Git Fundamentals

Donald Dong

November 27, 2018

CST438 Software Engineering @ Cal State Monterey Bay

Introduction

Introduction: Version control

- Keeping track of everything

  Version control lets you offload the work of keeping track of everything related to a project, including documents, visualizations, and data.

- Publish your changes

  You initialize it in a directory, and then tell it when you want your changes to be permanent.

- Travel in time

  You can go back to a previous iteration at any time, and can see exactly what has changed between versions.

Introduction: Distributed Version Control Systems

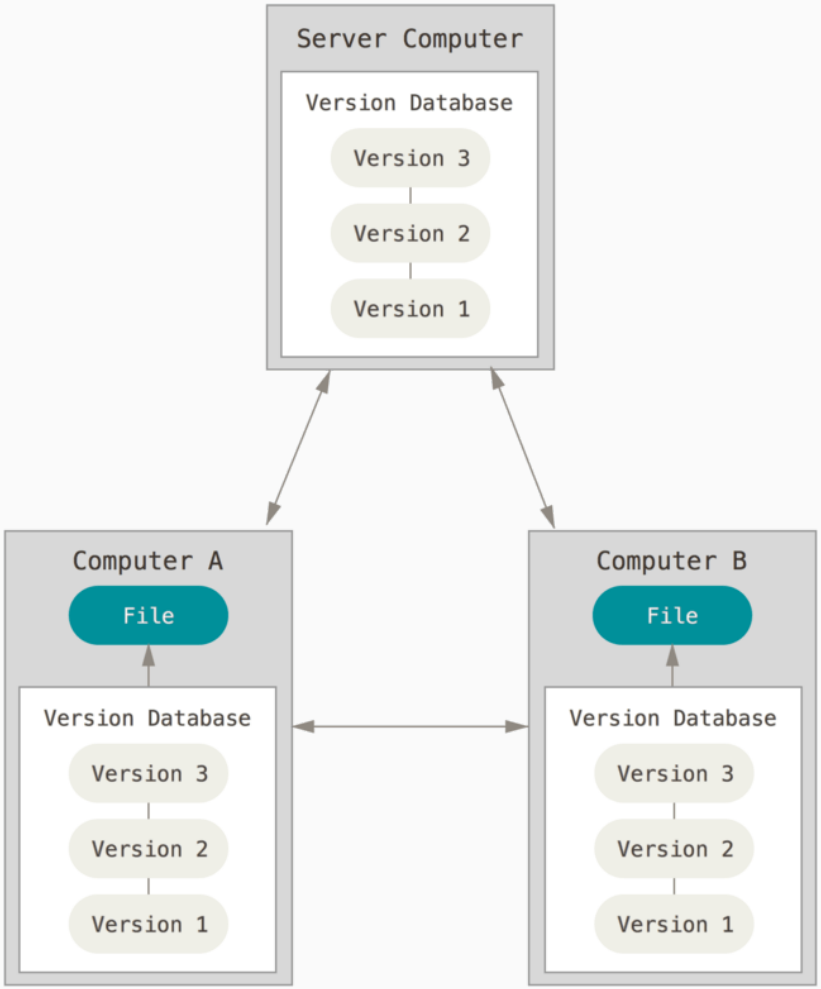


Figure 1: Distributed version control

Fully mirror the repository

- Recovery

  If any server dies, and these systems were collaborating via that server, any of the client repositories can be copied back up to the server to restore it.

- Collaboration

  You can collaborate with different groups of people in different ways simultaneously within the same project.

Introduction: Git and Github

"Simplicity is prerequisite for reliability." - Edsger W. Dijkstra

Git

A powerful DVCS written on top of a content-addressable filesystem

- Speed

  Able to handle large projects like the Linux kernel efficiently
- Simplicity

  Blob, Tree, Commit
- Integrity

  Tracking the full history. Almost (link) unbreakable.

