Advanced Git and Bash Techniques for Software Development

An In-depth Workshop for Undergraduate Students

Your Name

Your Institution, Department of Computer Science

April 16, 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.

Time: 10 minutes (demo and discussion)

Git Workflow Example

Example Workflow:

```
mkdir my_project
cd my_project
git init
echo "Hello, Git!" > README.md
git add README.md
git commit -m "Initial commit"
git remote add origin https://github.com/username/
    my_project.git
git push -u origin master
```

Demonstration: Walk through these commands live or use a recorded terminal session.

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.
- `cd aliases`: Speed up directory navigation.
- `find`: Locate files by pattern.

Time: 15 minutes (interactive session and examples)

Bash Commands in Action

`tldr` and `fzf`:

```
tldr tar
fzf --reverse
```

Custom Alias Example:

```
alias proj='cd ~/projects'
```

Bash Commands in Action

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

```
tldr tar
fzf --reverse
```

Custom Alias Example:

```
alias 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.

Discussion: Share tips on how these commands can be integrated into daily workflows.

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
```

Use these commands to visually inspect the commit history and branch merges.

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 checkout -b feature/new-feature
2  # Make changes and commit
3  git checkout master
4  git merge feature/new-feature
```

Note: Explain fast-forward merges versus recursive merges.

Cleaning Up History: Squash and Revert

Squash Commits:

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

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 HEAD~3
```

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.

Tip: Use GitHub's GUI and command line tools to manage this workflow.

Detailed Contribution Workflow

Commands:

```
# Fork on GitHub, then clone your fork:
git clone https://github.com/yourusername/repository.git
cd repository
git checkout -b feature-improvement

# Make changes, then commit:
git add .
git commit -m "Improve feature X"

# Push your branch:
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.
- Read error messages carefully and use online searches (e.g., Stack Overflow).

Common Pitfalls & Troubleshooting in Git

Tips

- Always backup your work before performing a rebase.
- Use `git status` and `git diff` frequently.
- Read error messages carefully and use online searches (e.g., Stack Overflow).

Interactive: Invite questions on real-world issues students have encountered.

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?

Your Name

your.email@institution.edu

Back to TOC