GitHub Workflow for Collaborative Development

Creating a new project

To start a project, one team member should make a GitHub repository as follows:

Go to your GitHub Account and select the **Repositories** page. Click the **New** button to make a new repository. Be sure to add a README and a GitIgnore file with Visual Studio or Java template.

Go to the repository on your GitHub account. Select the **Setting** tab.

Manage access: Add your collaborators to the project. You can add up to three collaborators on a private repo, more on a public one.

Branches: You should be on the **main** branch. Add a branch protection rule with the same name (main). Select "Require pull request reviews before merging", 2 required approving reviews and save changes.

Now create a **dev** branch based on the **main** branch. You will normally merge your code changes into the **dev** branch, and push this code to **main** at intervals when you have a stable build (for example, after each sprint review).

Update the default branch to **dev.** Decide on a branch protection rule for the dev branch, normally it should also have 1 or 2 reviewers.

Working on the project

You might want to have one pair of developers set up the initial code project and folder structure, and push it to **dev** and **main** before adding the protection rules.

Team members can now clone the project on to their own computer.

Do NOT edit the code in the **main** branch or **dev** branch. Instead:

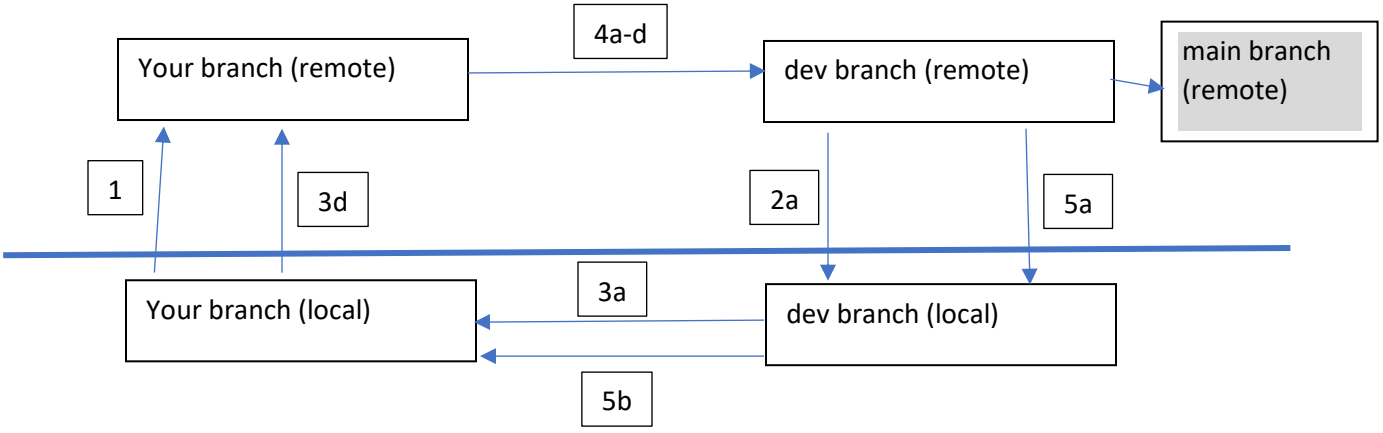
- Pull the **dev** branch to your local machine
- Create a feature branch locally. Give it your own name, and/or the name of the feature you are working on – for example, Cathy_Fireball_API.

  ```
  git checkout –b Cathy_Fireball_API
  ```

- Develop your code on this branch

Check-in Process

1) Commit the changes in your local branch regularly, and push them to remote
2) When you are ready to merge to **dev**, first update your branch to include the latest changes from **dev**:
    a) Locally, switch to **dev** branch and Sync with the server (`fetch` followed by `pull`)
    b) If there are any merge conflicts, resolve them and commit locally
3) Merge the latest changes with your branch
    a) Switch to local your branch and Merge from **dev**
    b) If there are any merge conflicts, resolve them and commit locally
    c) Check everything still builds and tests pass. Commit any changes
    d) Push your branch to remote
4) Pull request
    a) Create a pull request
    b) fill in the code reviewer(s)
    c) The reviewer will look at your code, may add comments, may approve the request
    d) If your code is approved and doesn't need changes, complete the pull request
    e) If your code does need changes, make them locally. Make sure the code still compiles and runs, commit and push to remote. Complete the pull request once it is approved
    f) If you are finished with this feature branch, delete it
5) Update your local **dev**
    a) Switch to **dev** and Sync with the server (pull)
    b) Switch back to your branch and merge from local **dev,** or create a new branch for your next piece of work, based on local **dev**.



Seeing remote branches

If you can't see someone else's remote branch, open a command window in the top directory of your solution and type

$git status - should show you which branch you are on and if there is anything to commit

$git fetch origin – fetches references to any new remote branches.

You still need to pull down a local copy of a branch if you want to use it.

Visual Studio Integration

In Visual Studio, use Extensions -> Manage Extensions to get GitHub Extension for Visual Studio

Tortoise Git

Another useful tool for using Git – download from https://tortoisegit.org/. I find it useful when resolving merge conflicts and viewing revision graphs