Revision Notes for Git Basics Class

Introduction

This class serves as an introduction to Git, a popular Version Control System (VCS) used widely in the software industry. The primary focus is on understanding basic Git concepts such as branches, commits, merge, and rebase. These concepts are vital for managing code changes efficiently in software projects.

Version Control Systems (VCS)

What is VCS?

VCS is a system that helps in managing changes to source code over time. It allows developers to:

- Track changes
- Revert to previous versions of the code if necessary
- Collaborate with others on the same project
- Manage contributions from multiple developers

Why Use VCS?

1. **Undo Mistakes**: Easily revert back to a previous stable state if a deployment fails.
2. **Track Changes**: View a history of who changed what, potentially saving significant time in diagnosing issues.
3. **Backup**: Acts as a backup system for the source code.

Types of VCS

1. **Centralized Version Control Systems (CVCS)**: All versions are stored in a single central server. Example: Subversion (SVN).
2. **Distributed Version Control Systems (DVCS)**: Every contributor has a local copy of the entire version history. Example: Git.

Git Basics

Commits

- A **commit** is a snapshot of changes in the project at a particular point in time【6:0†source】.
- Commits contain metadata including the author, committer, date, and a message describing the change.

Branches

- A **branch** in Git is essentially a pointer to a commit. It lets you work on different versions of your codebase simultaneously【6:9†source】.
- The **main branch** (or master) typically holds the production-ready code【6:19†source】.

Merging

- **Merge** is a command to integrate changes from another branch into the current branch.
- It involves a "merge commit" if the branches have diverged, creating a commit with two parents【6:18†source】.

Rebasing

- **Rebase** is a way to integrate changes from one branch onto another. It "replays" your changes on the new base commit.
- Although it can clean up the commit history, it should be used cautiously on shared branches【6:1†source】.

- This state occurs when your HEAD is not pointing to the latest commit on a branch. Instead, it's pointing directly to a specific commit【6:0†source】.

Analogy

An analogy used in class described the linked structure of commits as similar to a linked list, where commits point to previous ones. This helps visualize the history traversal but with the capability of multiple branches like a tree【6:8†source】.

Practical Commands

1. `git checkout <branch-name>` : Switch to the specified branch.
2. `git merge <branch-name>` : Merge the specified branch into the current branch.
3. `git rebase <branch>` : Rebase the current branch onto the specified branch.
4. `git log` : View the commit history【6:16†source】.

Conclusion

This introductory class covered the fundamental aspects of Git necessary to manage a local Git repository. Understanding these core concepts will enable developers to collaborate more effectively and maintain code integrity over time. Further classes will explore advanced topics such as remote branches and collaboration workflows.

These notes encapsulate the key points from the class. Any further questions or topics like pull requests and remote branches will be addressed in subsequent classes.