Github

A Git hosting service with powerful features

- Profile

  Involvement, Contribution and Influence
- Code Review

  Social Coding, Pull Requests
- Project Management

  Issues, Feature Requests, Projects, and Milestones

Tracking

Tracking: Repository

Getting a Git Repository

- Make a Git repository

```
git init [directory]
```

- Clone a Git repository

```
git clone <repository>
 ↪   [<directory>]
```

(Link to Documentation)

Subdirectory: .git

- Contains repository files

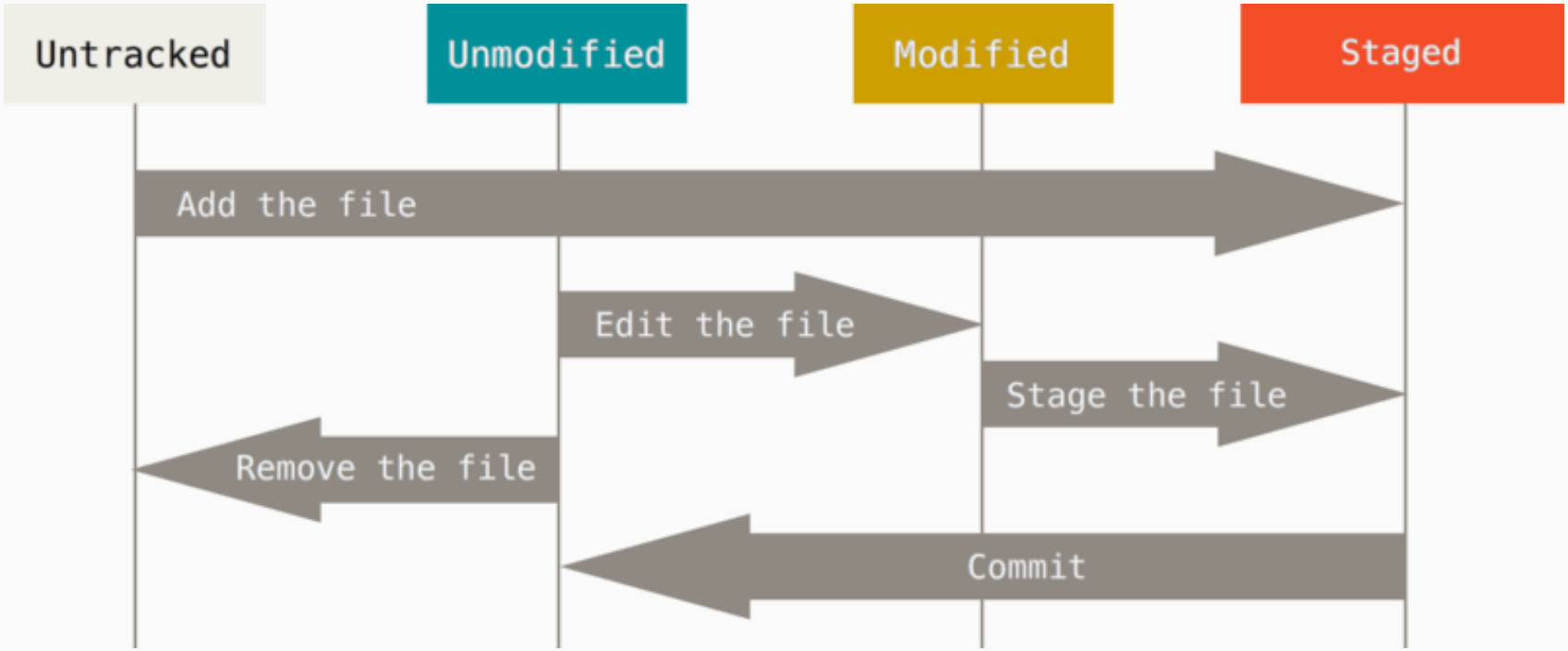
  Every version of every file for the history of the project is pulled down by default when you run git clone.

- The skeleton for the repository

  The source of magic. What are contained in the .git directory? (Git Internals)

Tracking: The Four File Status

Figure 2: The lifecycle of the status of your files



In a repository, a file can exist in one of four status: Untracked, Unmodified, Modified, or Staged.

