Git Terms

Repository / "Repo"

A repository is the what Git uses to manage projects. Every version of every file, every change, every diff, every reference log, every blob, is all stored in the repository. You can think of it as the thing that contains the entire project.

Some projects put all of their project content into a singular repository making one massive "monorepo". This is not really appropriate for game development since it's more ideal to separate out the art source content from the in-game content. For BUGJam we will have two repos: One for art source content, and one for in-game content.

Origin

Each of the project repositories are stored in the software forge on the BUGJam server at https://git.bugjam.dev/ Origin refers to the version of the repository that is hosted on the server. Since it's not stored on your computer, we call this a "remote" repository. Origin is the "source of truth" for the project.

Clone

Nearly identical to "downloading", cloning is a function of Git that can reach out to a remote repository and copy the files down to your computer. With one very important difference: It keeps a connection to that remote repository so that you can track your changes and push them back up to the remote repository.

History

A full list of every change that has happened within the repository. Every file that was checked in, every modification, addition and deletion is stored in the project's history.

Branch

Git is able to track multiple timelines of history for the project. Whenever you make a split in that history, it's called a branch. The primary branch is called 'main'. The main branch is meant to represent a stable state of the project, you should always be able to view the main branch and play a version of the game that's free from crashes or major issues. We do not develop directly in the main branch. Other branches will contain the actively developed new features, this ensures that each contributor can add content to the game without breaking the content that other people are working on at the same time.

Local Changes

Any change you make to the files within the repository will automatically be detected by Git: Creating a file, editing a file, moving a file, deleting a file. These will all show up in your local changes. These changes will not immediately show up for other users. Local changes are volatile, they aren't checked into the version control system yet, so be careful, if you revert your local changes without committing, they are lost forever.

Diff (Difference)

Whenever a change is made, Git isn't just tracking the change that happened, it looks through the file to track the difference between the two versions. For text files, it's able to show you line-by-line what changed, this is the "diff".

Stage

After you've made changes to files in the repository, Git will be aware of the differences in the files and they will appear in the local changes. But they still need to be added as new versions in the repository. We have to tell Git which files we want to add, picking which files we want is called "staging" those files.

Commit

Every time you have a change you want to submit to the repository, you need to commit it into Git's history. This represents an actual check-in which saves your work into the repository. Without a commit, your work is in a volatile state, you must commit to make your changes permanent. First stage your files, then write a message, then commit. Note that this still will not make these changes available for the other users yet, since the commit only happens on your local computer.

Commit Message

Sometimes it's unclear what changes took place within a commit, all commits must have a commit message which is usually one or two lines of text describing the changes you're about to check in. Example: "fire extinguisher: Create block mesh". This will make it clear what this commit contains when we view it in the project history.

Commit Hash

Every time you commit it is given a unique identifier in history, this is the hash. You won't need to concern yourself with this unless you're doing very advanced Git operations.

Fetch

Because Git is decentralized, it's unaware of the remote repository on the server for most operations that you run on your local machine. Changes could be happening upstream from you and your local copy of the repository will become out of date. "Fetch" is an action that reaches out to the remote repository to ask for the latest information.

Pull

Where "Fetch" just asks the server for the latest information, "Pull" will take those changes and pull them into your local copy of the branch. This is how you stay up to date. Keep in mind that "Pull" actively makes changes on your machine, if you're not careful it can erase your local data in favor of what's on the server. Don't forget to commit your changes first.

Push

"Push" is the opposite of "Pull", it takes the commits in your local repository, and pushes them up to origin. This is how you share your changes with the rest of the team.

Merge

If multiple branches were used, you can't see the content of both branches at the same time. Eventually you have the "merge" the content of one branch into the other to unify the timeline.

Revision #3
Created 2026-02-14 09:54:51 PST by xury
Updated 2026-02-14 09:55:43 PST by xury