to) their own feature branch locally by running `$ git checkout -b <branch-name>`. Note that repos come with a main branch by default; you will not need to create one manually.

Tip: If you're ever uncertain which branch you're currently on, run `$ git branch`

3. Code

Pairs complete work on their own branches, adding and committing throughout the process.

4. Pull Origin Main into Local Main

Before pushing work, pairs pull any new code teammates may have merged into the Github main branch (AKA *origin main* or *remote main*) into their local main branch.

This is done by navigating into local main with `$ git checkout main`, then running `$ git pull origin main`. This pulls code from Github's main into the local main.

Generally speaking, this command triggers one of the following three results:

Example A - *Pulling down no new changes*. There is no new content to pull into the local branch, it is already up-to-date.

```
$ git checkout main
Switched to branch 'main'
Your branch is up-to-date with 'origin/main'.
$ git pull origin main
From https://github.com/test-user/my_project
 * branch            main     -> FETCH_HEAD
Already up-to-date.
```

Example B - *Pulling down new changes with no merge conflicts*. There was new content from the Github's main (also known as *origin main*), but Git was able to merge it into the branch automatically.

```
$ git checkout main
Switched to branch 'main'
$ git pull origin main
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused
0
Unpacking objects: 100% (3/3), done.
From https://github.com/test-user/my_project
 * branch            main     -> FETCH_HEAD
   2833d6c..51f2f03  main     -> origin/main
Updating 2833d6c..51f2d03
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

Example C - *Pulling down new changes with merge conflicts*. There was new content from the Github's main, but Git was *not* able to merge it into the branch automatically. The user will need to do so by hand.

```
$ git checkout main
Switched to branch 'main'
$ git pull origin main
From https://github.com/test-user/my_project
* branch            main     -> FETCH_HEAD
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the r
esult.
```

5. Resolve Conflicts (if necessary)

If conflicts occur (as seen in Example C above), conflict tags will appear in the impacted files:

```
...
<<<<<<< HEAD
   This is code from the local main branch!
=======
   This is code from the origin main branch!
>>>>>>>
....
```

Content above the `=======` refers to code from the branch you wish to update (in this case, local main). Content below comes from the branch that is *not* being updated (in this case, Github's main). Resolve these conflicts by replacing everything between `<<<<<<<` and `>>>>>>>` with the code you ultimately want in the project's 'final draft' or main. Then remove conflict tags, and commit the changes.

6. Merge Local Feature into Local Main

After gathering the most up-to-date code from main origin, ensure code in the local feature branch works correctly with code from the main branch by merging feature branch into local main.

Confirm you're still located in your local main branch by running `$ git branch`. Then, merge local feature into local main with `$ git merge <feature-branch-name>`. If any merge conflicts occur, follow the steps above.

7. Add Local Main Code to Origin Main

Pairs ensure the application still looks and functions correctly. Once everything is working as desired, the local version of the main branch can be added to origin main.

There are two primary ways to do this. If all of the collaborators for a project are not physically present to review code, then it is best practice to submit a Pull Request via GitHub so that each member has a chance to review the changes before any new code is added. If all collaborators are present and approve of the changes to the code, or if you are working solo, then it is acceptable to merge into main directly and push the updated main to Github.

If all collaborators are **not** present (most common):

Pull Requests

a. Push Branch to Github

Feature code has been merged into local main branch successfully, and any subsequent refactoring committed. Push this to GitHub as the new feature branch by running `$ git push origin <branch-name>`.

b. Pull Request

A pull request is created by navigating to the feature branch on Github, selecting "New Pull Request" , clicking "Create Pull Request", and including a description of new features in the resulting form. Finally, "Create Pull Request" will create and send the request to project collaborators.

c. Review and Merge

Repo owners and collaborators may then view this notification for more information, and options to merge the request into main. More information on that is available here (https://help.github.com/articles/merging-a-pull-request/).

If **all** collaborators are present, and have okay'd changes, or you are working solo:

Merge to Main

a. Merge Feature into Main

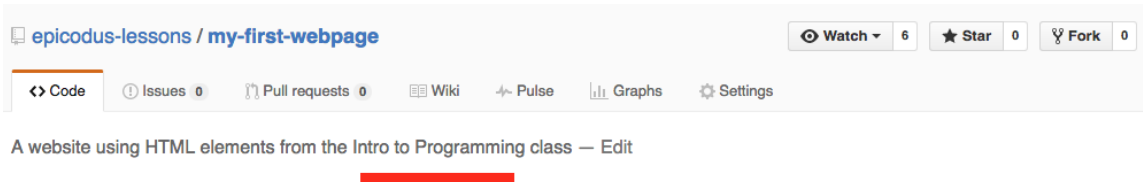
Feature code has been merged into the local main branch successfully, and any subsequent refactoring committed. Contents of local main are then pushed directly into origin main by switching to main with `$ git checkout main`, and merging the appropriate feature branch into main with `$ git merge <branch-name>`.

Again, the result should be very similar to one of the code snippets listed in Step 5. If examples A or B are received, there is either nothing new to merge or new code was merged automatically. If something akin to example C is received, pairs must repeat Step 6 in order to resolve any merge conflicts.

---

8. Delete

If merging was successful, the feature branch may be deleted if it is no longer in use. The easiest way to do this is through the GitHub repo. Simply visit the repository, and select the "branches" option:



Then, in the "branches" area of your GitHub repository, select the red delete icon next to the branch your group would like to delete:

Previous (/intermediate-javascript/team-week/git-with-collaborators-setup)

Next (/intermediate-javascript/team-week/rewriting-history-with-git)

Lesson 3 of 13

Last updated more than 3 months ago.

disable dark mode

Epicodus (http://www.epicodus.com)