

Git Workflow Lab for Two Programmers

Author: [@PiotrBerebecki](#)

Maintainer: [@PiotrBerebecki](#)

Adapted by: [@jpmaldonado](#)

Initial setup

- You're working in a team of two on a project for a client.
- Steps 1 to 8 in this section should be completed by one of you, which we'll refer to as `Programmer 1`.

Step 0 - Setup (for both)

- Add username and email to use with your Github account.

```
git config --global user.name "Your Name"
```

```
git config --global user.email "your.name@email.server"
```

- Set up merge to visually deal with conflicts.

```
git config --global mergetool.meld.path "path to meld.exe"
```

```
git config --global merge.tool meld
```

- You will also need to [add an SSH key to your Github account](#).

Optional

- Create a cool alias command to have more explicit history.

```
git config --global alias.tree "log --graph --abbrev-commit --decorate --  
date=relative --format=format:'%C(bold blue)%h%C(reset) - %C(bold green)  
(%ar)%C(reset) %C(white)%s%C(reset) %C(dim white)- %an%C(reset)%C(bold  
yellow)%d%C(reset)' --all"
```

Step 1 - Programmer 1 creates a new GitHub repo and clones it

1. Go to your Github and create a new repo, initialising it with a `README.md`.
2. Clone this new repository using your terminal and `cd` into it.

```
$ git clone 'PASTE THE URL OF YOUR REPOSITORY HERE'
```

Step 2 - Raise your issues on the work to be done

Normally, you would decide on which "features" you were going to build and then break these down into smaller issues before starting the work.

For the sake of this exercise, we're just going to [add one issue](#) at the moment. Your client wants a beautifully styled heading for the homepage.

1. Raise a new issue with a descriptive title.
2. In the body of the issue, provide more detail about how to complete the work.
3. Assign yourselves to this issue.

Step 3 - Create and move to a new branch

1. Create a branch with a unique and descriptive name. For example, `create-heading-with-shadow`.

```
$ git branch create-heading-with-shadow
```

2. Leave the master branch by switching to the new branch you have just created.

```
$ git checkout create-heading-with-shadow
```

Step 4 - Code the required feature

1. Add the following code into a file called `index.html`.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <link rel="stylesheet" href="style.css">
  <title>Git Workflow Workshop</title>
</head>
<body>
  <h1 class="some-heading">GIT WORKFLOW WORKSHOP</h1>
</body>
</html>
```

2. Create a new file called `style.css` which contains:

```
.page-heading {
  color: red;
}
```


Step 5 - Add the new files to the staging area

1. Add `index.html` and `style.css` to the [staging area](#).

```
$ git add index.html style.css
```

Step 6 - Commit your changes

1. Commit the files that are in the staging area. It is a good practice to link your commit to an existing issue by typing `Relates #1`. Thanks to using the hash symbol followed by the relevant issue number your commit will be [automatically linked to an existing issue](#).

```
$ git commit -m 'add git workshop heading & shadow styling  
> Relates #1'
```

Step 7 - Push your local version up to GitHub

At this point, your remote repo looks exactly the same as at the beginning. You need to push your changes.

1. **Push** the `create-heading-with-shadow` branch up to the "origin" i.e. the GitHub repo that you cloned from.

```
$ git push origin create-heading-with-shadow
```

Step 8 - Create a pull request

1. Programmer 1 navigates to the repository on [GitHub.com](#) and creates a [pull request](#).
 - Add a descriptive title (e.g. `Create page heading`) and leave a comment linking the pull request to the issue.
 - Select Programmer 2 as an [assignee](#).

Step 9 - Programmer 2 merges the pull request 👍

- You should never merge your own pull requests. A PR gives the rest of your team the chance to review before your changes are merged into `master`. In your projects, you will be asking the other pair to do this.

Programmer 2 reviews the changes and [merges the pull request](#) on [GitHub.com](#).

Splitting the work

Your client has just called you and asked to improve the heading.

1. Spelling mistake in the heading (the word 'WORKSHOW' should be replaced with 'WORKSHOP')
2. The name of the css class in the heading needs to be updated so that existing styles in the `style.css` file can take effect (`class="some-heading"` should be replaced with `class="page-heading"`).

You decide that one of you (**Programmer 1**) will resolve issue number 1 while the other person (**Programmer 2**) will resolve issue number 2.

Note: Only one line in the `index.html` file needs to be modified.

Step 1 - Programmer 2 clones the repo

1. Make sure both teammates have a cloned repo, so you each have a local version on your own computer

```
$ git clone 'PASTE THE URL OF YOUR REPOSITORY HERE'
```

Step 2 - Raise these 2 new issues

1. Create the following two issues and assign each one to a different person

- Fix spelling typo in `<h1>` heading (Programmer 1)
- Correct the class name of `<h1>` heading to match the existing class name in the css file (Programmer 2)

Step 3 - Both programmers create one branch each and switch to them

1. Both programmers create one branch each: `fix-typo-heading` (Programmer 1) and `update-class-heading` (Programmer 2).

```
# Programmer 1:  
$ git branch fix-typo-heading  
  
# Programmer 2:  
$ git branch update-class-heading
```

1. Both programmers leave the master branch by switching to the new branches.

```
# Programmer 1:  
$ git checkout fix-typo-heading  
  
# Programmer 2:  
$ git checkout update-class-heading
```

Step 4 - Both programmers open their `index.html` files and make one requested change each

1. Programmer 1 fixes only the spelling typo in the heading (WORKSHOW -> WORKSHOP). Please do not update the class name. This is dealt with by Programmer 2.

