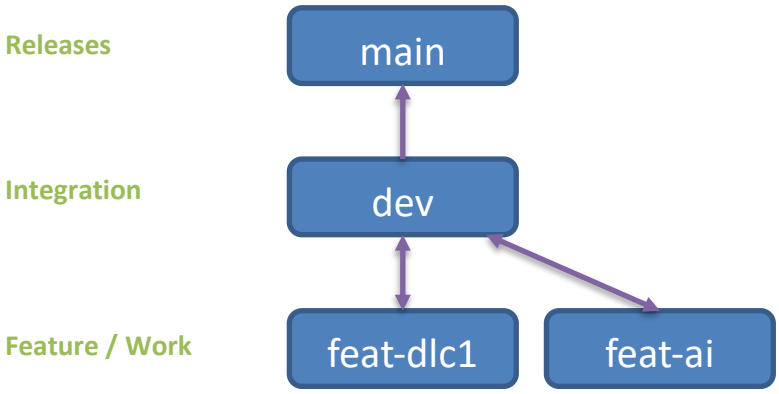
Tutorial – Git Branching

Prerequisite: Git Basics

Before you begin this tutorial, ensure you have already created a repository containing files that you can work with. This tutorial will build upon those files and should not be started before the other is complete.

Reading: Git Flow

The Git Flow branching approach calls for three layers of branches:



These are as follows:

- **Release** branches, stable enough to release to market with
- **Integration** branches, stable enough to test and merge work into
- **Feature/Work** branches, purpose-built and subject to heavy change as work occurs
    - These are often by topic or by person
    - These may be deleted after their usefulness has ended (e.g., a feature is complete)

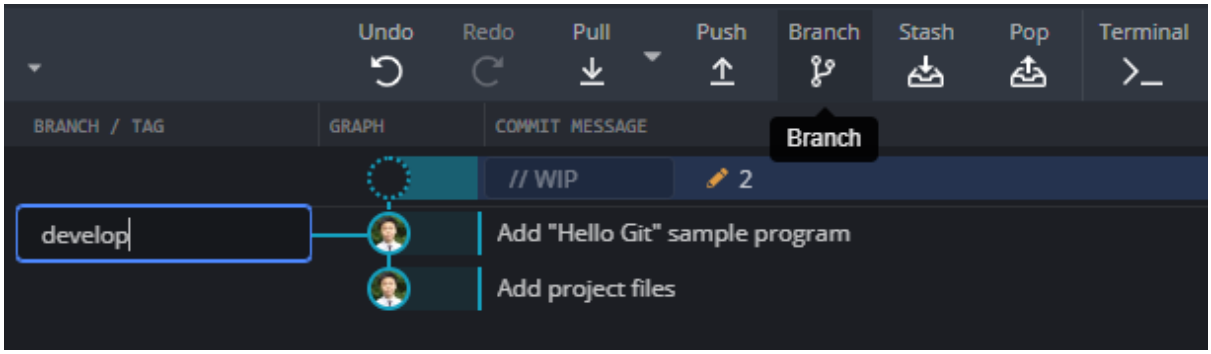
In this tutorial, we will create those branches and create work to merge from our “Feature” branches into the “Integration” branches.

Creating a Branch

By default, a Git repository begins with a "main" or "master" branch containing your initial and subsequent commits. If we want to allow for different people or teams to work on separate parts of the project simultaneously, we can create a new branch, which can be pushed and pulled from separately.

"develop" Branch

When you create a new branch, its **HEAD** revision (aka its latest commit) is the same as your current branch's latest commit. In this case, let's create a "**develop**" branch to start working from.