The main tool you use to determine which files are in which status is:

```
git status
```

Tracking: The Three Main Areas

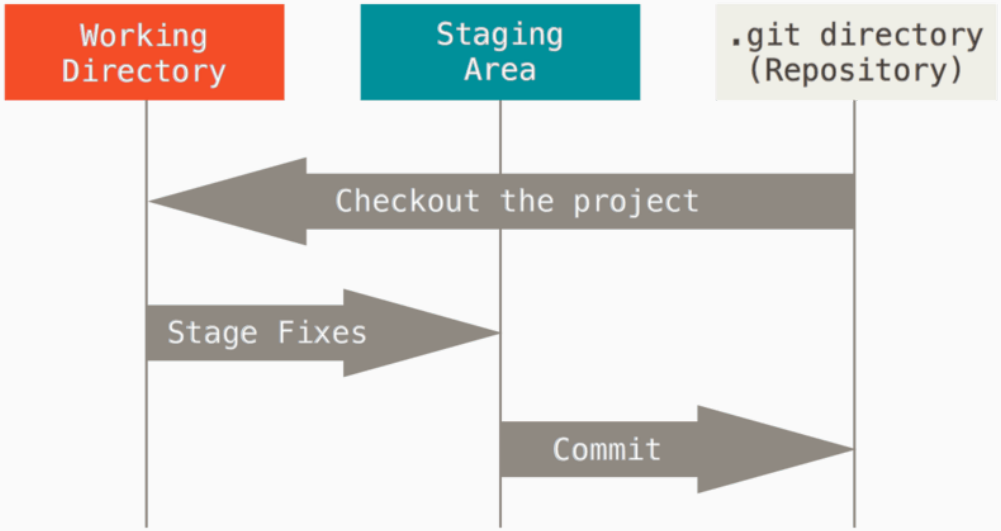


Figure 3: The three main areas

Corresponds to the three file stages: Modified, Staged, and Committed.

Working Directory (Tree)

A single checkout of one version of the project.

Staging Area

Stores information about what will go into your next commit.

Git directory

Stores the version database for your project.

