Team Project and Git Setup

Overview

You will set up your team project in RStudio and configure local branches in order to collaborate with teammates.

Things you will need:

- Git installed on your computer and configured with RStudio
- A GitHub PAT
- A team GitHub repository:
  - Team CPS: https://github.com/slicesofdata/cps-167-2025
  - Team EIA: https://github.com/slicesofdata/eia-167-2025
  - Team APT: https://github.com/slicesofdata/tennis-167-2025

Part A: Create an RStudio Project a GitHub Repository

Step 1: Set the Project Type

In RStudio, from the toolbar, select:

File > New Project > Version Control > Git

Step 2: Enter the Repository Url to Clone

Use the **https** repository url for your team:

- Team CPS: https://github.com/slicesofdata/cps-167-2025
- Team EIA: https://github.com/slicesofdata/eia-167-2025
- Team APT: https://github.com/slicesofdata/tennis-167-2025

Warning: Make sure you use **https** and append the suffix **.git** to the

Step 3: Enter the Repository Name

Use the repository name:

- Team CPS: cps-167-2025
- Team EIA: eia-167-2025
- Team APT: tennis-167-2025

Step 4: Specify the project a Sub-directory

Ensure that your project is going to be a sub-directory of your `dataviz` uber directory. **Never** make projects sub-directories of other projects.

Step 5: Open in a New Session

Check the box to open the project in a new session.

Step 6: Finalize Creation

Click **Create Project** to clone the project.

Part B: Check `main` Branch and Upstream

Step 1: Checking Branches

1. In your RStudio Terminal (not R console and not Mac Terminal), check what branches exist both locally and on the remote using `branch -a`. This will provide a listing of **all** branches. `git branch` will return local branches only and `git branch -r` will return the remote branches only.

```
git branch -a
```

This terminal command tells your computer to use Git and run the `branch` command with the `-a` flag. All Git commands will start with `git` and a space.

You should see something like:

```
  dev
  * main
  remotes/origin/HEAD -> origin/main
  remotes/origin/dev
  remotes/origin/main
```

If there are multiple branches, an * will appear next to the active branch. You should see a `main` branch and you may see other branches. Remote branches will start with `remotes/` whereas local branches do not.

Step 2: Switching Branches

You will not work on the `main` branch so you will need to ensure other branches are present both remotely and locally. Switching branches is done using `switch`. By default, you should be on the `main` branch of the repo. You can ensure this using the `switch`. In this case, you likely `switched` to the same branch you were already on.

```
git switch main
```

As before, you can check your local branch using `git branch` to ensure the * is on `main`.

Step 3: Pulling Remote Updates

In order work on branches, you need to have files from them. To obtain changes from the remote (on origin), use `git pull origin main`. You will *pull* changes from the the remote `main` to your local `main` branch.

```
git pull origin main
```

This command tells Git to pull the recent changes, if any, from the GitHub remote.

If you just cloned the repo, pulling should bring in no new files. You will likely see a message that *Already up to date* as you have fetched the branch head. Let's test the command anyway.

```
* branch            main       -> FETCH_HEAD
Already up to date.
```

Notes:

- `origin` = the name of your remote repository on GitHub
- `git pull origin main` and `git push origin main` are foolproof commands as they require you to set the remote (origin) and branch explicitly, leaving no ambiguity.

Step 4: Verifying Upstream Connection

You will need branches that exist locally on your computer to communicate with those on GitHub. This requires what is known as an upstream connection. Without it, you cannot push files.

Set the upstream (one-time only) connection. You will not need to do this again.

```
git push --set-upstream origin main
```

Note: You will be unable to set this upstream if you have not obtained your PAT and configured git.

Part C: First Time Setup for 'dev' Branch

When projects require branches other than `main`, you may need to pull them or create them and switch to them. You will also need to integrate and merge files from one branch into another. The `main` branch will contain the files of the final project. Until the project if ready, you will use a `dev` branch.

Step 1: Switching to and Tracking a `dev` Branch and Checking Upstream

The `dev` branch should exist on the remote. You can check.

```
git branch -a
```

You should see something like the following, which should contain reference to `dev` on the remote, (e.g., `remotes/origin/dev`):

```
dev
* main
remotes/origin/HEAD -> origin/main
remotes/origin/dev
remotes/origin/main
```

If `dev` exists on the remote, you can fetch those changes from the remote to your local machine.

```
git fetch origin
```

And then *switch* to, *c*reate a local branch, and track the remote `dev` branch

```
git switch --track origin/dev
```

The command will create a local `dev` branch that will connect to the remote `dev` branch and it will allow you to track the changes you and your collaborators push there.

WARNING: Do not create the `dev` branch locally as this will create a separate local branch having the same name as an existing remote branch. This would be terribly confusing.

Note:

- `--track` ensures your local `dev` is linked to `origin/dev`

3. Set your upstream (one-time only) so that you can push and pull to the remote `dev` branch.

```
git push --set-upstream origin dev
```

Part D: Switching to and Creating your Local Personal Feature Branch and Setting Upstream

Step 1: Creating your Personal Branch

Each team member will **work on their own branch** to avoid inadvertently overwriting files on either the `main` branch, the `dev` branch, or someone else branch. This approach will allow you to troubleshoot errors on your personal branch rather than on the `dev` branch you share with your team.

Always **be very vigilant of the branch you are working on**. If you are editing files on the `dev` branch or on a collaborator's branch and try to push changes to the remote repo, you will likely have to troubleshoot a bunch of issues. This experience will be educational for sure but will likely not be pleasant for you to troubleshoot.

You will create your personal branch off of the `dev` branch so you need to ensure you are on it first.

1. Ensure that you are on the local `dev` branch.

```
git switch dev
```

2. Switch to and create your own personal branch using `switch -c`. Name your branch using your name in lowercase characters, no spaces, for example, `<my-branch-name>`. Please, don't use Beavis.

```
git switch -c beavis
```

Step 2: Setting your Upstream Connection

Set your upstream connection so that you can pull and push file changes made to your personal branch.

```
git push --set-upstream origin beavis
```

Note: You will not need to complete this step again.

Step 2: Check all Branches

Check that you see your branches.

```
git branch -a
```

You should see 3 local branches, `main`, `dev`, and your personal branch with the * on your personal branch as well as your new personal branch on the remote.

```
  * beavis
  dev
  main
  remotes/origin/HEAD -> origin/main
  remotes/origin/beavis
  remotes/origin/dev
  remotes/origin/main
```