Git Challenge Documentation

Progressive Git Learning Exercise - Complete Command Guide

Task #1: I Can Win

Objective: Basic Git setup, repository creation, and GitHub integration

Step 1: Install Git and Generate SSH Keys

bash

```bash
# Install git (if not already installed)
# On Windows: Download from git-scm.com
# On Linux: sudo apt install git
# On macOS: brew install git

# Generate SSH key pair
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"

# Start SSH agent and add key
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_rsa

# Display public key to copy to GitHub
cat ~/.ssh/id_rsa.pub
```

Description: Install Git and create SSH authentication keys for secure GitHub access.

Step 2: Configure Git User Information

bash

```bash
# Set your name and email globally
git config --global user.name "Your Name"
git config --global user.email "your_email@example.com"

# Verify configuration
git config --list
```

Description: Configure Git with your identity information for commit attribution.

Step 3: Create Repository and Clone Locally

bash

```
# After creating repository on GitHub via web interface
git clone git@github.com:yourusername/your-repo-name.git
cd your-repo-name
```

Description: Create a new repository on GitHub and clone it to your local machine.

Step 4: Create Song File with First Half

bash

```
# Create song.txt with your favorite song's first half
echo "First line of your favorite song" > song.txt
echo "Second line of your favorite song" >> song.txt
echo "Third line of your favorite song" >> song.txt
```

Description: Create a text file containing the first half of your favorite song lyrics.

Step 5: Make First Commit and Push

bash

```
# Add file to staging area
git add song.txt

# Create commit with specified message
git commit -m "add first half of my favorite song"

# Push to GitHub
git push origin main
```

Description: Stage, commit, and push the first half of the song to the remote repository.

Step 6: Verify File on GitHub

Description: Check GitHub web interface to confirm `song.txt` file exists with the lyrics.

Step 7: Add Second Half via GitHub Web Interface

Description: Use GitHub's web editor to add the remaining lyrics and commit with message "finish my song".

Step 8: Pull Changes Locally

```bash
# Pull the commit made on GitHub
git pull origin main

# Verify you have complete lyrics
cat song.txt
```

Description: Synchronize local repository with GitHub changes and verify complete song lyrics.

Task #2: Bring It On

Objective: Branching, merging, gitignore, and conflict resolution

Step 1: Create .gitignore File

```bash
# Create .gitignore file
echo "*.db" > .gitignore
echo "*.log" >> .gitignore
echo "target/" >> .gitignore
echo "bin/" >> .gitignore

# Add and commit .gitignore
git add .gitignore
git commit -m "add gitignore file"
```

Description: Create .gitignore to exclude database files, logs, and build directories.

Step 2: Create Feature Branch and Add Commits

bash

```bash
# Create and switch to feature branch
git checkout -b feature

# Create first commit
echo "Feature content 1" > feature1.txt
git add feature1.txt
git commit -m "add feature1"

# Create second commit
echo "Feature content 2" > feature2.txt
git add feature2.txt
git commit -m "add feature2"
```

Description: Create a feature branch and add two commits to demonstrate branching workflow.

Step 3: Merge Feature Branch to Master

bash

```bash
# Switch to master and merge
git checkout main
git merge feature
```

Description: Merge the feature branch into the main branch using fast-forward merge.

Step 4: Create arrows.txt on Feature Branch

bash

```bash
# Switch back to feature branch
git checkout feature

# Create arrows.txt with specified content
echo "The ship glides gently on the waves" > arrows.txt
echo "As day turns into night" >> arrows.txt

# Commit the file
git add arrows.txt
git commit -m "add arrows.txt to feature branch"
```

Description: Create conflicting content on the feature branch to demonstrate merge conflicts.

Step 5: Create arrows.txt on Master Branch

bash

```
# Switch to master
git checkout main

# Create arrows.txt with different content
echo "One thousand burning arrows" > arrows.txt
echo "Fill the starlit sky" >> arrows.txt

# Commit the file
git add arrows.txt
git commit -m "add arrows.txt to master branch"
```

Description: Create the same filename with different content on master to create a merge conflict.

Step 6: Merge with Conflict Resolution

bash

```
# Attempt to merge feature into master
git merge feature

# Git will show conflict - manually edit arrows.txt to contain:
# The ship glides gently on the waves
# As day turns into night
# One thousand burning arrows
# Fill the starlit sky

# Resolve conflict by editing the file
echo "The ship glides gently on the waves" > arrows.txt
echo "As day turns into night" >> arrows.txt
echo "One thousand burning arrows" >> arrows.txt
echo "Fill the starlit sky" >> arrows.txt

# Complete the merge
git add arrows.txt
git commit -m "resolve merge conflict in arrows.txt"
```

Description: Resolve merge conflict by combining both versions of arrows.txt in chronological order.

Task #3: Hurt Me Plenty

Objective: Advanced branching, tagging, and rebasing

Step 1: Create Storm Branch and First Commit

bash

```
# Create and switch to storm branch
git checkout -b storm

# Create storm.txt with first content
echo "Twenty ships with Norsemen braves" > storm.txt
echo "Riding the northern wind" >> storm.txt

# Commit the changes
git add storm.txt
git commit -m "add first verse to storm.txt"
```

Description: Create a new branch for storm-related content and add initial verse.

Step 2: Add More Lines to storm.txt

bash

```
# Add additional lines to storm.txt
echo "They left their shores at early dawn" >> storm.txt
echo "As a red sun was rising in the east" >> storm.txt

# Commit the additional content
git add storm.txt
git commit -m "add second verse to storm.txt"
```

Description: Extend the storm.txt file with additional verses and create another commit.