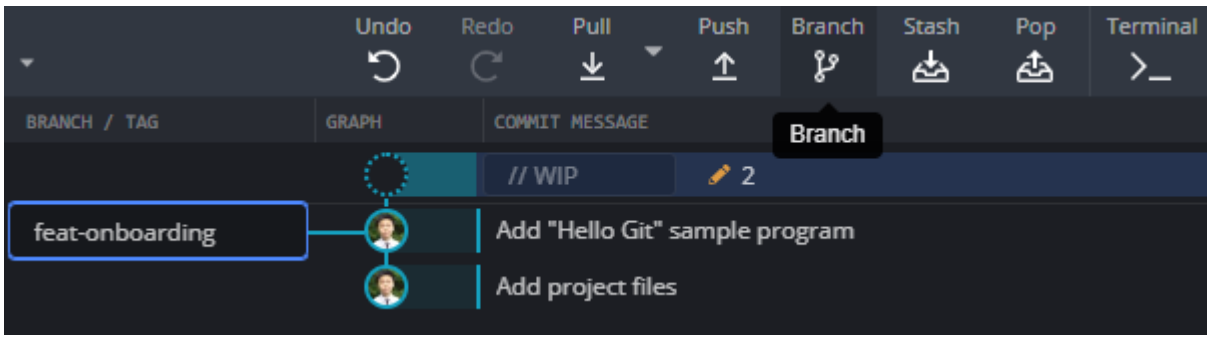
Creating a Branch in GitKraken

If there any uncommitted changes in your work, it will remain as you switch from your current branch to your newly created branch.

This branch is for integration purposes and may be rarely worked on directly unless the individual is working on integration tasking. It is more often merged with to integrate work from other developers.

"feat-onboarding" Branch

Let's create a "feat-onboarding" branch that will provide the user with some helpful instructions on how to use our C++ CLI program. Once again, create a new branch, this time based off of the "develop" branch.



Creating the "feat-onboarding" branch in GitKraken

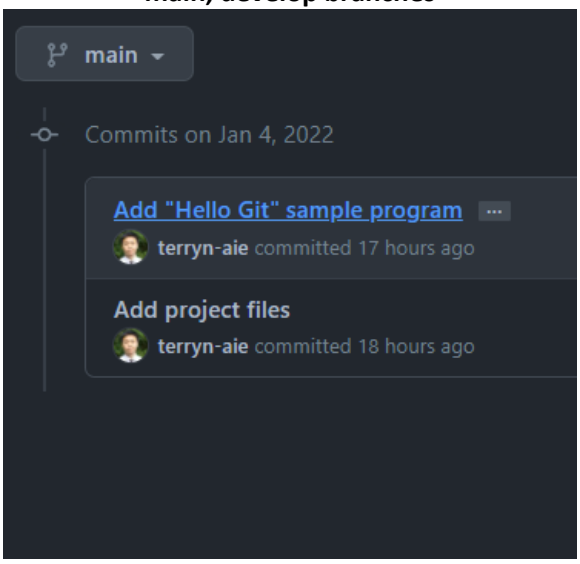
Once we're in that branch, we can begin working and creating commits in that branch with our changes.

Go ahead and open the C++ project and add a few lines as though we were instructing the user on how to use our program. Here's a basic example of how your program could work:

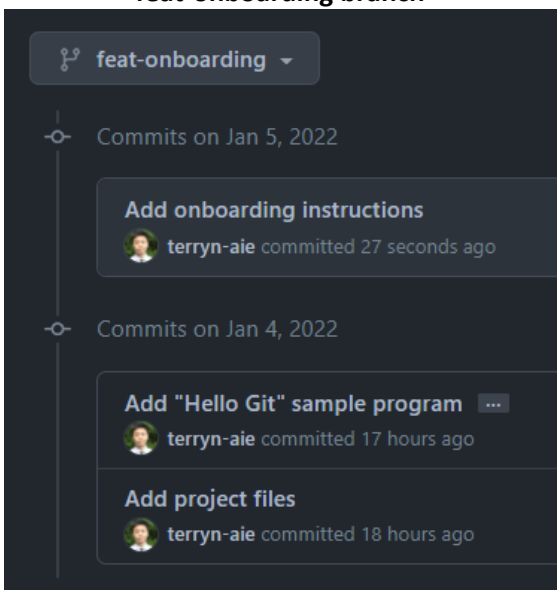
```cpp
#include <iostream>

int main()
{
    std::cout << "Welcome to the Australian Science program-aided enrichment center." << std::endl;
    std::cout << "We hope your time in the classroom has been a pleasant one." << std::endl;
    std::cout << "For your own safety and the safety of others, please ______." << std::endl;

    return 0;
}
```

Once you've finished adding instructions to your program, **add**, **commit**, and **push** your work, as you would normally. When you review the repository either locally or on GitHub, you'll find that this change is only present on the "feat-onboarding" branch.