```
<h1 class="some-heading">GIT WORKFLOW WORKSHOP</h1>
```

2. Programmer 2 updates only the class name of the heading (`class="some-heading"` -> `class="page-heading"`). Please do not fix the spelling mistake. This is dealt with by Programmer 1.

```
<h1 class="page-heading">GIT WORKFLOW WORKSHOW</h1>
```

Step 5 - Both programmers save their `index.html` files and check status

1. Both programmers save their `index.html` files.
2. Both programmers check the `status` of their files, to confirm that `index.html` has been modified.

```
$ git status
```

Step 6 - Both programmers add the modified `index.html` file to the staging area

1. Both programmers add their modified `index.html` files to the staging area.

```
$ git add index.html
```

Step 7 - Both programmers commit their changes

1. Both programmers commit the changes. Don't forget the multi-line commit message with the referenced issue.

```
# Programmer 1:  
$ git commit -m 'Fix typo in page heading  
> Relates #<issue number>'
```



```
# Programmer 2:  
$ git commit -m 'Update class name in heading  
> Relates #<issue number>'
```

Step 8 - Programmer 1 switches to `master` branch and pulls down the remote `master` branch

We have so many programmers working on this project now, who knows what changes may have happened to the `master` branch since the last time we looked at the remote version that's on GitHub?

1. Programmer 1 switches to `master` branch.

```
$ git checkout master
```

Step 8 (cont.)

2. Programmer 1 **pulls** the `master` branch from the remote (GitHub repo) to make sure that the local version of `master` is up to date with the remote (GitHub) version of `master`. (There should be no changes since neither of you has pushed any changes to the remote yet.) **It is a good practice to regularly check for changes on the remote before pushing your local changes.**

```
$ git pull origin master
```

3. Programmer 1 switches back to the `fix-typo-heading` branch.

```
$ git checkout fix-typo-heading
```

Step 9 - Programmer 1 pushes `fix-typo-heading` branch to remote

1. Programmer 1 `pushes` `fix-typo-heading` branch to remote

```
$ git push origin fix-typo-heading
```


Step 10 - Programmer 1 creates a pull request

1. Programmer 1 navigates to the repository on [GitHub.com](https://github.com) and creates a [pull request](#).
 - Add a descriptive title (e.g. `Fix the spelling mistake in page heading`) and leave a comment linking the pull request to the issue.
 - Select Programmer 2 as an [assignee](#).



Step 11 - Programmer 2 reviews the pull request

Programmer 2 [reviews the pull request](#)

1. Step through each commit (in this case one)
2. Check the "Files changed" tab for a line-by-line breakdown.
3. Click "Review changes" and choose:
 - "Comment"
 - "Approve"
 - "Request changes"

Step 12 - Programmer 2 merges the pull request 👍

1. Programmer 2 merges the pull request on [GitHub.com](#).

Step 13 - Programmer 2 switches to `master` branch, pulls the remote `master` branch, tries to merge it into `update-class-heading` branch and  resolves merge conflicts 

1. Programmer 2 switches to `master` branch.

```
$ git checkout master
```

2. Programmer 2 **pulls** the remote `master` branch to make sure that the latest version of the project is available locally.

```
$ git pull origin master
```

Step 13 (cont.)

3. Programmer 2 switches back to the `update-class-heading` branch.

```
$ git checkout update-class-heading
```

4. Programmer 2 tries to merge `master` branch into `update-class-heading` branch.

```
$ git merge master
```

Step 13 (cont.)

5. There should be a 🌟 merge conflict 🌟 since the line with the `<h1>` heading is different. Merge conflict should be highlighted with HEAD and master markers as follows:

```
<body>

<<<<<< HEAD
    <h1 class="page-heading">GIT WORKFLOW WORKSHOW</h1>
=====
    <h1 class="some-heading">GIT WORKFLOW WORKSHOP</h1>
>>>>>> master

</body>
```

- You can use `git mergetool your_file` to solve the merge graphically. Choose the correct version and save the changes.

Step 13 (cont.)

6. Programmer 2 removes HEAD and master markers and leaves only one line with `<h1>` heading so that both issues are addressed.

```
<body>  
    <h1 class="page-heading">GIT WORKFLOW WORKSHOP</h1>  
</body>
```

7. Programmer 2 adds the `index.html` file to staging area and commits the changes occurred during the merge conflict.

```
# First add to staging area  
$ git add index.html  
  
# Then commit changes  
$ git commit -m 'Fix merge conflict  
> Relates #<issue number> and #<issue number>'
```

Step 14 - Programmer 2 pushes `update-class-heading` branch to remote

1. Programmer 2 `pushes` `update-class-heading` branch to remote.

```
$ git push origin update-class-heading
```


Step 15 - Programmer 2 creates a pull request

1. Programmer 2 navigates to the repository on [GitHub.com](https://github.com) and creates a [pull request](#) selecting `master` as a base branch and `update-class-heading` as a head branch. Please add a descriptive title (e.g. `Update class name in page heading`) and leave a comment linking the pull request with the issue `#<number>`. Please also select Programmer 1 as an [assignee](#).

Step 16 - Programmer 1 merges the pull request 👍

1. Programmer 1 reviews and merges the pull request on [GitHub.com](#).
2. Programmer 1 opens the live website on GitHub pages to double check the new heading style.