Git for Quants

Amit Kumar Jha

Lets create small notes with all important git commands for quants from basic to advanced with definition , when to use with example command and results as well.

A guide with important Git commands can be very handy for quants managing codebases for mathematical models and financial analytics. Here's a comprehensive guide covering fundamental and advanced commands, when to use them, and examples:

Basic Commands

1. **Initialize Repository**
   Command: `` `git init` ``
   When to Use: Starting a new local repository
   Example:

   ```bash
   git init
   ```

   Result: Creates an empty `` `.git` `` directory indicating a new repository.

2. **Clone Repository**
   Command: `` `git clone <url>` ``
   When to Use: Copy an existing repository to your local machine
   Example:

   ```bash
   git clone https://github.com/username/repo.git
   ```

   Result: Creates a local copy of the repository.

3. **Stage Changes**
   Command: `` `git add <file>` `` or `` `git add .` ``

When to Use: Preparing changes to commit
Example:

```bash
git add script.py
git add .
```

Result: Stages `script.py` or all changed files.

4. **Commit Changes**
   Command: `git commit -m "message"`
   When to Use: Save changes in the local repository
   Example:

```bash
git commit -m "Initial commit"
```

Result: Commits staged changes with a specified message.

5. **View Log**
   Command: `git log`
   When to Use: See the history of commits
   Example:

```bash
git log
```

Result: Shows commit history with messages and hash codes.

Intermediate Commands

1. **Branching**
   Command: `git branch <branch-name>`
   When to Use: Create a new branch for isolated development
   Example:

```bash
git branch new-feature
```

Result: Creates a new branch named `new-feature`.

2. **Switch Branch**
   Command: `` `git checkout <branch-name>` `` or `` `git switch <branch-name>` ``
   When to Use: Switch between branches
   Example:

```bash
git checkout new-feature
git switch new-feature
```

   Result: Changes to the specified branch.

3. **Merge Branches**
   Command: `` `git merge <branch-name>` ``
   When to Use: Combine changes from one branch into the current branch
   Example:

```bash
git checkout main
git merge new-feature
```

   Result: Merges `` `new-feature` `` into `` `main` ``.

4. **Stash Changes**
   Command: `` `git stash` ``
   When to Use: Save changes temporarily for later use
   Example:

```bash
git stash
```

   Result: Saves current changes and resets the working directory.

5. **Apply Stashed Changes**
   Command: `` `git stash apply` ``
   When to Use: Restore changes previously stashed
   Example:

```bash
git stash apply
```

   Result: Applies the latest stashed changes.

Advanced Commands

1. **Rebase Branch**
   Command: `` `git rebase <branch-name>` ``
   When to Use: Re-apply commits on top of another base
   Example:

   ```bash
   git checkout new-feature
   git rebase main
   ```

   Result: Moves the `` `new-feature` `` branch commits to the `` `main` `` branch's tip.

2. **Interactive Rebase**
   Command: `` `git rebase -i <commit-hash>` ``
   When to Use: Squash or edit commits interactively
   Example:

   ```bash
   git rebase -i HEAD~3
   ```

   Result: Opens an editor to rework the last 3 commits.

3. **Cherry-Pick Commit**
   Command: `` `git cherry-pick <commit-hash>` ``
   When to Use: Apply a specific commit to the current branch
   Example:

   ```bash
   git cherry-pick d1e34f7
   ```

   Result: Applies the changes from commit `` `d1e34f7` ``.

4. **Reset to Commit**
   Command: `` `git reset --hard <commit-hash>` ``
   When to Use: Move the branch pointer back to a previous commit
   Example:

   ```bash
   git reset --hard d1e34f7
   ```

Result: Resets the current branch to commit `**d1e34f7**`.

5. **Fetch and Rebase**
   Command:

```bash
git fetch origin
git rebase origin/main
```

   When to Use: Update your local branch with the latest changes from the remote
   Result: Updates your branch to reflect changes in the remote `**main**`.

6. **Bisect**
   Command: `**git bisect start**`
   When to Use: Isolate a problematic commit by binary search
   Example:

```bash
git bisect start
git bisect bad HEAD
git bisect good <known-good-commit>
```

   Result: Identifies the first commit that introduced a bug.

These commands cover a range of common workflows that quants might use. Adjust them based on your team's specific branching strategy and development environment.