main, develop branches



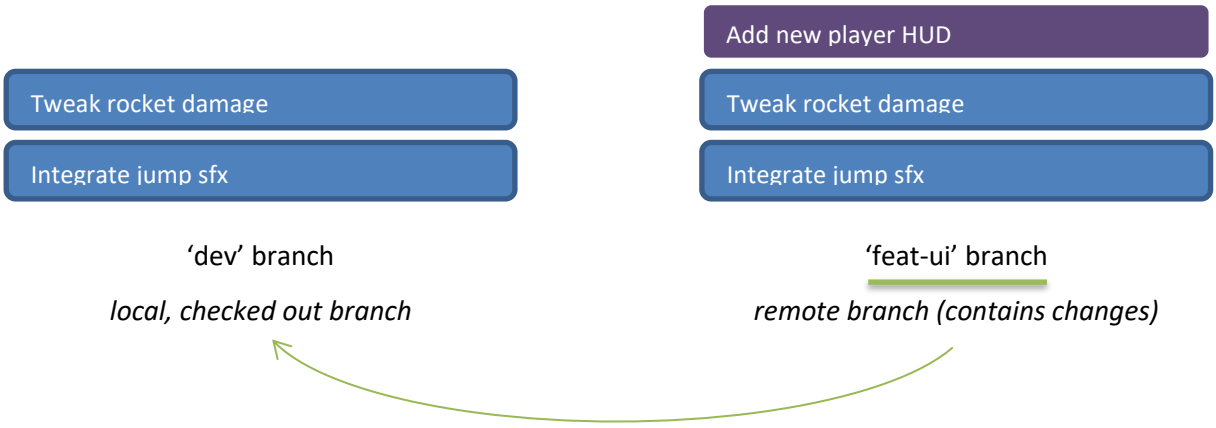
feat-onboarding branch



Since this branch has its own history of commits, you will be able to work on this separately from other people without having to worry about who is pushing first/second/etc. Once your work is ready to be shared with others, we'll want to integrate it with the rest of the project by **merging** into the **develop** branch.

Merging Branches

When we merge branches, Git looks at the differences between the history of the two branches.

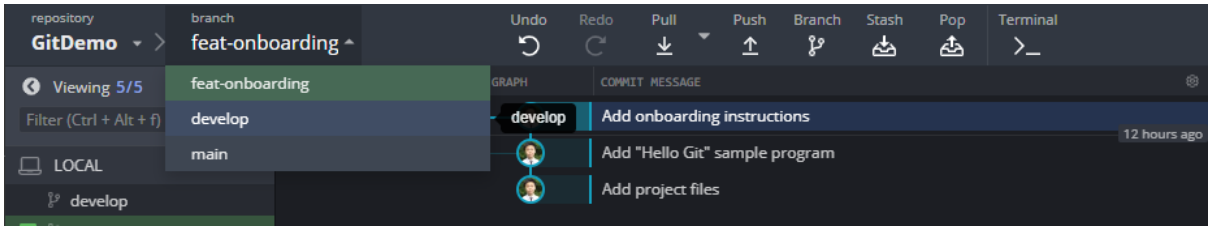


If our current branch (referred to as the "local" branch) is missing commits that are present in the branch we are merging in (referred to as the "remote" branch), then those commits will be added to the local branch.

We can use this process to integrate work on our "feat-onboarding" branch into our "develop" branch.

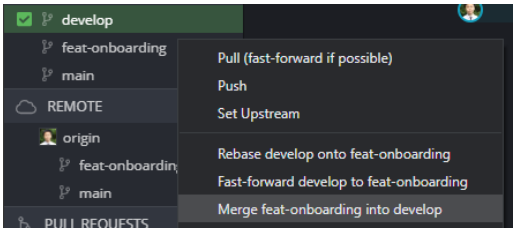
Merging "feat-onboarding" into "develop"

Git only changes the branch we are currently checked out on, so if we want to update "develop" we need to check it out. Using your Git client, switch to the "develop" branch we created earlier.



Switching Branches in GitKraken

Once we are on the "develop" branch, we can tell Git to merge the "feat-onboarding" branch into the "develop" branch.