Tracking: Snapshots, Not Differences

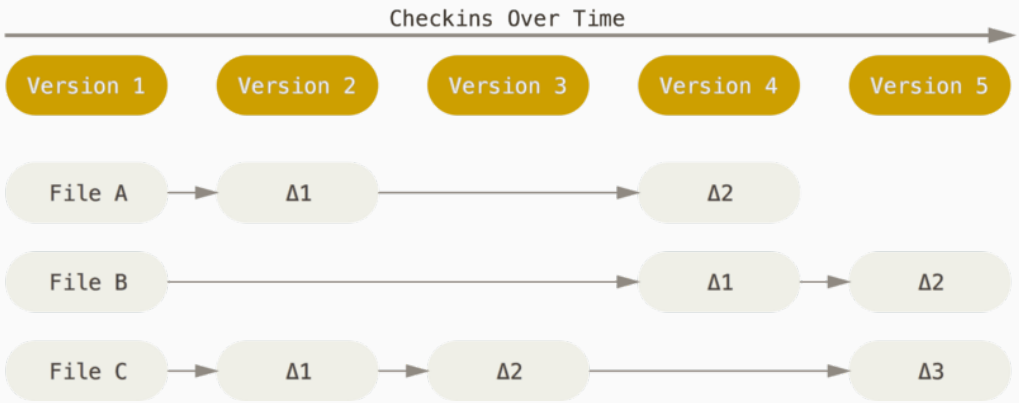


Figure 4: Storing data as differences

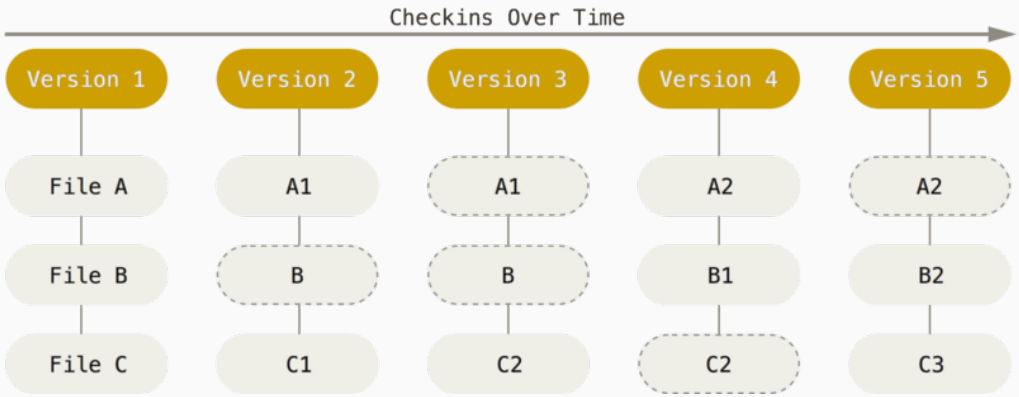


Figure 5: Storing data as snapshots

With deltas

- Most other VCSs store information as a list of file-based changes (deltas)
- Space efficient

With snapshots

- Fetches the history instantly
- Calculates the differences on the fly
- Imitates a Unix-style file system
- Improves branching

Tracking: Create a Git blob manually

1. Create a header

```
"blob #{content.length}\0"
```

2. Create the SHA1 checksum

```
hash = Digest::SHA1.hexdigest(
  header + content
)
```

3. Compresses the content

```
blob = Zlib::Deflate.deflate(
  header + content
)
```

4. Create the path for the blob

```
path = ".git/objects/#{
  hash[0, 2]
}/#{
  hash[2, 38]
}"
FileUtils.mkdir_p(
  File.dirname(path)
)
```

5. Save the blob as a file

```
File.write(path, blob)
```

Tracking: Git Objects

Tree Objects

contain one or more entries, each of which is the SHA-1 hash of a blob or subtree with its mode, type, and filename.

Commit Objects

contain the parent commits (if any), the top-level tree, the author/committer information, and the commit message.

