Gitflow Workflow and Other Aspects

Among the several known Git workflows that teams can utilize when working together, the one that will be discussed for now is Gitflow, which has longer living branches and commits compared to the trunk-based development that will be touched on later. The other common workflows are centralized, feature branch, and forking. Before going into Gitflow and the other topics, it is best to understand what Git is. Git was "designed with performance, security and flexibility in mind" and is a modern version control system that was developed in 2005 by Linus Torvalds ("What is Git"). With performance, Git can focus on the file content "when determining what the storage and version history of the file tree should be" ("What is Git"). To protect all objects of the repository, it is secured with SHA1, a cryptographic hashing algorithm, that protects against any accidental or malicious changes. For flexibility, Git can be flexible with support for the various kinds of developed workflows. It has also been designed to support branching and tagging, and any changes with branching and tags are stored as part of the change history. There are some criticisms of the version control system and the top one is that it can be difficult to learn and navigate for beginners.

Gitflow, according to Atlassian, a website that has tutorials on Git, is another branching model that has "feature branches and multiple primary branches" ("Gitflow Workflow"). The workflow also does not have any new commands or concepts as it assigns roles and defines how the branches work. Thus, there are basically two main branches to the workflow, the main branch and then the develop branch. From the develop branch, there is then a feature branch. The feature branch is never merged directly with main; it will have to merge back into the develop branch first and then into the main. The reason why the feature branch must merge back into develop is because of the Git Feature Branch Workflow. Due to the separation of branches, it is good for projects that are on a scheduled release cycle and continuous delivery.

Additionally, the Future Branch Workflow makes future development in a separate branch to make it easier for developers to work on their own feature without messing with the main branch ("Git Feature Branch Workflow"). This means that the main branch will not have broken code unless the feature was merged into the develop branch and then into the main, which no client and developer wants a broken feature. When naming the feature branch that you want to work on, it should be a descriptive name that gives a clear idea of what the branch contains and what it is meant to do. When working on a feature branch, developers can add, commit, and push the changes to update the branch. With pull requests, reviewing code is a huge benefit for developers before officially pushing it to the main branch with the changes("Git Feature Branch Workflow").

Back to Gitflow, the merging of the develop branch into the main branch requires a release branch between the main and develop. After the release branch is forked, it can be merged into the main branch and named as a new version. Once the version has been named and created, the release branch is then merged back into the develop branch as the develop branch may have progressed since the version release. The last branch in the Gitflow workflow is the maintenance branch, or as hotfix branches, which are used to "quickly patch production releases" ("Gitflow Workflow"). Only the maintenance branch is forked from the main branch. After fixing any needed changes, the maintenance branch is merged back into main with an updated version name, but also is merged into the develop branch. Thus, here are the five branches assigned to the role they are given and work accordingly to it. However, since its creation in 2010, the use of Gitflow workflow has declined in favor of trunk-based development.

Before going into trunk-based development, there is a system that helps the development teams and their projects and it is called version control, also known as source control. As mentioned in the first paragraph, an example of this is Git. Version control is the tracking and managing of changes to the software code ("What is version control?"). These systems are software tools that help teams work more efficiently and help reduce the development time. Teams that do not use any form of version control often run into problems such as not being able to see what changes that were made or even the incompatible changes between two pieces of work that should be unrelated. Therefore, the use of version control can provide teams the ability to use traceability and are able to annotate the changes. Annotating can then help others who look at the history to understand what the code is doing and why it was designed.

Compared to Gitflow, trunk-base development has one main branch and makes code integration easier. When developers finish their code, they can merge it into the main branch, but only when the code has been verified. While with Gitflow, developers will have to wait to merge it until it is complete and would require more collaboration (Zettler). Once a developer merges a branch into main, it is good to delete it so there will be less branches that are active. Being able to make quick small commits makes the process of code review more efficient and helps with CI/CD. CI/CD is continuous delivery, meaning there are updates constantly, and it is the "practice of using automation to release software in short iterations" ("Continuous delivery"). So, the quick updates and merging of features compared to the long branches in Gitflow, trunk-based development is a much-preferred workflow.

Now, onto what GitHub repositories are. GitHub is a repository management system, which basically means that it is a server-side flash drive or a cloud drive. The repositories on GitHub can be public or private depending on what you are doing with your code. It will contain all your project files and each of the file's revision history. The repositories can be owned individually or share ownership with your colleagues or organization. Along with managing your work and collaborating, it can see issues to collect user feedback about the code, pull requests to have changes to the repository, and organize tasks to accomplish. There are limits to the repository when viewing content, limits are set so the requests can be completed in a timely manner (GitHub).

The other two workflows that Atlassian's website goes into are centralized and forking. Centralized workflow is great for teams that are transitioning from SVN, Subversion, and it uses a central repository for the single-point entry for all changes to the project. It does not need any other branches as it only has the main branch. If there are any conflicts to the project, the commit history is the best way to go. If a developer tries to commit, but it does not work, Git will refuse to push the code and diverge as it would overwrite the official commits ("Centralized workflow"). This workflow is great for small development teams, however as the team size increases, it is best to look at other workflows.

Finally, the other workflow is forking. It means that each team member will have two Git repositories, a private local repository and a public server-side one. Here the server-side repository acts as the central codebase. It is "most often seen in public open source projects" and begins with the official public repository on the server ("Forking Workflow"). But when a developer wants to work on the project, they do not directly copy the server, but a fork is made to make a copy of it also on the server. Their own copy cannot be accessed by their colleagues as they have their own. To commit, developers will have to push the commit to their own repository and then will have to file a pull request with the main one that will then let the project know that an update is ready.

Works Cited

Atlassian. “Centralized workflow.” *Git Workflow | Atlassian Git Tutorial*, www.atlassian.com/git/tutorials/comparing-workflows#centralized-workflow.

---. “Continuous delivery.” *Continuous Delivery | Get started with CI/CD | Atlassian*, www.atlassian.com/continuous-delivery.

---. “Forking Workflow.” *Forking Workflow | Atlassian Git Tutorial*, www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow.

---. “Git Feature Branch Workflow.” *Git Feature Branch Workflow | Atlassian Git Tutorial*, www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow.

---. “Gitflow Workflow.” *Gitflow Workflow | Atlassian Git Tutorial*, www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow.

---. “What is Git.” *What is Git: become a pro at Git with this guide | Atlassian Git Tutorial*, www.atlassian.com/git/tutorials/what-is-git.

---. "What is version control?" *What is version control | Atlassian Git Tutorial*, www.atlassian.com/git/tutorials/what-is-version-control.

GitHub. “About Repositories.” *About Repositories - GitHub Docs*, docs.github.com/en/repositories/creating-and-managing-repositories/about-repositories.

Zettler, Kev. “Trunk-based development.” *Trunk-based Development | Atlassian*, www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development .