
Git Exercises

Peter J. Jones
✉ pjones@devalot.com
🐦 @devalot
<http://devalot.com>
OCTOBER 1, 2018



DEVALOT

Copyright © 2018 Peter J. Jones
Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International
Public License: <https://creativecommons.org/licenses/by-nc-sa/4.0/> To license
this work for use in a commercial setting contact Peter J. Jones.

Contents

1	Git Introduction	5
2	Branching	9
3	Merging	11
4	Rebasing	13
5	Resetting Trees	15
6	Reviewing and Editing the Commit History	17
7	Improving Your Daily Workflow	21
8	Including External Repositories	23
9	Git as a Debugging Tool	25

CONTENTS

Chapter 1

Git Introduction

1.0.1 Exercise: Cloning a Repository

1. Change to the following directory:

```
repos
```

2. Clone the `basic.git` repository:

```
$ git clone basic.git
```

3. This should have created a `basic` directory

1.0.2 Exercise: Viewing the History Log

1. Change to the directory holding a clone of the following repository:

```
repos/basic.git
```

2. Use the following command to see the commit history:

```
$ git log --oneline
```

3. Pick a commit and run:

```
$ git cat-file -p <commit>
```

4. Also `cat-file` the listed tree, then the listed blob

1.0.3 Exercise: Telling Git Who You Are

Goal: Ensure basic user settings are configured.

If you haven't done so previously:

-
1. Set your name and email address using `git config`
 2. Verify your settings:

```
$ git config --list
```

1.0.4 Exercise: Telling Git Which Editor to Use

Goal: Ensure Git can use your favorite text editor.

If you haven't done so previously:

1. Tell Git which text editor to use:

```
$ git config --global core.editor <program>
```

Here are some examples for `<program>`:

- `emacs`
- `vi`
- `"C:/Program Files/Notepad++/notepad++.exe"`
- `"code --wait"`

Or search the web for instructions for your preferred editor.

1.0.5 Exercise: Adding a New File

Goal: Practice adding new files to a Git repository and seeing how Git responds at each step.

1. Change to the directory holding a clone of the following repository:

```
repos/basic.git
```

2. Create a new file called `config.h`
3. Add some content to the file (any text will do)
4. Review the output of `git status`
5. Add the file to the index
6. Check the output of `git status` again
7. Commit the change

1.0.6 Exercise: Editing an Existing File

Goal: Practice editing files that are tracked by Git, seeing how Git responds, and then staging and committing those changes.

1. Change to the directory holding a clone of the following repository:

`repos/basic.git`

2. Edit the `main.c` file
3. Add some content to the file (any text will do)
4. Review the output of `git status`
5. Review the output of `git diff`
6. Add and commit the change

1.0.7 Exercise: Renaming a File

Goal: Practice renaming files with and without Git and see how it response along the way.

1. Change to the directory holding a clone of the following repository:

`repos/basic.git`

2. Rename the `config.h` file from the previous exercise to `init.h`
3. Review the output of `git status`
4. Notice that Git already staged the rename
5. Commit your change

1.0.8 Exercise: Removing a File

Goal: Practice deleting a file and then restoring it.

1. Change to the directory holding a clone of the following repository:

`repos/basic.git`

2. Remove the `init.h` file
3. Review the output of `git status`
4. Notice that the file deletion was staged for you
5. Commit your changes
6. Restore the file

Chapter 2

Branching

2.0.1 Exercise: Moving HEAD Around

Goal: Practice using `git checkout` to move the HEAD pointer.

1. Change to the directory holding a clone of the following repository:
`repos/basic.git`
2. List the names of all branches
3. For each branch:
 1. Check out the branch
 2. Use `git log --oneline` to find the latest commit hash

2.0.2 Exercise: Creating Branches

Goal: Practice creating and switching branches.

1. Change to the directory holding a clone of the following repository:
`repos/basic.git`
2. Switch to the `master` branch
3. Create a new branch called `m2`
4. Use `git log` to see the latest commit:
`$ git log --oneline -1`
5. Create a branch called `f2` that starts at `merge-start`
6. Compare the latest commits on `m2` and `f2`

Chapter 3

Merging

3.0.1 Exercise: Merging a Branch

Goal: Practice merging one branch into another and examining the resulting merge commit.

1. Change to the directory holding a clone of the following repository:

```
repos/basic.git
```

2. Create and switch to a new branch that starts at `v1.0`:

```
$ git checkout -b NAME v1.0
```

3. Merge the `origin/feature` branch into your new branch

3.0.2 Exercise: Merging with Conflicts

Goal: Practice resolving conflicts when merging one branch into another.

1. Change to the directory holding a clone of the following repository:

```
repos/conflicts.git
```

2. Create and switch to a new branch that starts at `v1.0`:

```
$ git checkout -b NAME v1.0
```

3. Merge the `origin/feature` branch into your new branch

4. Resolve conflicts and finish the merge

3.0.3 Exercise: Automatically Resolving Conflicts

Goal: Practice advanced merging where we pass options to control the merge strategy.