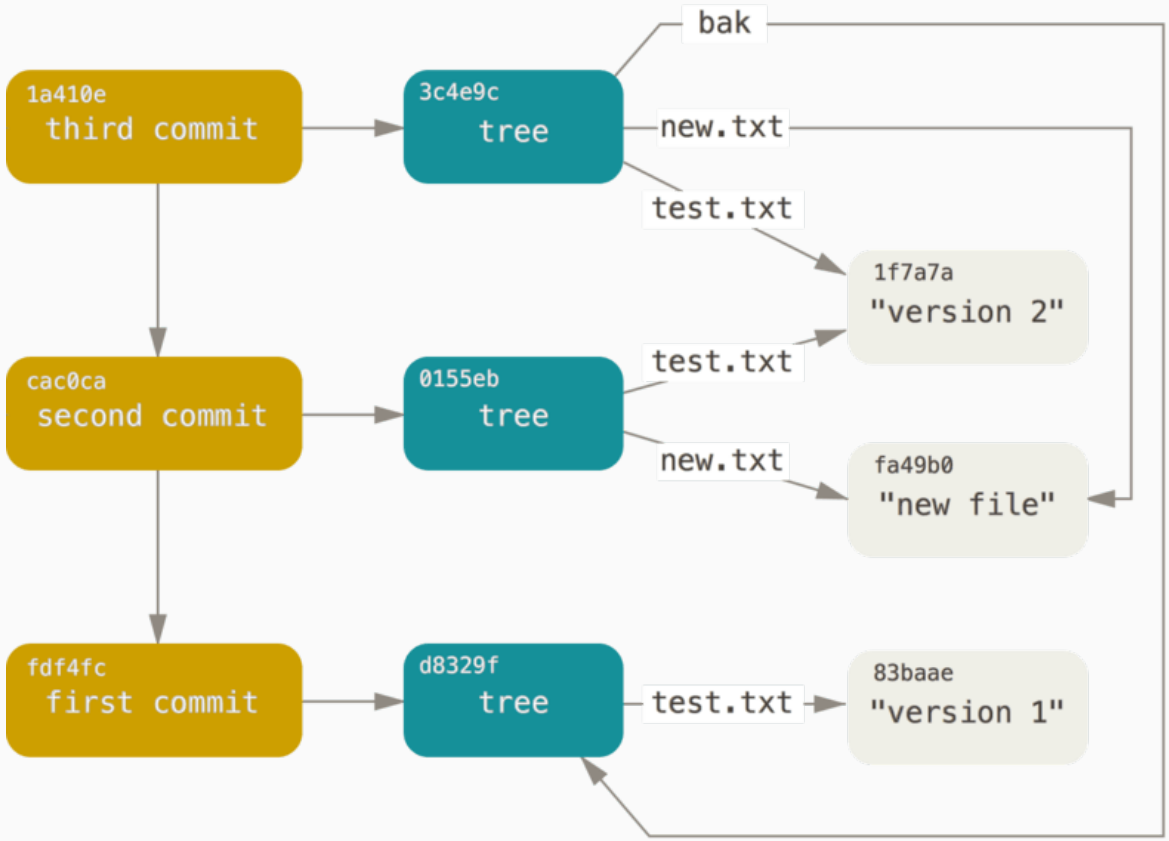


Figure 6: All the reachable objects in your Git directory

The Git objects form a Directed Acyclic Graph (DAG).

Tracking: Summary

Basic Snapshotting

```
git add
git status
git diff
git commit
git reset
git checkout
git rm
```

Four file status: Untracked, Unmodified, Modified, or Staged.

Three main areas: Working Tree, Staging Area, Git directory.

