Git Commands Documentation

Udacity: Programming for Data Science with Python - Git Project

Student Name: Sebastien Henry **GitHub Repository URL:** [Your GitHub Repository URL - To be added after creating remote repository]
Date: December 17, 2025

Section I: Set Up Your Repository

This section documents all Git commands used to initialize the repository and set up version control.

Task	Git Command	Description
1. Initialize a new Git repository	`git init`	Creates a new Git repository in the current directory
2. Rename default branch to main	`git branch -m main`	Renames the default branch from 'master' to 'main'
3. Create .gitignore file	N/A - File created manually	Created .gitignore to exclude CSV files and other unnecessary files
4. Check repository status	`git status`	Displays the state of the working directory and staging area
5. Stage files for initial commit	`git add .gitignore bikeshare.py requirements.txt run_bikeshare.sh readme.txt`	Adds specified files to the staging area
6. Create initial commit	`git commit -m "Initial commit: Add bikeshare project files and .gitignore"`	Creates the first commit with project files
7. Verify repository status	`git status`	Confirms all changes are committed and working tree is clean
8. Create remote repository on GitHub	N/A - Done via GitHub web interface	Create empty repository on GitHub (do not initialize with README)
9. Add remote origin	`git remote add origin <repository-url>`	Links local repository to GitHub remote repository
10. Push to remote repository	`git push -u origin main`	Pushes main branch to remote and sets up tracking

repository URL (e.g., `https://github.com/SebAustin/bikeshare-project.git`)

Section II: Improve Documentation - Create README

This section documents commands used to create a documentation branch and add a comprehensive README file.

Task	Git Command	Description
1. Create and switch to documentation branch	`git checkout -b documentation`	Creates a new branch called 'documentation' and switches to it
2. Verify current branch	`git branch`	Lists all branches and highlights the current branch
3. Create README.md file	N/A - File created manually	Created comprehensive README.md with Markdown formatting
4. Check status of changes	`git status`	Shows README.md as an untracked file
5. Stage README.md	`git add README.md`	Adds README.md to the staging area
6. Commit README changes	`git commit -m "docs: Add comprehensive README.md with project documentation"`	Commits the new README file with descriptive message
7. View commit history	`git log --oneline`	Displays abbreviated commit history
8. Push documentation branch to remote	`git push origin documentation`	Pushes documentation branch to GitHub

---

Section III: Additional Changes to Documentation

This section documents additional documentation improvements made on the documentation branch.

Task	Git Command	Description
1. Verify current branch	`git branch`	Confirms we're on the documentation branch
2. Modify bikeshare.py comments	N/A - File edited manually	Enhanced docstrings and inline comments throughout the code
3. Check changes made	`git diff`	Shows differences between working directory and last commit
4. Stage modified bikeshare.py	`git add bikeshare.py`	Stages the updated file
5. Commit documentation improvements	`git commit -m "docs: Enhance code comments and docstrings for better clarity"`	Commits improved code documentation

6. View branch commit history	`git log --oneline documentation`	Shows commits on documentation branch
7. Push updates to remote	`git push origin documentation`	Pushes latest documentation changes to GitHub

---

Section IV: Refactor Code

This section documents the refactoring branch where code improvements were made with multiple commits.

Task	Git Command	Description
1. Switch back to main branch	`git checkout main`	Returns to the main branch
2. Create and switch to refactoring branch	`git checkout -b refactoring`	Creates refactoring branch from main
3. Verify current branch	`git branch`	Confirms we're on the refactoring branch
First Refactoring Change		
4. Modify code - Extract constants	N/A - Code edited manually	Created constants and helper function for input validation
5. Check changes	`git diff bikeshare.py`	Reviews the changes made to the file
6. Stage changes	`git add bikeshare.py`	Stages the refactored code
7. Commit first refactoring	`git commit -m "refactor: Extract constants and create helper function for input validation"`	First meaningful refactoring commit
Second Refactoring Change		
8. Modify code - Improve efficiency	N/A - Code edited manually	Optimized time calculations and extracted hour once
9. Check changes	`git diff`	Reviews the efficiency improvements
10. Stage changes	`git add bikeshare.py`	Stages the updated code
11. Commit second refactoring	`git commit -m "refactor: Improve efficiency by extracting hour once and creating time conversion helper"`	Second meaningful refactoring commit
Third Refactoring Change		

12. Modify code - Modernize formatting	N/A - Code edited manually	Replaced old string formatting with f-strings
13. Check changes	`git diff`	Reviews the formatting improvements
14. Stage changes	`git add bikeshare.py`	Stages the modernized code
15. Commit third refactoring	`git commit -m "refactor: Modernize string formatting using f-strings for better readability"`	Third meaningful refactoring commit
16. View refactoring branch history	`git log --oneline refactoring`	Shows all commits on refactoring branch
17. Push refactoring branch to remote	`git push origin refactoring`	Pushes refactoring branch to GitHub

---

Section V: Merging Branches

This section documents the process of merging both documentation and refactoring branches into main.