We'll be using the **ours** flag and passing it to the recursive strategy so all conflicts are resolved by choosing the hunk present in **HEAD**. Then we'll restart the exercise, this time using the **theirs** conflict resolution strategy.

1. Change to the directory holding a clone of the following repository:
`repos/conflicts.git`
2. Create a new branch whose ancestor is the commit before **HEAD**
3. Merge with the **ours** option
4. Repeat the above steps, except this time use the **theirs** option

Chapter 4

Rebasing

4.0.1 Exercise: Rebasing a Branch

Goal: Practice rebasing a local topic branch onto its local ancestor. We'll also review the new commits that Git created.

1. Change to the directory holding a clone of the following repository:
`repos/basic.git`
2. Create a branch that starts at the commit named `merge-start`
`$ git checkout -b NAME merge-start`
3. Use `git log --oneline` to review the commit history
4. Rebase onto of the `master` branch
5. Review the commit history again and identify the rewritten commits and the merged commits from `master`

4.0.2 Exercise: Rebasing with Conflicts

Goal: Practice a rebasing a local topic branch onto its local ancestor like before, but this time in the presence of merge conflicts.

1. Change to the directory holding a clone of the following repository:
`repos/conflicts.git`
2. Create a branch that starts at the commit named `merge-start`
`$ git checkout -b NAME conflict-start`
3. Use `git log --oneline` to review the commit history

-
4. `Rebase` onto of the `master` branch
 5. Fix the conflict and use `git rebase --continue` to resume
 6. Review the commit history again and identify the rewritten commits and the merged commits from `master`

4.0.3 Exercise: Interactive Rebasing

Goal: Practice editing commits by performing an interactive rebase.

1. Change to the directory holding a clone of the following repository:
`repos/basic.git`
2. Create a branch that starts at the commit named `merge-start`
3. Interactively edit the commit which has the message “Add a version number”
4. Amend the commit when the rebase stops on it:
`$ git commit --amend`
5. Resume the rebase:
`$ git rebase --continue`

Chapter 5

Resetting Trees

5.0.1 Exercise: Soft Reset

Goal: Practice resetting `HEAD` so that we back out a commit but leave the index and working directory intact.

1. Change to the directory holding a clone of the following repository:

```
repos/basic.git
```

2. Create a branch that starts at the commit named `merge-start`

```
$ git checkout -b NAME merge-start
```

3. Use `git log --oneline` to review the commit history

4. Use `git reset --soft` to back up 2 commits:

```
$ git reset --soft HEAD~2
```

5. Use `git log --oneline` to see how the history changed

6. Use `git status` to review the index and working directory

7. Use `git commit` to create a new commit

5.0.2 Exercise: Mixed Reset

Goal: Practice resetting `HEAD` and the index so that we can back out a commit and leave the working directory intact.

1. Change to the directory holding a clone of the following repository:

```
repos/basic.git
```

2. Create a branch that starts at the commit named `merge-start`

-
- ```
$ git checkout -b NAME merge-start
```
  3. Use `git log --oneline` to review the commit history
  4. Use `git reset --mixed` to back up 2 commits:  

```
$ git reset --mixed HEAD~2
```
  5. Use `git log --oneline` to see how the history changed
  6. Use `git status` to review the index and working directory
  7. Use `git commit` to create a new commit

### 5.0.3 Exercise: Hard Rest

Goal: Practice resetting all three trees so that we restore them to a previous commit. We'll also see how stage changes and modifications to the working directory are **lost** during a hard reset.

1. Change to the directory holding a clone of the following repository:  
`repos/basic.git`
2. Create a branch that starts at the commit named `merge-start`  

```
$ git checkout -b NAME merge-start
```
3. Use `git log --oneline` to review the commit history
4. Use `git reset --hard` to back up 2 commits:  

```
$ git reset --hard HEAD~2
```
5. Use `git log --oneline` to see how the history changed
6. Use `git status` to review the index and working directory
7. Notice that the working directory was also changed



## Chapter 6

# Reviewing and Editing the Commit History

### 6.0.1 Exercise: Amending the Last Commit

Goal: Practice amending the previous commit in order to reword a commit message.

1. Change to the directory holding a clone of the following repository:

```
repos/basic.git
```

2. Create a branch that starts at the commit named `merge-start`

```
$ git checkout -b NAME merge-start
```

3. Take note of the last commit hash
4. Amend the last commit, changing the commit message
5. Notice that the most recent commit was rewritten

### 6.0.2 Exercise: Unstaging a File

Goal: Practice using `git reset` to unstage a file that we no longer want to be part of the next commit. Later we'll see how to create an alias to act as a shortcut for this common task.

1. Change to the directory holding a clone of the following repository:

```
repos/basic.git
```

2. Create a branch that starts at the commit named `merge-start`

---

```
$ git checkout -b NAME merge-start
```

3. Edit the `main.c` file, making a simple change to it
4. Stage (`git add`) and unstage (`git reset`) the file

### 6.0.3 Exercise: Restoring a File

Goal: Practice using `git checkout` to restore a file from the last commit.

