Git & GitHub Hands-on Practical Guide

This comprehensive guide covers all essential Git and GitHub operations from initialization to pull requests.

Prerequisites

- Git installed on your machine
- GitHub account created
- Terminal/Command Prompt access

Part 1: Repository Initialization

1.1 Create Local Repository

```
# Create a new directory for your project
mkdir git-practical
cd git-practical

# Initialize Git repository
git init

# Check repository status
git status

# Create initial files
echo "# Git Practical Session" > README.md
echo "console.log('Hello Git!');" > app.js
echo "node_modules/" > .gitignore

# Add files to staging area
git add README.md app.js .gitignore

# Make first commit
git commit -m "Initial commit: Add README, app.js,
        and .gitignore"

# Check commit history
git log --oneline
```

1.2 Connect to GitHub

```
# Create repository on GitHub first, then:
# Replace 'username' with your GitHub username
git remote add origin https://github.com/username/git-
        practical.git

# Push to GitHub
git branch -M main
git push -u origin main

# Verify remote connection
git remote -v
```

Part 2: Branch Management

2.1 Create and Work with Branches

```
# Create and switch to feature branch
git checkout -b feature/user-authentication

# Alternative way (Git 2.23+)
# git switch -c feature/user-authentication

# Create authentication file
cat << EOF > auth.js
class Authentication {
    constructor() {
        this.users = [];
    }

    login(username, password) {
        console.log(\`Attempting login for: \${username}\`);
        return true;
    }

    logout() {
        console.log('User logged out');
    }
}

module.exports = Authentication;
EOF
```

```
# Add and commit changes
git add auth.js
git commit -m "Add user authentication module"

# Push feature branch to GitHub
git push -u origin feature/user-authentication
```

2.2 Create Another Feature Branch

```
# Switch back to main
git checkout main

# Create another feature branch
git checkout -b feature/database-setup

# Create database configuration
cat << EOF > database.js
const config = {
    host: 'localhost',
    port: 3306,
    database: 'myapp',
    user: 'root',
    password: 'password'
};

function connectDB() {
    console.log('Connecting to database...');
    console.log(\`Host: \${config.host}:\${config.port}\`);
}

module.exports = { config, connectDB };
EOF

# Update app.js to use database
cat << EOF > app.js
const { connectDB } = require('./database');

console.log('Hello Git!');
connectDB();
console.log('Application started successfully');
EOF

# Commit changes
git add database.js app.js
git commit -m "Add database configuration and update app.js"
git push -u origin feature/database-setup
```

Part 3: Creating Merge Conflicts (Intentionally)

3.1 Modify app.js in Both Branches

```
# First, modify app.js in feature/user-authentication branch
git checkout feature/user-authentication

cat << EOF > app.js
const Authentication = require('./auth');

const auth = new Authentication();
console.log('Welcome to Secure App!');
auth.login('admin', 'password123');
console.log('Authentication system loaded');
EOF

git add app.js
git commit -m "Integrate authentication in app.js"
git push origin feature/user-authentication

# Now checkout the other branch and modify the same file
        differently
git checkout feature/database-setup

cat << EOF > app.js
const { connectDB } = require('./database');

console.log('Database Application v2.0');
connectDB();
console.log('Database connection established');
console.log('Ready to process data');
EOF

git add app.js
git commit -m "Update app.js with enhanced database integration"
git push origin feature/database-setup
```

3.2 Merge and Resolve Conflicts

```
# Switch to main branch
git checkout main

# Merge first feature (this will work smoothly)
git merge feature/database-setup

# Now try to merge the second feature (this will create conflict)
git merge feature/user-authentication

# Git will show conflict message. Check status:
git status

# View the conflicted file
cat app.js
```

The conflicted `app.js` will look like this:

```
<<<<<<< HEAD
const { connectDB } = require('./database');

console.log('Database Application v2.0');
connectDB();
console.log('Database connection established');
console.log('Ready to process data');
=======
const Authentication = require('./auth');

const auth = new Authentication();
console.log('Welcome to Secure App!');
auth.login('admin', 'password123');
console.log('Authentication system loaded');
>>>>>>> feature/user-authentication
```

3.3 Resolve the Conflict

```
# Edit app.js to combine both features
cat << EOF > app.js
const { connectDB } = require('./database');
const Authentication = require('./auth');

const auth = new Authentication();

console.log('Secure Database Application v2.0');
```

```
connectDB();
console.log('Database connection established');

auth.login('admin', 'password123');
console.log('Authentication system loaded');
console.log('Application ready to process secure data');
EOF

# Mark conflict as resolved and commit
git add app.js
git commit -m "Resolve merge conflict: combine auth and database
        features"

# Push resolved changes
git push origin main
```