Plumbing Commands

```
git cat-file
git hash-object
git ls-files
```

Git directory

```
.git/index
.git/objects
```

Three typs of Git Objects: Blob, Tree, Commit

Branching and Merging

Branching: Git References

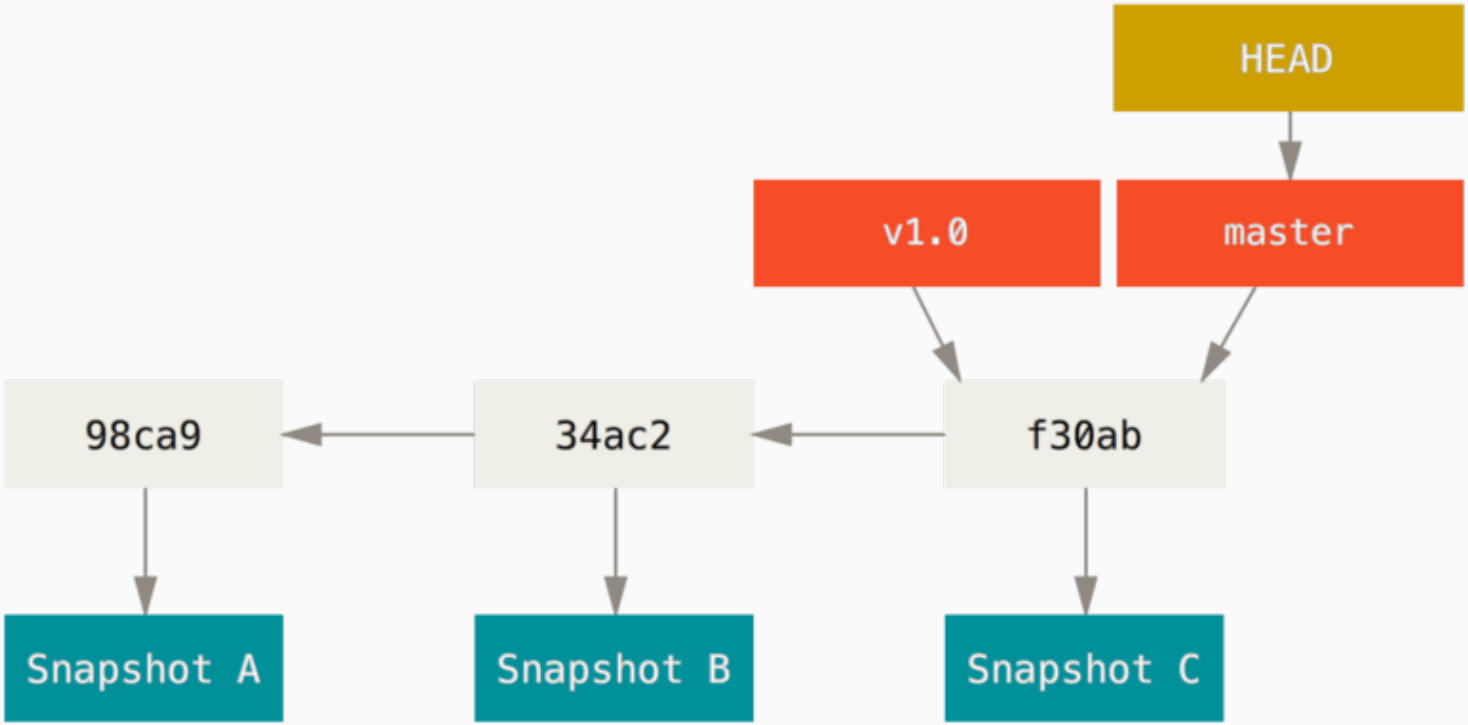


Figure 7: A branch and its commit history

HEAD: a pointer to the local branch you are currently on.

Branch / Tag: a pointer to a commit object.

Merging: Merge vs. Rebase

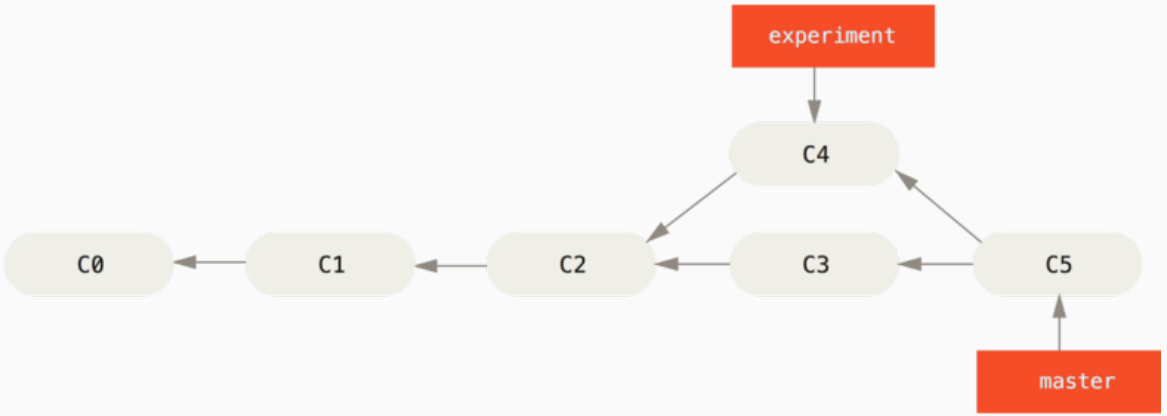


Figure 8: Merging to integrate diverged work history

What actually happened

Avoid to change the commit history

vs.