1. Change to the directory holding a clone of the following repository:  
`repos/basic.git`
2. In the last exercise we changed the `main.c` file, restore it back to its previous state

### 6.0.4 Exercise: Restoring a File

Goal: Practice using `git log` and `git checkout` to restore a file from any previous version.

1. Change to the directory holding a clone of the following repository:  
`repos/basic.git`
2. Look through the commit history and find the commit that added a version number to `main.c`
3. Restore `main.c` to its content before the version number was added

### 6.0.5 Exercise: Reverting a Commit

Goal: Practice using the `git revert` command to undo a specific commit and see how the repository's history changes.

1. Change to the directory holding a clone of the following repository:  
`repos/basic.git`
2. Create a branch that starts at the commit named `merge-start`  

```
$ git checkout -b NAME merge-start
```
3. Edit the `main.c` file, making a simple change to it
4. Commit the change
5. Revert the commit you just created

### 6.0.6 Exercise: Squashing Commits

Goal: Practice using `git rebase` to squash several commits into a single commit.

1. Change to the directory holding a clone of the following repository:

`repos/basic.git`

2. Create a branch that starts at the commit named `merge-start`

`$ git checkout -b NAME merge-start`

3. Take note of the commit history
4. Start an interactive rebase with the first commit in the history
5. Squash all commits into the first listed commit
6. Review the commit history



## Chapter 7

# Improving Your Daily Workflow

### 7.0.1 Exercise: Staging Patch Hunks

Goal: Practice interactively staging parts of a file while leaving the rest of the file unstaged.

1. Change to the directory holding a clone of the following repository:

```
repos/conflicts.git
```

2. Create a branch that starts off the `origin/feature` branch
3. Edit `main.c`; add comments to the top and bottom of the file
4. Review the difference between the working directory and the index:

```
$ git diff
```

5. Interactively stage one of the patch hunks by splitting the first hunk:

```
$ git add -p main.c
...
Stage this hunk [y,n,q,a,d,/,j,J,g,e,?]? s
```

6. Review the updated index:

```
$ git status
$ git diff --cached
```

---

### 7.0.2 Exercise: Pushing and Popping the Stash

Goal: Practice using `git stash` to save and restore changes to the index and working directory.

1. Change to the directory holding a clone of the following repository:  
`repos/conflicts.git`
2. Create a branch that starts off the `origin/feature` branch
3. Add a comment to the bottom of `main.c`
4. Create a new stash
5. Review the index and working directory state
6. Restore the stash that was previously created

### 7.0.3 Exercise: Create Some Aliases

Goal: Practice creating and using Git aliases as shortcuts for commonly used commands.

1. Take a moment and create some helpful aliases
2. Pick a repository and test your new aliases

### 7.0.4 Exercise: Tagging the Current Commit

Goal: Practice creating tags and using the `git tag` tool to list them.

1. Change to the directory holding a clone of the following repository:  
`repos/basic.git`
2. Create a tag called `v1.1`
3. Use `git tag --list` to list your tags
4. Try using `git rev-parse` to see how the tag name resolves
5. Create a branch starting from the tag
6. Push the tag back to the `origin` remote
7. Use `git ls-remote` to confirm the tags were pushed

## Chapter 8

# Including External Repositories

### 8.0.1 Exercise: Experimenting with Submodules

Goal: Practice using submodules as a way to manage a subproject. We're going to take our time and experiment with the following features:

- Playing with a detached **HEAD**
- Updating a submodule to the locked commit
- Updating a submodule to its latest upstream commit
- Pushing changes back to the submodule's upstream

For each of the following steps you should create a unique branch off of **master** so you can easily jump back to the previous state.

1. Change to the directory holding a clone of the following repository:  
`repos/submodules.git`
2. Go into the **other** submodule and see what branch it is on
3. Update the submodule to its latest upstream commit
4. Commit and push the project
5. Make a change to the submodule and commit it
6. Commit to the project and push everything
7. Add another submodule

---

## 8.0.2 Exercise: Experimenting with Subtrees

Goal: Practice using subtrees as a way to manage subprojects. We're going to experiment with the following features:

- Adding another repository as a subtree
- Making changes to a subtree
- Pushing changes back to the subproject

1. Change to the directory holding a clone of the following repository:

```
repos/subtrees.git
```

2. Alter the `other/main.c` file and commit the change

3. Notice that all files are local to the repository

4. Read the `notes.txt` file and then push the subtree



## Chapter 9

# Git as a Debugging Tool

### 9.0.1 Exercise: Using the Blame Command

Goal: Practice using `git blame` to identify when each line of a file was altered and by whom.

(Instructions forthcoming.)

### 9.0.2 Exercise: Bisecting a Project

Goal: Practice using `git bisect` to quickly track down and identify which commit introduced a bug.

(Instructions forthcoming.)