Advanced Git and some other shell programs
Not so advanced

Pratham Lalwani

University of California, Merced, Department of Applied Math

April 22, 2025

Outline

Review Git Basics: Introduction

In this section we review the core Git commands and operations that every developer should know.

Review Git Basics: Introduction

In this section we review the core Git commands and operations
that every developer should know.

- `git init`: Initialize a new repository.
- `git add`: Stage changes for commit.
- `git commit`: Commit the staged changes.
- `git push`: Push the commits to a remote repository.

Git Workflow Example

Example Workflow:

```
1    mkdir movies
2    cd movies
3    git init
4    echo "#␣My␣Movies" > README.md
5    git add .
6    git commit -m "Initial␣commit"
7    git remote add origin https://github.com/username/
         my_project.git
8    git push -u origin master
```

Enhancing Your Shell Workflow

In this section, we cover some handy Bash tools to improve productivity:

- `tldr`: Concise command help.
- `fzf`: Fuzzy finder for quick navigation.
- `grep`: Powerful text search.
- `shopt -s cdable_vars`: Speed up directory navigation.
- `find`: Locate files by pattern.

Bash Commands in Action

`tldr` **and** `fzf`:

```
1    tldr tar
2    fzf
```

Custom Alias Example:

```
1    shopt -s cdable_vars
2    proj='cd ~/projects'
```

Bash Commands in Action

tldr **and** fzf:

```
1    tldr tar
2    fzf
```

Custom Alias Example:

```
1    shopt -s cdable_vars
2    proj='cd ~/projects'
```

Try these commands in your terminal after the demonstration.

Combining Commands in Scripts

Using grep **and** find:

```
1    find . -name "*.log" | xargs grep -i "error"
```

This command searches for "error" in all .log files recursively in the current directory.

Combining Commands in Scripts

Using grep **and** find:

```
1    find . -name "*.log" | xargs grep -i "error"
```

This command searches for "error" in all .log files recursively in
the current directory.

Advanced Git: Exploring History

Git Log Options:

- `git log --oneline` for a compact view.
- `git log --graph --decorate` to see branch structure.

```
1    git log --oneline --graph --decorate
```

Advanced Git: Exploring History

Git Log Options:

- git log --oneline for a compact view.
- git log --graph --decorate to see branch structure.

```
1    git log --oneline --graph --decorate
```

Branching and Merging Strategies

Key Concepts:

- **Branches:** Create feature branches using `git checkout -b`.
- **Merging:** Use `git merge` to integrate changes.
- **Conflict Resolution:** Use diff and merge tools to resolve conflicts.

Example:

```
1   git switch new-feature
2   # Make changes and commit
3   git switch main
4   git merge new-feature
```

Cleaning Up History: Squash and Revert

Squash Commits:

```
1    git rebase -i
```

Use interactive rebase to combine several commits into one for a cleaner history.

Reverting Changes:

```
1    git revert <commit-hash>
```

This command safely undoes a specific commit without rewriting history.

Cleaning Up History: Squash and Revert

Squash Commits:

```
1    git rebase -i
```

Use interactive rebase to combine several commits into one for a cleaner history.

Reverting Changes:

```
1    git revert <commit-hash>
```

This command safely undoes a specific commit without rewriting history.

Discussion: When is it more appropriate to revert versus perform a rebase?

Contributing to Open Source: Getting Started

Key Steps:

1. **Forking:** Create your own fork of the repository.
2. **Cloning:** Clone your fork locally using `git clone`.
3. **Feature Branch:** Create a branch for your changes.
4. **Making Changes:** Code improvements, fixes, or documentation edits.
5. **Pull Request:** Submit your changes for review via a PR.

Contributing to Open Source: Getting Started

Key Steps:

1. **Forking:** Create your own fork of the repository.
2. **Cloning:** Clone your fork locally using `git clone`.
3. **Feature Branch:** Create a branch for your changes.
4. **Making Changes:** Code improvements, fixes, or documentation edits.
5. **Pull Request:** Submit your changes for review via a PR.

Detailed Contribution Workflow

Commands:

```
1   # Fork on GitHub, then clone your fork:
2   git clone https://github.com/yourusername/repository
        .git
3   cd repository
4   git checkout -b feature-improvement
5
6   # Make changes, then commit:
7   git add .
8   git commit -m "Improve␣feature␣X"
9
10  # Push your branch:
11  git push origin feature-improvement
```

Then, open GitHub and create a Pull Request. Discuss best practices and code review tips.

Common Pitfalls & Troubleshooting in Git

Tips

- Always backup your work before performing a rebase.
- Use `git status` and `git diff` frequently.

Common Pitfalls & Troubleshooting in Git

Tips

- Always backup your work before performing a rebase.
- Use `git status` and `git diff` frequently.

Recap and Further Resources

- **Review Git Basics:** Core commands and simple workflows.
- **Bash Tips:** Useful commands to boost shell efficiency.
- **Advanced Git:** Viewing history, branching strategies, squashing, and reverting.
- **Open Source:** Contributing workflow from forking to pull requests.

Additional reading:

- Pro Git Book
- Atlassian Git Tutorials
- GNU Bash Manual

Advanced Git Log Visualization

```
1    git log --oneline --graph --decorate --all
```

Use this command to view a complete, branch-inclusive
visualization of the repository history.

Advanced Git Log Visualization

```
1    git log --oneline --graph --decorate --all
```

Use this command to view a complete, branch-inclusive visualization of the repository history.

Q&A: Open the floor for student questions or share additional examples as needed.

Thank You!

Questions?