How your project was made

Tell the story in the way thats best for future readers

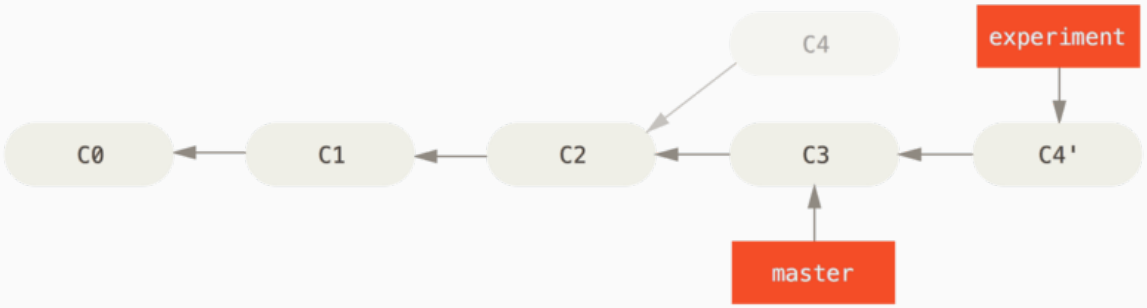


Figure 9: Rebasing the change introduced in C4 onto C3

Branching: Fetch Remote Branches

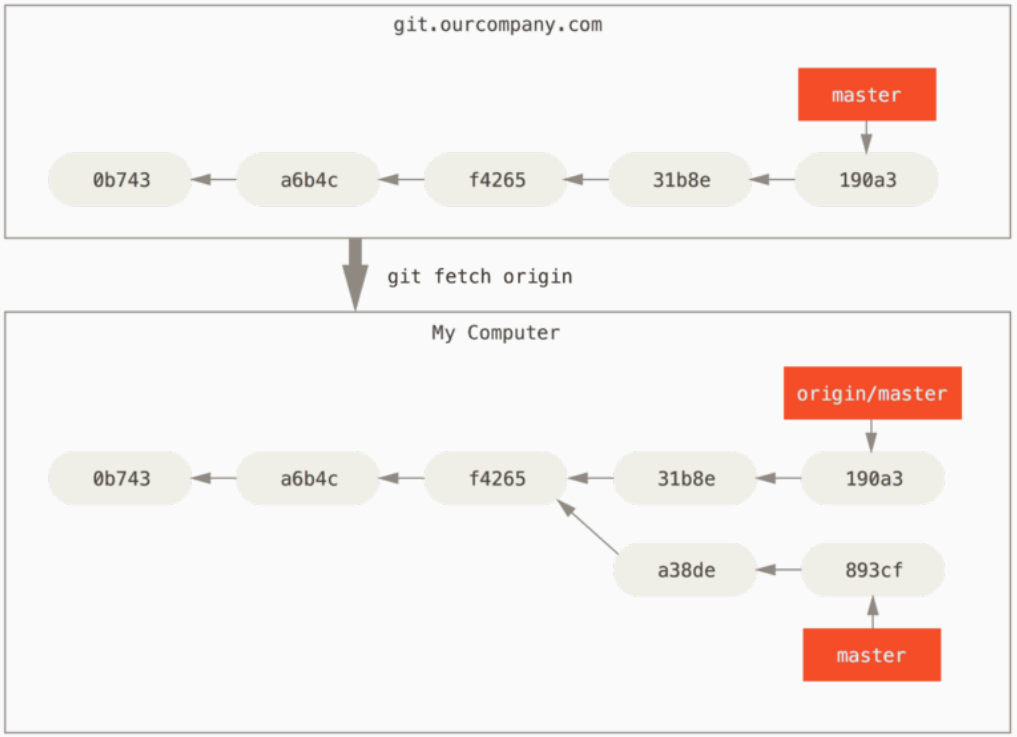


Figure 10: Updates remote-tracking branches

1. Fetch any data that you dont yet have from remote

2. Update the local version database (the Git directory)

3. Move the ‘origin/master‘ pointer to the new commit

Patching

Patching: Resolve Conflicts

```
> git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the
 ↪  result.
```

```
<<<<<<< HEAD:index.html
<div id="footer">
  contact :
  email.support@github.com
</div>
=======
<h3 id="footer">
  please contact us at
  support@github.com
</h3>
>>>>>>> iss53:index.html
```

Helpful commands

```
git merge --abort
git rebase --abort
git status
git diff --check
git mergetool
```

