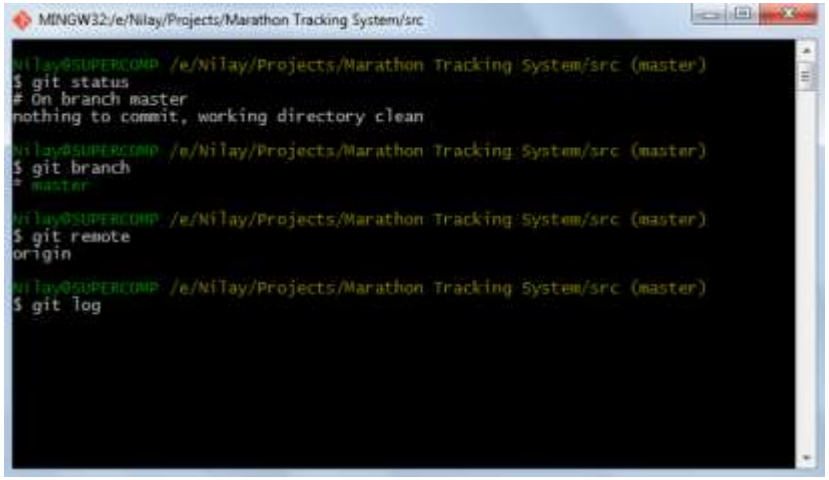
git – Basic Crash Course

Misunderstanding

git ≠

github
SOCIAL CODING



DRIVES

Who uses Git?

CIOFCND

Installation

Debian/Ubuntu based Linux Distributions
$ apt-get install git

Fedora/Redhat based Linux Distributions
$ yum install git

OpenSUSE
$ zypper install git

Arch Linux
$ pacman -S git

Windows
1. Download latest version (1.9.4) from http://git-scm.com/download/win
2. Run the installer
3. Select “Run Git and included Unix tools from the Windows Command Prompt”
4. Select “Checkout Windows-style, commit Unix-style line endings”

Setup

Username and Email
$ git config --global user.name “Nilay Binjola"
$ git config –global user.email “nilaybinjola@gmail.com”

Activate colored messages
$ git config --global color.status auto
$ git config --global color.branch auto

More git configurations settings - http://git-scm.com/book/en/v2/Customizing-Git-Git-Configuration

And you are all set!

Where to go for help?

Git has extensive documentation all over the internet. Some of them are:-

1. **Pro Git book** by **Scott Chacon** and **Ben Straub** - http://git-scm.com/book/en/v2
2. **Git Reference -** http://gitref.org/
3. **Git Manual - $** git help <verb>
4. **Stack Overflow** - http://stackoverflow.com/questions/tagged/git
5. **Google** – http://www.google.com

"Most images/figures/diagrams in this presentation are taken from [1]"

What is Git?

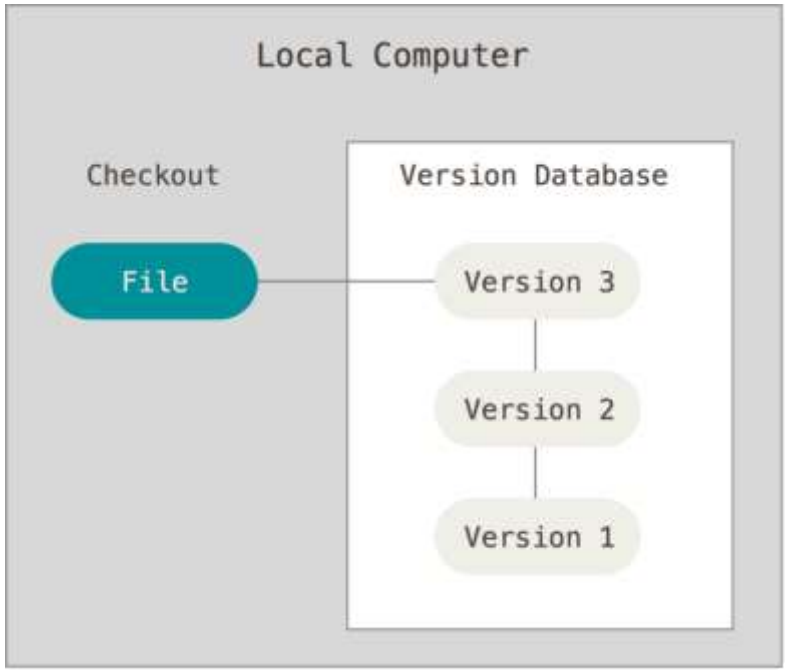
Git is a distributed revision control system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows.

- **What is Version Control?**
  - Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
  - It allows you to revert files back to a previous state, revert the entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem, who introduced an issue and when, and more.