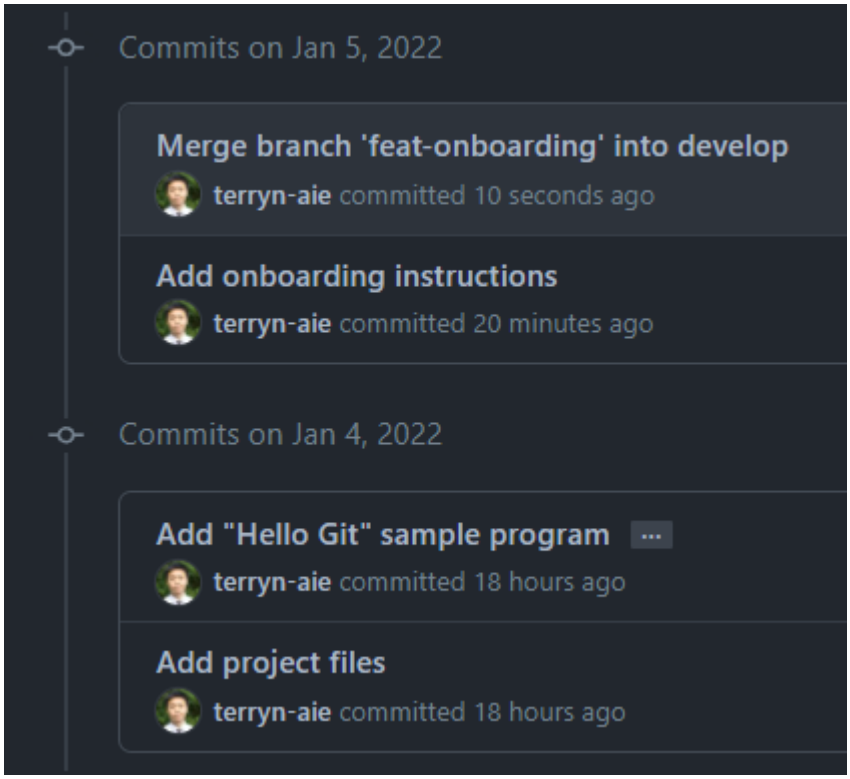
Merging Branches in GitKraken

Since there is one new commit on "feat-onboarding" that is not in "develop", this commit will be added to the "develop" branch, making it fully up-to-date with all of those changes.

Now that "develop" is up-to-date, we can push it to GitHub so the next person who needs to merge their work can pull to have an accurate understanding of what work has been integrated.

If we review our repository on GitHub, we'll see that **develop** has one or two new commits (depending on your settings) in addition to the commits that we originally created in the **main** branch.

develop branch



The next time you or someone else tries to push their changes to **develop**, they will also have to merge these changes in, adding their work on top of yours. By frequently ensuring your branch is up-to-date and on top of any changes that are landing in **develop**, you can regularly integrate your work with the latest changes to the project.

When **develop** is polished enough for a release, we repeat this same process with the **master** branch as well, contributing changes from **develop** into **master**.

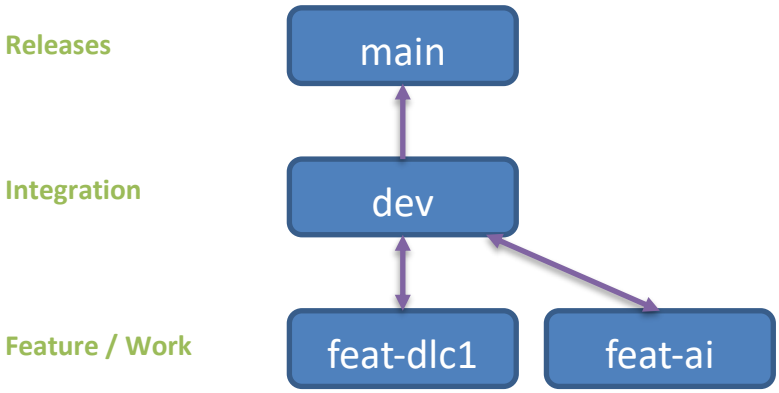
main branch

If you were releasing a project, you would create a build of your project and tag this release in Git to track which version of your project was created from which commit in Git.

Summary

With this, we've demonstrated the complete Git Flow workflow:

1. Branch off from **develop** to create a **feature** branch
2. Work directly in the **feature** branch on your features/bug fixes/misc. changes
3. When work is ready to be shared, merge it into the **develop** branch
    a. Merge **develop** into your **feature** branch first if you are not up-to-date
4. When **develop** is ready for release, merge it into the **main** branch



Git Flow – three layers of branches

This structure maintains a balance between favouring stability (in the **main** and **dev** branches) vs. prioritizing ease of development (in the **feature** branches).

The next team (or company) you work with may not necessarily follow this approach exactly, but the concepts we've introduced here: branching, merging, integrating work, will apply to any configuration you encounter in the future.