Below are the comprehensive revision notes for the live class on Git concepts and commands, structured coherently for easy understanding:

# Git Revision Notes

## 1. Introduction to Git

Git is a version control system that helps in tracking changes in source code during software development. It's designed for coordinating work among programmers, but it can be used to track changes in any set of files.

## 2. Basic Git Commands

Here are some fundamental commands introduced in the class:

- **`git add`**: This command adds changes in the working directory to the staging area. You can use `git add .` to add all new and modified files to the staging area. Files in the staging area are marked in green when you use `git status`.
- **`git commit`**: This command captures a snapshot of the project's currently staged changes. You can commit with a message using `git commit -m "Your commit message"` 【8:0†source】.
- **`git status`**: This command shows the state of the working directory and the staging area. It tells you what changes have been staged, which files aren't being tracked by Git, and which files haven't been staged.

## 3. Understanding Staging Area and Commits

- **Staging Area:** A place where Git collates changes before they're committed. Files added to this area will appear in the `git status`

- **Commits**: These are snapshots of your repository at a given point in time. Commits hold information about file changes and states. You can view previous commits with `git log`, which provides a list of commits including their IDs, authors, dates, and messages 【8:17†source】.

## 4. Working with Branches

- **Branch Creation and Checkout:** Use `git branch <branch-name>` to create a new branch, and `git checkout <branch-name>` to switch to it. The command `git checkout -b <branch-name>` is a shortcut to create and switch to a new branch 【8:18†source】.
- **Tracking Changes Across Branches:** When you create a new branch, it takes a snapshot from the point in the original branch. However, future changes in the original won't reflect in the new one unless merged 【8:9†source】.

## 5. Resetting Changes

Git provides different reset types to handle committed and uncommitted changes:

- **Soft Reset**: Moves the commit history pointer without changing the index file. It allows you to include more changes in your last commit by bringing fields from the previous commit back to the staging area 【8:1†source】【8:6†source】.
- **Mixed Reset**: Resets index to match a previous commit but leaves the working directory as it is. It's useful to unstage changes but keep them in your working directory 【8:5†source】【8:6†source】.
- **Hard Reset**: The head and the project state both move to the specified commit, discarding all changes after it. This is useful for getting rid of unwanted changes 【8:19†source】.

Merge conflicts occur when changes in different branches clash with each other. They must be resolved manually:

- **Conflict Indicators:** These are often depicted between `<<<<<<`, `======`, and `>>>>>>` in conflict markers left in your files 【8:2†source】.
- **Manual Resolution:** Open the conflicted file, decide which changes should be kept, and remove the conflict indicators before saving the file. Then add (`git add .`) and commit these resolved changes 【8:5†source】.

## 7. Advanced Concepts: Rebase and Merge

- **Merging**: It integrates changes from different branches into a single branch, maintaining the commit history. To execute a merge, use `git merge <branch-name>` 【8:5†source】.
- **Rebasing**: It allows you to reapply commits on top of another base tip. This can rework your commit history to a linear path, which is often more readable 【8:8†source】【8:13†source】.

## 8. Pull Requests and Code Review

- **Pull Requests (PRs):** A method of submitting contributions to a project. It is handled outside of Git, often using platforms like GitHub 【8:12†source】【8:16†source】.

With these notes, learners should have a solid understanding of the foundational Git commands and workflows as covered in the class.