

---

## Astrology Versus Astronomy – Pull Requests and Code Review

Learning Outcomes:

- Review Git commands from the previous exercise
- Practice branching and creating Pull Requests
- Understand the typical GitHub flow

Astrology Versus Astronomy is an app which tells you what your future holds. If the user picks astrology, they will get a horoscope based on their birthdate. If they choose astronomy, they pick a destination planet and enter their age and will receive an itinerary.

In your group you will be pair programming – one pair will work on **astrology** and one on **astronomy**. Choose your teams! One of each pair can drive, and screen share in order to collaborate.

This sprint you will be tasked with finishing the services layer.

In GitHub you will now see that you have access to a repo called GitHubLabTwo.

Whoever of the pairs is driving, click on the green “Code” button and click the clipboard to copy the repository URL. Using Bash, navigate to a place on your computer where you wish to store this lab and type: **git clone <paste the link here>** and then cd into it.

Let’s create a branch and switch to it. Type: **git checkout -b your-branch-name**

Open the project in Visual Studio. Open the “Services” folder. In it you will see four files – in particular AstrologyServiceLayer.cs and AstronomyServiceLayer.cs. Open the appropriate file for your pair. You will see two unfinished methods with some instructions. Start your sprint and complete the Service Layer!



Run the app and test out your cool code (remember only the files you worked on will be functional).

Once you are happy with your portion of the code (Astrology or Astronomy), it's time to push it to GitHub and make a Pull Request!

First – while on your branch you need to add and commit your code. Type:

**git status**

**git add -A**

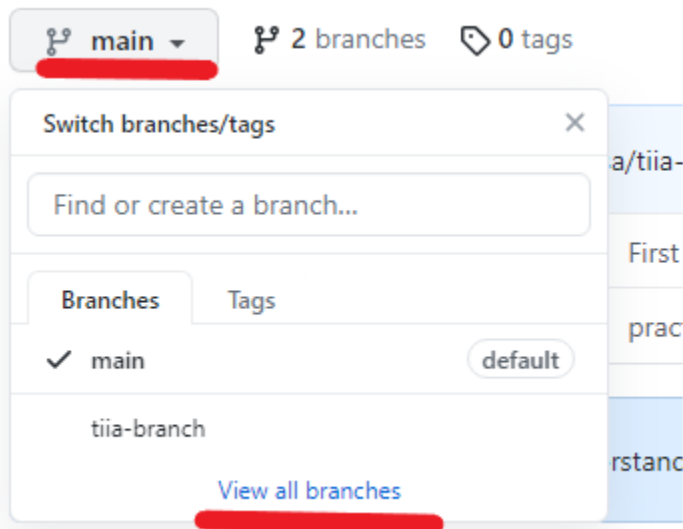
**git commit -m "Finished the Service Layer for <Astrology or Astronomy>"**

To push, type: **git push origin your-branch-name**

Review the console output starting at the asterisk. Note that it is telling you a new remote branch has been set up to track the changes from origin.

Refresh GitHub and view the new branch. Next let's make a Pull Request.

Click on "main" and in the dropdown select "View all branches".




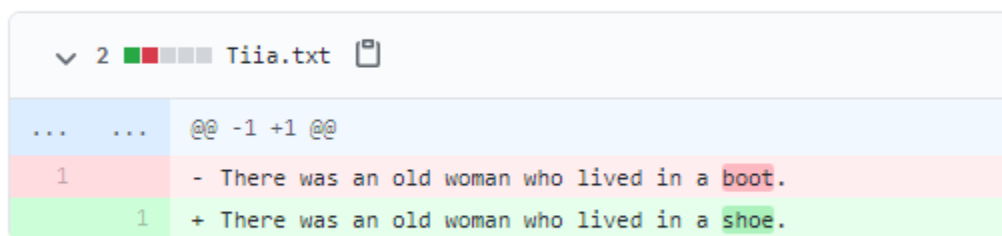


On your branch click “New pull request”.

On the “Open a pull request” page you can see your commit comment. You may add additional information to the message in the “Write” tab.

Scrolling down you now should also be able to see the differences between your branch, and main. Review.

 Showing 1 changed file with 1 addition and 1 deletion.