Step 3: Create pursuit.txt on Master

bash

```bash
# Switch back to master
git checkout main

# Create pursuit.txt with specified content
echo "The warming sun returns again" > pursuit.txt
echo "And melts away the snow" >> pursuit.txt
echo "The sea is freed from icy chains" >> pursuit.txt
echo "Winter is letting go" >> pursuit.txt

# Commit the new file
git add pursuit.txt
git commit -m "add pursuit.txt with winter theme"
```

Description: Create a new file on master branch with winter/spring themed content.

Step 4: Tag the Commit and Switch to Storm

bash

```bash
# Tag the current commit
git tag session1

# Verify tag was created
git tag --list

# Switch back to storm branch
git checkout storm
```

Description: Create a tag to mark an important milestone in the project's development.

Step 5: Rebase Storm Branch

bash

```bash
# Rebase storm branch onto master
git rebase main

# If conflicts occur, resolve them and continue:
# git add <resolved-files>
# git rebase --continue
```

Description: Rebase storm branch to include the latest changes from master, maintaining a linear history.

Task #4: Hardcore

Objective: Remote repository management and multiple remotes

Step 1: Push Repository to GitHub

bash

```
# Push all branches and tags
git push origin main
git push origin storm
git push origin feature
git push --tags
```

Description: Ensure all local commits, branches, and tags are synchronized with GitHub.

Step 2: Create New Repository on GitHub

Description: Use GitHub web interface to create a second repository (e.g., "your-repo-backup").

Step 3: Change Remote Repository

bash

```
# View current remotes
git remote -v

# Change origin to point to new repository
git remote set-url origin git@github.com:yourusername/your-repo-backup.git

# Verify the change
git remote -v
```

Description: Reconfigure the local repository to point to the new remote repository.

Step 4: Push to New Repository

bash

```
# Push all content to new repository
git push origin main
git push origin storm
git push origin feature
git push --tags
```

Description: Transfer all repository content to the new remote repository.

Step 5: Restore Original Remote

bash

```bash
# Change remote back to original repository
git remote set-url origin git@github.com:yourusername/your-original-repo.git

# Verify the restoration
git remote -v

# Test connectivity
git fetch origin
```

Description: Restore the original remote configuration to maintain normal workflow.

Task #5: Nightmare! (Command Line Only)

Objective: Complete all previous tasks using only command line tools

Essential Commands for Console-Only Operations

File Operations

bash

```bash
# Display file contents
cat filename.txt

# Create/append to files
echo "content" > filename.txt    # overwrite
echo "content" >> filename.txt   # append

# Navigate directories
cd directory_name
pwd                              # show current directory
ls -la                          # list files with details
```

Text Editing with vi

bash

```bash
# Open file in vi editor
vi filename.txt

# Vi commands:
# i           - enter insert mode
# Esc         - exit insert mode
# :w          - save file
# :q          - quit
# :wq         - save and quit
# :q!         - quit without saving
# dd          - delete current line
# yy          - copy current line
# p           - paste
```

Command-Line Execution of All Tasks

Task 1 - Console Version:

bash

```bash
# SSH key generation (same as GUI version)
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
cat ~/.ssh/id_rsa.pub

# Git configuration
git config --global user.name "Your Name"
git config --global user.email "your_email@example.com"

# Repository operations
git clone git@github.com:yourusername/repo.git
cd repo

# Create song file using echo or vi
echo "First half of song lyrics" > song.txt
echo "More lyrics here" >> song.txt
# OR
vi song.txt  # then use vi commands to add content

# Git operations
git add song.txt
git commit -m "add first half of my favorite song"
git push origin main
git pull origin main
cat song.txt  # verify complete content
```

Task 2 - Console Version:

bash

```bash
# Create .gitignore
echo "*.db" > .gitignore
echo "*.log" >> .gitignore
echo "target/" >> .gitignore
echo "bin/" >> .gitignore

# Branch operations
git checkout -b feature
echo "content1" > file1.txt
git add file1.txt
git commit -m "commit1"

# Continue with all branching and merging operations using same commands
# Use vi for conflict resolution instead of GUI editor
```

Tasks 3-4 - Console Version:

bash

```bash
# Use same git commands as previous tasks
# Replace any GUI text editing with vi editor
# Replace file browsing with ls, cd, cat commands
# All git operations remain identical to GUI versions
```

Key Differences for Console-Only Approach

- **File Creation:** Use `echo` commands or `vi` editor instead of GUI text editors
- **File Viewing:** Use `cat`, `less`, or `vi` instead of GUI file managers
- **Directory Navigation:** Use `cd`, `pwd`, `ls` commands exclusively
- **Conflict Resolution:** Use `vi` to manually edit conflicted files
- **Verification:** Use `cat` and `git log` to verify file contents and commit history

Success Criteria for Nightmare Mode

1. Complete all tasks 1-4 using only MINGW64 terminal
2. No GUI applications except web browser for GitHub operations
3. All file operations performed via command line
4. Demonstrate proficiency with vi editor for text editing

5. Successfully navigate and manage repository using console commands only

---

Summary

This progressive Git challenge develops skills from basic repository management to advanced command-line proficiency. Each task builds upon previous knowledge while introducing new Git concepts:

- **Task 1:** Fundamentals of Git and GitHub
- **Task 2:** Branching and conflict resolution
- **Task 3:** Advanced workflows with rebasing and tagging
- **Task 4:** Remote repository management
- **Task 5:** Complete command-line mastery

Successfully completing all tasks demonstrates comprehensive Git proficiency suitable for professional development environments.