Github Flow Walkthru

Introduction

The following document provides a walk thru of a typical development workflow used to add enhancements or bugfixes using the GitHub flow development process. Specifically using a main branch to track the current working state and multiple feature branches for individual changes. Feature branches are kept in sync with updates to the main branch using rebase on those branches rather than merges into the branches to maintain a cleaner, linear commit history. The commit history can then be more easily interpreted when reviewing the changes that occurred over time.

Rebasing is not recommended on feature branches if many developers are committing to the remote feature branch at once, but the goal here is to encapsulate small changes worked on by only one or two development team members at a time. If this is not the case, merge is recommended rather than rebase on feature branches. For this reason, rebasing into the main branch should NOT be done and pull requests should use a merge commit strategy.

This is only meant to illustrate a common case development workflow. Other possible edge cases such as removing, amending, or cherry picking commits are not covered.

Creating and working with feature branches

1. Create a new issue in GitHub
2. On the new issue page, on the right-hand Development menu click `Create a branch`

   > Note: The new branch name should start with the issue number and the name should be a short summary of the issue and its type.

3. Github will provide instructions on how to populate and use the new branch, but in general, you can use the git command line on your local machine to fetch and checkout the new branch.

   ```
   git fetch origin
   git checkout 1-feature-add-capability-xyz
   ```

4. Make changes to the code to address the issue.

5. Commit the changes back to the local feature branch via `git commit` or in VSCode go to the Source Control tab, stage the relevant changes, add a commit message, and click `Commit`.

6. When the changes are ready for review or to save them back to GitHub, do a `git push` or synchronize changes in Visual Studio Code (VSCode) or other development enivronment tool.

   > Synchronizing changes in VSCode will perform both a push and pull on the current branch by default. Changes others have made on the same branch will be integrated into your local branch and made result in conflicts that require resolution. Synchronize with caution.

Creating a Pull Request

1. Once all changes are pushed, create a pull request in Github from the feature branch to the main branch. Since the issue is already linked to the branch, the pull request (PR) will also be linked. A well-formed PR includes the following:

   - Assign the PR to a member of the development team
   - Assign a minimum of two reviewers:
     - A technical reviewer
     - A process reviewer with rights to merge
   - Provide a summary of changes in the description > If linked to one or more issues, description should also include `Closes #N` directive at the end of the description so closing the PR also closes the associated issue(s) where N is the issue number. Multiple statements are allowed for multiple issues.
   - Assign any relevant labels (e.g. bug, enhancement)

2. Monitor the PR for updates and comments from the reviewers. Participate in discussion with reviewers, resolving comments as needed.

3. When all comments are resolved to the reviewers and assignees satisfaction and all automated checks pass, reviewers should approve the PR and one approver should be assigned to merge the PR to the main branch.

Integrating updates from main branch

In some cases, changes will happen in the main branch while you are working a feature. Most likely, you will notice this after creating a pull request for your branch. It will automatically compare the two branches and indicate if there are any conflicts which can result from updates in the main branch.
So to integrate these additional changes into your branch, we will use a rebase to rewrite our commit history with those additional changes. This may result in

conflicts that require resolution. The following commands will perform a rebase and write the changes back to the remote feature branch.

```
# in your feature branch
git fetch origin
git rebase origin/main
# If there are conflicts, rebase will pause
# Resolve conflicts that result and git add/remove fixed files
# then execute 'git rebase --continue' to process
# Repeat as needed until all conflicts are resolved
git push --force-with-lease
```

The first fetch makes sure your local repository has the latest commit history from the remote origin. The rebase then adds the main branch's newer commit history onto your current feature branch. If conflicts occur during this process, it will stop and notify the user. Then, you would resolve any resulting conflicts in the normal way by editing files and using `git rebase --continue` to complete it. If no conflicts occur, it will simply complete its changes to your local branch. Finally, you would push changes to the remote branch, but with an added option that ensures you do not overwrite others changes from the rebased main branch that occurred after your original branching point.

At this point, the remote feature branch should contain the final version of changes that should be submitted back to the main branch. That is, your feature branch changes resolved against any conflicts due to updates to the main branch that were made after your feature branch with a clean commit history. You can review the branch commit history with the `git log` command or by clicking `Commits` link on the Github code page for your feature branch.

A review of the pull request page in GitHub should indicate that the merge has no conflicts and the PR review and approval process can proceed.

Merging a Pull Request

After a pull request has been reviewed, all comments resolved, and approved it can be merged by a maintainer. Use the default merge method which will create a new merge commit on the target branch. This approach appends the feature branch commit history to the target branch commit history. This is why we used rebase on the feature branch in the previous steps. A rebase ensures the feature branch commits align with the current HEAD of the target branch so that only feature branch changes are merged cleanly into the commit history.

Once the merge is complete, the PR will be closed along with its associated issues. The person merging the PR should also delete the feature branch which can be done via a single button click in the PR itself. The develoeprs can also delete their local feature branches on their machines via `git branch -d <branch_name>`. If the PR and issues are on a GitHub project board, the maintainer should also move them into the `In Review` column for final arbitration at the next development team discussion and then moved to `Done`.