Advanced Git Commands

1 Introduction

Git provides many powerful commands beyond the commonly used ones like `commit`, `push`, and `merge`. This document covers nine lesser-known but useful Git commands, including their syntax, use cases, and examples.

2 git worktree

Description: Allows multiple working directories from a single repository.
Syntax:

```
git worktree add <path> <branch>
```

Use Cases:

- Work on multiple branches simultaneously.
- Maintain separate environments for testing.

Example:

```
git worktree add feature-work feature-branch
cd feature-work
```

3 git bisect

Description: Finds the commit that introduced a bug via binary search.
Syntax:

```
git bisect start
git bisect good <commit>
git bisect bad <commit>
git bisect reset
```

4 git rerere

Description: Remembers conflict resolutions to reuse them later.
Syntax:

```
git config --global rerere.enabled true
git merge <branch>
```

5 git reflog

Description: Tracks all changes to the branch references.
Syntax:

```
git reflog
git checkout HEAD@{n}
```

6 git replace

Description: Temporarily substitutes one commit for another.
Syntax:

```
git replace <old-commit> <new-commit>
```

7 git cherry-pick

Description: Applies a specific commit from another branch.
Syntax:

```
git cherry-pick <commit>
```

8 git submodule

Description: Manages repositories inside repositories.
Syntax:

```
git submodule add <repo-url> <path>
```

Use Cases:

- Include external libraries inside a repository.
- Track dependencies separately from the main project.

9 git blame

Description: Shows who made changes to each line in a file.
Syntax:

```
git blame <file>
```

Use Cases:

- Identify who introduced a particular change.
- Debugging code history.

10 git bundle

Description: Creates a single file containing Git repository data.
Syntax:

```
git bundle create <file> <branch>
```

Use Cases:

- Share repositories without direct access.
- Archive a repository snapshot.

11 Conclusion

These advanced Git commands can enhance productivity, improve debugging, and streamline workflows. Mastering them can help developers manage complex Git projects more efficiently.