The screenshot shows a diff view for a file named 'Tiia.txt'. At the top, it says '2' with a green square and a red square, indicating 2 lines changed. Below this, there are two lines of code. The first line is a deletion, marked with a red background and a '1' in the left margin, showing '- There was an old woman who lived in a boot.' The second line is an addition, marked with a green background and a '1' in the left margin, showing '+ There was an old woman who lived in a shoe.' The diff is presented in a side-by-side comparison format.

You can click on the “Split” button to the right to see a side-by-side comparison of the differences

Also note there is a handy message at the top that says “Able to merge” in green. It will also alert you if there are merge conflicts. (Don’t merge just yet!)

Once you are ready, click “Create pull request”. Great! Now your work is ready to be reviewed! Once the other pair is done, you can review one another’s work.

Change gears – now you are a code reviewer! When the other pair is ready, review their request.



Click on Pull requests and find theirs. Open it and view the page. There are several tabs near the top – Conversation, Commits, and Files Changed. Review these. On Files changed, you can start your review.

You will now see the other team's new code. If you hover over lines of code a blue plus sign will appear. You can click on it and add comments to specific lines of code:

After you've input your first comment, click "Start a review":

The other pair is pretty good. But they can probably make some changes / improvements. How well did they name their variables and other class members? Maybe you think Pluto should be included among the planets. Maybe you met with the product owner and the requirements have changed. Leave several comments asking for changes.

After you've finished your comments, click "Finish your review" and select the "Request changes" radio button and click submit. Let the other pair know you're asking for changes before you'll merge their code.

The screenshot shows the 'Finish your review' dialog box in GitHub. At the top, there is a progress bar indicating '0 / 1 files viewed' and a green button labeled 'Finish your review 1'. Below this, the dialog has a title bar 'Finish your review' with a close button. Inside, there are two tabs: 'Write' (active) and 'Preview'. The 'Write' tab contains a rich text editor with a toolbar (bold, italic, link, etc.) and a large text area with the placeholder 'Leave a comment'. Below the text area is a section for attaching files. At the bottom, there are three radio button options: 'Comment' (unselected), 'Approve' (unselected), and 'Request changes' (selected). Each option has a brief description. A red bar highlights the 'Request changes' option. At the bottom left, there is a green 'Submit review' button and a status indicator '1 pending comment'. The bottom of the dialog shows a navigation bar with links: 'Contact GitHub', 'Pricing', 'API', 'Training', 'Blog', and 'About'.



---

When the other pair reviews your Pull Request and requests changes, it's time to get back to coding. In the open Pull Request, on the conversation tab you will see their feedback. Make the requested changes and push back to GitHub.

Let the other pair know you're ready for another review.

And now you can also review the other pair's Pull Request when they are done following up on your feedback. Go into their request.

Scroll down to "New changes since you last viewed" and click "View changes". You can also click on their most recent commit on the commits tab to see the differences.

If you approve, click "Review changes" and select "Approve" and "Submit review".

Now on the "Conversation" tab of the Pull Request, you can click "Merge pull request" and "Confirm merge".

As you're working on a feature, other teammates will be committing and merging code into main as they complete their work. You'll want to keep your branch up to date to minimize conflicts. Depending on the cadence, while you work on a feature, you will want to update your branch at least once a day. As well as before pushing your code and submitting a pull request.

Once both teams' code has been merged in, get each other's changes. In the terminal, make sure you are on the main branch: **git checkout main** and get the latest: **git pull origin main**

Now your local main branch is current.

Switch to your feature branch: **git checkout your-branch-name**



---

Merge the updated code into your branch: **git merge main**

Once everyone's pull requests have been merged you can clean up by clicking "Delete branch" on the feature branches on GitHub.

**\*\*IMPORTANT NOTE** – If you are creating the initial repo for your group / team that they will be cloning – you'll want to include a .gitignore so you won't be pushing all your local metadata up to GitHub (and neither will they). You can Google search for one for .NET Core apps, or, from the root of your project you can type: **dotnet new gitignore** at the command line and one will be generated for you.

Additional resources:

GitHub: Getting Started by Gill Cleeren has good hands-on practice and demos: <https://app.pluralsight.com/library/courses/github-getting-started/table-of-contents>

The chapter "Collaborating Using the GitHub Flow" is particularly useful.

Rob Duncan's guide:

<https://rockfin.sharepoint.com/sites/KitchenSink/SitePages/Git-...a-guide.aspx>