Git Workflow

Ryan Minneo

To understand what a Git Workflow is you first need to understand what git is and what it does. Git itself is a database available to everyone to share (mainly) programs. With Git you can upload your code via *committing* and *pushing* it. Git also allows you to set up a group where plenty of people can work on the same programming project at the same time; however, you cannot edit the same exact file at the same time (or at least without overriding one person's work). Editing a line of programs without affecting the main program is known as working on a *branch*. You can either have local branches which will only be on your individual device or you can set up a remote branch so anyone who has access can commit to the branch. Of course, you can *merge* branches back into the main/master branch.

By committing in the IDE of your choice to Git, you save the changes of your repository locally while pushing updates the changes you made in your commit and then sends it to a remote Git repository. You can make multiple commits before you push, but the commits will only be on a local repository until the program is pushed. If you push a branch, it will only affect the relevant branch on the remote repository.

Branches represent different lines of development. Say you want to do a crazy experiment in your project. You can make a new branch and work on that crazy experiment while not actually affecting the main branch in the slightest. Eventually if the crazy experiment completely fails, you can simply delete the branch and go back to updating the main branch. However, if the crazy experiment ends up paying out, then you can merge the branch back into the main branch, effectively overriding the main branch with the crazy experiment branch.

There can however be *merge conflicts*. Merge conflicts can either happen at the beginning of the merging process or during a merge process. Git can sometimes fail to start the merge. This would likely be the effect if the pending changes have already been written over by other commits that have come in from pending local changes. There can also be a merge conflict during a merge where your code will conflict with another developer’s code. Generally GIt will try to merge the files but you will have to resolve conflicted files yourself.

Git workflows don’t have to do with an actual function that git has implemented, but rather strategies for programming teams to use. One example of a git workflow would be a *Centralized Git Workflow.*

Centralized Git Workflow allows every team member to make changes to the main/master branch directly. This workflow would work horribly with large teams, but a small team would thrive using this workflow as it would be very simple to work on for everyone. However, developers under a centralized git workflow have to be very careful with making changes, as committing an unstable change could be catastrophic as the changes would affect the only branch.

Feature branching Git Workflow is, obviously, a feature-based git workflow where every feature has a new branch made specifically for it. This branch should only stay separate from the main branch for so long. As the branch lifespan gets longer, the higher the chances of there being conflicts with merging. Generally, there would be a lot of teams each working on separate branches and this allows the main branch to steer clear of unfinished features as only the finished features would actually be included in the main branch.

Personal Branching Git Workflow relates to the previous workflow, but every branch has a single developer rather than every branch implementing a feature. This workflow works especially well for small teams where every single member is making their own section of the application. This workflow also allows each developer to continuously try different experiments since the worst case scenario would simply be a programmer deleting his branch due to the experiment being a failure. Since there generally won’t be a lot of branches in this workflow, there should be a very low chance of a merge conflict occurring.

Overall, Git workflows are strategies that are implemented for teams working on projects that generally include programming. There are different workflows which all have benefits and downfalls and it is important for groups to know which workflow would work best for their programs/projects.

Works Cited

“Git merge conflicts.” *Atlassian*, https://www.atlassian.com/git/tutorials/using-branches/merge-conflicts. Accessed 1 March 2023.

“Git Workflow.” *Atlassian*, https://www.atlassian.com/git/tutorials/comparing-workflows. Accessed 1 March 2023.

“What is a Git workflow?” *GitLab*, https://about.gitlab.com/topics/version-control/what-is-git-workflow/. Accessed 1 March 2023.

Wisniowski, Kamil. “Git Commit vs Push - What's the Difference ? (Tutorial).” *Cloud Infrastructure Services*, 21 November 2022, https://cloudinfrastructureservices.co.uk/git-commit-vs-push-whats-the-difference-tutorial/. Accessed 1 March 2023.

Work Cited

“Git Workflow.” *Atlassian*, https://www.atlassian.com/git/tutorials/comparing-workflows. Accessed 1 March 2023.