Part 4: Pull Requests Workflow

4.1 Create a New Feature for Pull Request

```
# Create new feature branch
git checkout -b feature/user-profile

# Create user profile functionality
cat << EOF > profile.js
class UserProfile {
    constructor(username) {
        this.username = username;
        this.profile = {
            name: '',
            email: '',
            createdAt: new Date()
        };
    }

    updateProfile(name, email) {
        this.profile.name = name;
        this.profile.email = email;
        console.log(\`Profile updated for \${this.username}\`);
    }

    getProfile() {
        return this.profile;
    }
}

module.exports = UserProfile;
EOF

# Update README with new features
cat << EOF > README.md
# Git Practical Session

A comprehensive application demonstrating Git workflow with
        multiple features.

## Features

-   User Authentication System
-   Database Configuration
-   User Profile Management
-   Secure Data Processing

## Files Structure

- \`app.js\` - Main application file
- \`auth.js\` - Authentication module
- \`database.js\` - Database configuration
- \`profile.js\` - User profile management

## Getting Started

\`\`\`bash
node app.js
\`\`\`

## Recent Updates

- Added user authentication
- Integrated database connectivity
- Implemented user profile system
- Resolved merge conflicts between features
\`\`\`
EOF

# Commit changes
git add profile.js README.md
git commit -m "Add user profile management system
```

```
- Implement UserProfile class with CRUD operations
- Update README with comprehensive project information
- Add profile management to feature list"

# Push feature branch
git push -u origin feature/user-profile
```

4.2 Create Pull Request (GitHub Web Interface Steps)

After pushing the branch, go to GitHub and:

1. Navigate to your repository
2. Click "Compare & pull request" button
3. Fill in PR details:

Title: Add User Profile Management System

Description:

```
## Changes Made
- Added UserProfile class with profile management capabilities
- Updated README with comprehensive project documentation
- Integrated profile system with existing authentication

## Testing
- [x] Profile creation works correctly
- [x] Profile updates function as expected
- [x] Integration with auth system verified

## Related Issues
Implements user story for profile management functionality

## Screenshots/Examples
N/A - Backend functionality

## Checklist
- [x] Code follows project standards
- [x] Tests added (if applicable)
- [x] Documentation updated
- [x] No breaking changes introduced
```

4.3 Review and Merge Pull Request

```
# Reviewer commands (simulate code review):

# Fetch all remote branches
```

```
git fetch origin

# Check out the PR branch for local testing
git checkout feature/user-profile

# Test the code
node -e "
const UserProfile = require('./profile.js');
const profile = new UserProfile('testuser');
profile.updateProfile('John Doe', 'john@example.com');
console.log(profile.getProfile());
"

# Switch back to main for merge
git checkout main

# If review is approved, merge the PR (this can be done via
        GitHub UI or CLI)
git pull origin main  # Get latest changes
git merge feature/user-profile
git push origin main

# Clean up: delete merged branch
git branch -d feature/user-profile
git push origin --delete feature/user-profile
```

Part 5: Advanced Git Operations

5.1 View Project History

```
# Comprehensive commit history
git log --graph --pretty=format:'%Cred%h%Creset -%C(yellow)
        %d%Creset %s %Cgreen(%cr) %C(bold blue)<%an>%Creset' --
        abbrev-commit

# See branch history
git log --oneline --graph --all

# Check differences between commits
git diff HEAD~2 HEAD

# See what changed in last commit
git show HEAD
```

5.2 Useful Git Commands for Projects

```
# Check which files have been modified
git status --porcelain

# See unstaged changes
git diff

# See staged changes
git diff --cached

# Undo last commit (keep changes)
git reset --soft HEAD~1

# Undo last commit (discard changes) - BE CAREFUL!
git reset --hard HEAD~1

# Create and apply stash
git stash push -m "Work in progress on new feature"
git stash list
git stash apply stash@{0}

# Tag a release
git tag -a v1.0.0 -m "First stable release"
git push origin v1.0.0
```

Part 6: Collaboration Workflow

6.1 Simulate Team Collaboration

```
# Clone the repository (simulate another team member)
cd ..
git clone https://github.com/username/git-practical.git git-
        practical-teammate
cd git-practical-teammate

# Create feature branch
git checkout -b feature/error-handling

# Add error handling
cat << EOF > errorHandler.js
class ErrorHandler {
    static handle(error) {
        console.error('Application Error:', error.message);
        console.error('Stack:', error.stack);
    }

    static logError(message) {
        const timestamp = new Date().toISOString();
        console.log(\`[\${timestamp}] ERROR: \${message}\`);
    }
}

module.exports = ErrorHandler;
EOF

# Update app.js with error handling
cat << EOF > app.js
const { connectDB } = require('./database');
const Authentication = require('./auth');
const ErrorHandler = require('./errorHandler');

try {
    const auth = new Authentication();

    console.log('Secure Database Application v2.0');
    connectDB();
    console.log('Database connection established');

    auth.login('admin', 'password123');
    console.log('Authentication system loaded');
    console.log('Application ready to process secure data');

} catch (error) {
    ErrorHandler.handle(error);
}
EOF

git add errorHandler.js app.js
git commit -m "Add comprehensive error handling system"
git push -u origin feature/error-handling
```

Summary of Commands Used

Repository Setup

```
git init
git remote add origin <url>
git clone <url>
```

Branch Management

```
git branch <branch-name>
git checkout <branch-name>
git checkout -b <new-branch>
git merge <branch-name>
git branch -d <branch-name>
```

Basic Operations

```
git add <files>
git commit -m "message"
git push origin <branch>
git pull origin <branch>
git status
git log
```

Conflict Resolution

```
git merge <branch>  # May create conflicts
# Edit conflicted files manually
git add <resolved-files>
git commit -m "Resolve merge conflict"
```

This practical covers all essential Git and GitHub operations you'll use in real projects. Practice these commands and workflows to build confidence with version control!