Task	Git Command	Description
1. Switch to main branch	`git checkout main`	Returns to main branch to prepare for merging
2. Verify current branch	`git branch`	Confirms we're on main branch
3. View all branches	`git branch -a`	Lists all local and remote branches
4. Merge documentation branch	`git merge documentation -m "Merge documentation branch: Add README and improve code comments"`	Merges documentation branch into main (fast-forward)
5. View merge result	`git log --oneline --graph`	Shows merge in commit history
6. Merge refactoring branch	`git merge refactoring -m "Merge refactoring branch: Improve code structure and efficiency"`	Attempts to merge refactoring branch
7. Check merge status	`git status`	Shows merge conflict in bikeshare.py
8. View conflict in file	`git diff` or manually open file	Identifies conflicting sections
9. Resolve merge conflicts	N/A - File edited manually	Manually resolved conflicts in bikeshare.py
10. Stage resolved file	`git add bikeshare.py`	Marks conflict as resolved

11. Complete merge commit	`git commit -m "Merge refactoring branch: Improve code structure and efficiency"`	Completes the merge commit
12. View complete history	`git log --oneline --graph --all --decorate`	Shows full branch and merge history
13. Push merged changes to remote	`git push origin main`	Pushes all merged changes to GitHub
14. Verify remote branches	`git branch -r`	Lists remote branches
15. View final repository state	`git status`	Confirms clean working directory

---

Additional Useful Git Commands

These commands were helpful throughout the project:

Command	Purpose
`git log --oneline --graph --all`	Visualize branch structure and history
`git diff <branch1> <branch2>`	Compare differences between branches
`git branch -d <branch-name>`	Delete a local branch (after merging)
`git remote -v`	View remote repository URLs
`git fetch origin`	Download changes from remote without merging
`git pull origin main`	Fetch and merge changes from remote main branch

Summary of Repository Structure

Branches Created:

- **main**: Primary branch containing all merged changes
- **documentation**: Branch for README and code documentation improvements
- **refactoring**: Branch for code structure and efficiency improvements

Total Commits: 8

1. Initial commit (main)
2. Add README.md (documentation)
3. Enhance code comments (documentation)
4. Extract constants and helper function (refactoring)
5. Improve efficiency (refactoring)
6. Modernize string formatting (refactoring)
7. Merge documentation branch (main)
8. Merge refactoring branch (main)

Files in Repository:

- `bikeshare.py` - Main Python script
- `README.md` - Project documentation
- `.gitignore` - Git ignore rules (excludes CSV files)
- `requirements.txt` - Python dependencies
- `run_bikeshare.sh` - Shell script to run the application
- `readme.txt` - Original readme file

Files Excluded (via .gitignore):

- `*.csv` - All CSV data files
- `__pycache__/` - Python cache
- `venv/` - Virtual environment
- `.DS_Store` - macOS system files

Key Learnings

1. **Branching Strategy**: Using separate branches for documentation and refactoring allowed parallel development without conflicts in most areas.

2. **Meaningful Commit Messages**: Used conventional commit prefixes (docs:, refactor:) to clearly indicate the purpose of each commit.

3. **Merge Conflicts**: Successfully resolved merge conflicts when both branches modified the same file.

4. **.gitignore Importance**: Prevented large CSV data files from being tracked in version control, keeping the repository clean and efficient.

5. **Git Workflow**: Followed a professional workflow: create branch → make changes → commit → merge → push.

Notes for Submission

- This document captures all Git commands used throughout the project
- The repository demonstrates proper use of branching, committing, and merging
- All CSV files are properly excluded from version control
- Multiple meaningful commits were made on each branch
- Branches were successfully merged into main with conflict resolution

Remember to:

1. Create your GitHub repository
2. Add the remote origin URL
3. Push all branches to GitHub
4. Update the GitHub Repository URL at the top of this document
5. Convert this document to PDF for submission