Patching: Reset

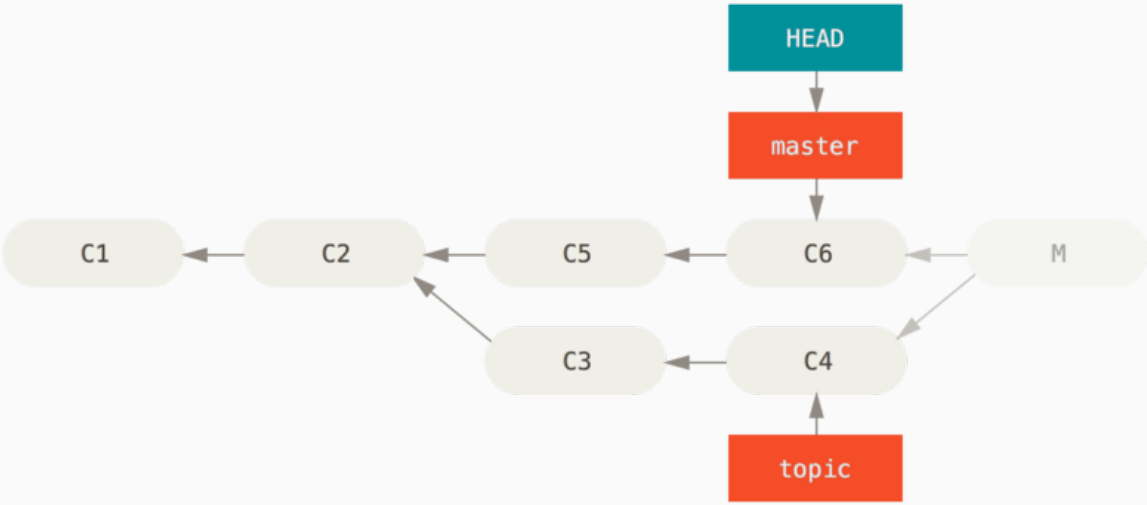


Figure 11: History after reset

```
git reset --hard HEAD^
```

- Move the branch HEAD points to the commit: C6.
- Make the index look like HEAD.
- Make the working directory look like the index.

Patching: Revert

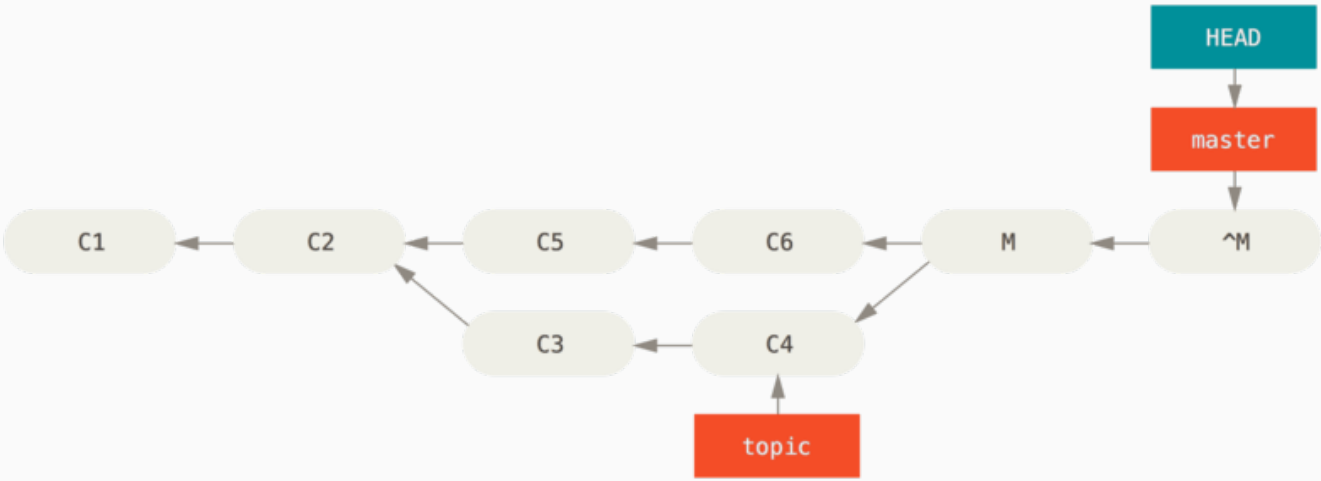


Figure 12: History after revert

```
git revert -m 1 HEAD
```

- The new commit has two parents: C4, C6.
- We want to undo all the changes introduced by C4.
- The ‘-m 1‘ flag indicates to keep all the content from parent 1 (C6).

Branching, Merging and Patching: Summary

Basic Branching and Merging

```
git branch
git checkout
git merge
git rebase
```

Working with remote branches

```
git remote -v
git remote add
git remote remove
git fetch
git push
git pull
```

Git References

HEAD, branch, tag

Resolve Merge Conflicts

1. Locate the conflicts
2. Resolve and stage the changes
3. Continue merging

Undo a commit

```
git reset
git revert
```

References

Chacon, S., & Straub, B. (2014). Pro Git. New York, NY: Apress.