Git Hands-On Lab Solutions

Lab 1: Git Setup and Basic Operations

Objectives

- Setup Git configuration
- Integrate Notepad++ as default editor
- Add files to repository

Solution Steps

Step 1: Setup Git Configuration

bash

```bash
# Check Git installation
git --version

# Configure user credentials
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"

# Verify configuration
git config --list
```

Step 2: Integrate Notepad++ as Default Editor

bash

```
# Test if notepad++ is accessible
notepad++

# If not accessible, add to PATH environment variable
# Then create alias (add to ~/.bashrc or ~/.bash_profile)
alias npp='notepad++'

# Configure Git to use notepad++ as default editor
git config --global core.editor "notepad++.exe -multiInst -notabbar -nosession -noPlugin"

# Verify editor configuration
git config --global -e
```

Step 3: Add File to Repository

```bash
```

```
# Create new directory and initialize Git repository
mkdir GitDemo
cd GitDemo
git init

# Verify initialization (shows hidden .git folder)
ls -la

# Create and add content to welcome.txt
echo "Welcome to Git Demo" > welcome.txt

# Verify file creation
ls -l
cat welcome.txt

# Check Git status
git status

# Add file to staging area
git add welcome.txt

# Commit with message (opens notepad++ for multi-line comments)
git commit

# Check status after commit
git status

# Connect to remote repository (GitLab)
git remote add origin https://gitlab.com/yourusername/GitDemo.git

# Pull from remote (if repository exists)
git pull origin master

# Push to remote repository
git push origin master
```

Lab 2: Git Ignore Implementation

Objectives

- Learn to ignore unwanted files using .gitignore
- Ignore .log files and log folders

Solution Steps

bash

```
# Navigate to your Git repository
cd GitDemo

# Create .log files and log folder
echo "This is a log file" > debug.log
echo "Another log file" > error.log
mkdir logs
echo "Log content" > logs/application.log

# Check current status (should show untracked files)
git status

# Create .gitignore file
notepad++ .gitignore

# Add the following content to .gitignore:
# *.log
# logs/

# Alternatively, create .gitignore from command line
echo "*.log" > .gitignore
echo "logs/" >> .gitignore

# Verify .gitignore content
cat .gitignore

# Check Git status (should ignore .log files and logs folder)
git status

# Add and commit .gitignore
git add .gitignore
git commit -m "Add .gitignore to ignore log files and folders"

# Verify status shows clean working directory
git status

# Push changes to remote
git push origin master
```

Lab 3: Branching and Merging

Objectives

- Create branches and merge with master
- Use P4Merge tool for visual differences

Solution Steps

Branching:

```bash
# Create new branch
git branch GitNewBranch

# List all branches (* indicates current branch)
git branch -a

# Switch to new branch
git checkout GitNewBranch
# Or use: git switch GitNewBranch

# Add files with content
echo "Content for new branch" > branch_file.txt
echo "Additional content" > feature.txt

# Add and commit changes
git add .
git commit -m "Add files to GitNewBranch"

# Check status
git status
```

Merging:

```bash
```

```
# Switch back to master
git checkout master

# Show differences between branches (command line)
git diff master GitNewBranch

# Show visual differences using P4Merge
git difftool master GitNewBranch

# Merge branch into master
git merge GitNewBranch

# View log with graph
git log --oneline --graph --decorate

# Delete merged branch
git branch -d GitNewBranch

# Check final status
git status
git branch -a
```

Lab 4: Conflict Resolution

Objectives

- Handle merge conflicts when multiple users modify the same file

Solution Steps

```bash
```

```
# Verify master is clean
git status

# Create new branch
git branch GitWork
git checkout GitWork

# Create hello.xml with content
echo "<hello>World from GitWork branch</hello>" > hello.xml

# Check status and commit
git status
git add hello.xml
git commit -m "Add hello.xml in GitWork branch"

# Switch to master
git checkout master

# Create hello.xml with different content
echo "<hello>World from master branch</hello>" > hello.xml

# Commit to master
git add hello.xml
git commit -m "Add hello.xml in master branch"

# View log with all branches
git log --oneline --graph --decorate --all

# Check differences
git diff master GitWork

# Use P4Merge for visual comparison
git difftool master GitWork

# Attempt to merge (will create conflict)
git merge GitWork

# Git will mark conflict in hello.xml
cat hello.xml
```

```
# Use 3-way merge tool to resolve
git mergetool

# After resolving conflict, commit
git add hello.xml
git commit -m "Resolve merge conflict in hello.xml"

# Add backup files to .gitignore
echo "*.orig" >> .gitignore
git add .gitignore
git commit -m "Update .gitignore for backup files"

# List branches
git branch -a

# Delete merged branch
git branch -d GitWork

# View final log
git log --oneline --graph --decorate
```

Lab 5: Cleanup and Push to Remote

Objectives

- Clean up local repository and push changes to remote Git

Solution Steps

```
bash
```

```
# Verify master is in clean state
git status

# List all available branches
git branch -a

# Pull latest changes from remote repository
git pull origin master

# Push all pending changes from previous labs to remote
git push origin master

# Verify changes are reflected in remote repository
# Check GitLab web interface to confirm all commits are visible

# Optional: Clean up any remaining branches
git branch -a
# Delete any unwanted local branches
# git branch -d branch_name

# Final status check
git status
git log --oneline --graph --decorate
```

Additional Useful Commands

Setup P4Merge as Merge Tool

```bash
# Configure P4Merge as merge tool
git config --global merge.tool p4merge
git config --global mergetool.p4merge.cmd 'p4merge.exe "$BASE" "$LOCAL" "$REMOTE" "$MERGED"'

# Configure P4Merge as diff tool
git config --global diff.tool p4merge
git config --global difftool.p4merge.cmd 'p4merge.exe "$LOCAL" "$REMOTE"'
```

Common Git Commands Reference

```bash
# Check repository status
git status

# View commit history
git log --oneline --graph --decorate --all

# Create and switch to branch
git checkout -b new-branch-name

# Stage files
git add filename
git add .  # Add all files

# Commit changes
git commit -m "Commit message"

# Push to remote
git push origin branch-name

# Pull from remote
git pull origin branch-name

# Merge branch
git merge branch-name

# Delete branch
git branch -d branch-name
```