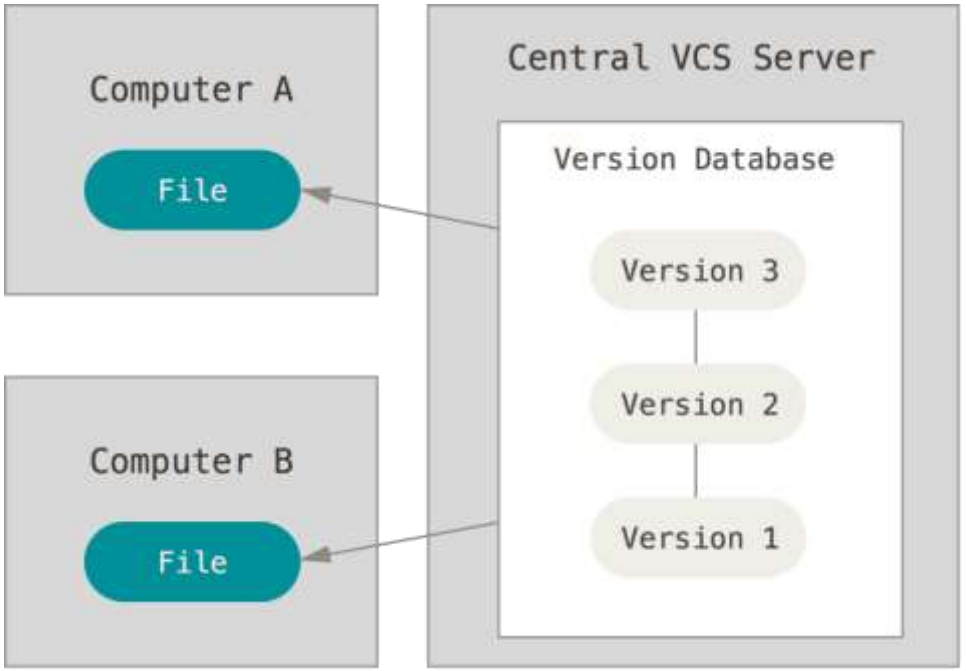
- **Categories of Version Control**
  1. Local Version Control Systems
  2. Centralized Version Control Systems
  3. Distributed Version Control Systems

"A VCS generally means that if you screw things up, you can easily recover."

What is Git?

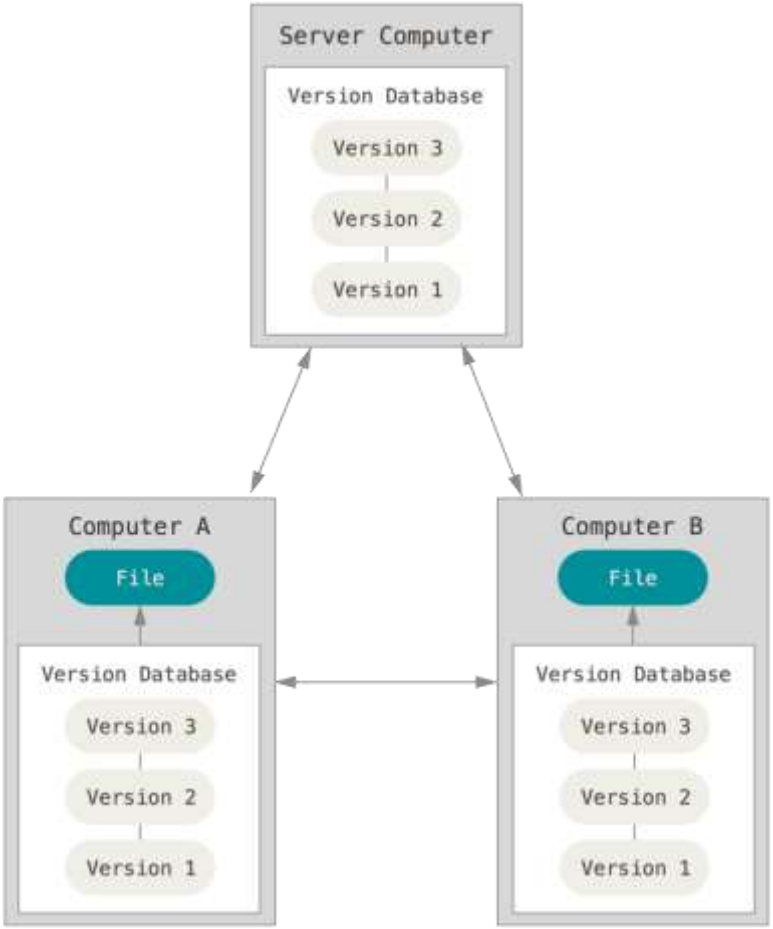


Local Version Control System
Eg: RCS (MAC OS)



Centralized Version Control System
Eg: CVS, Subversion, Perforce etc.

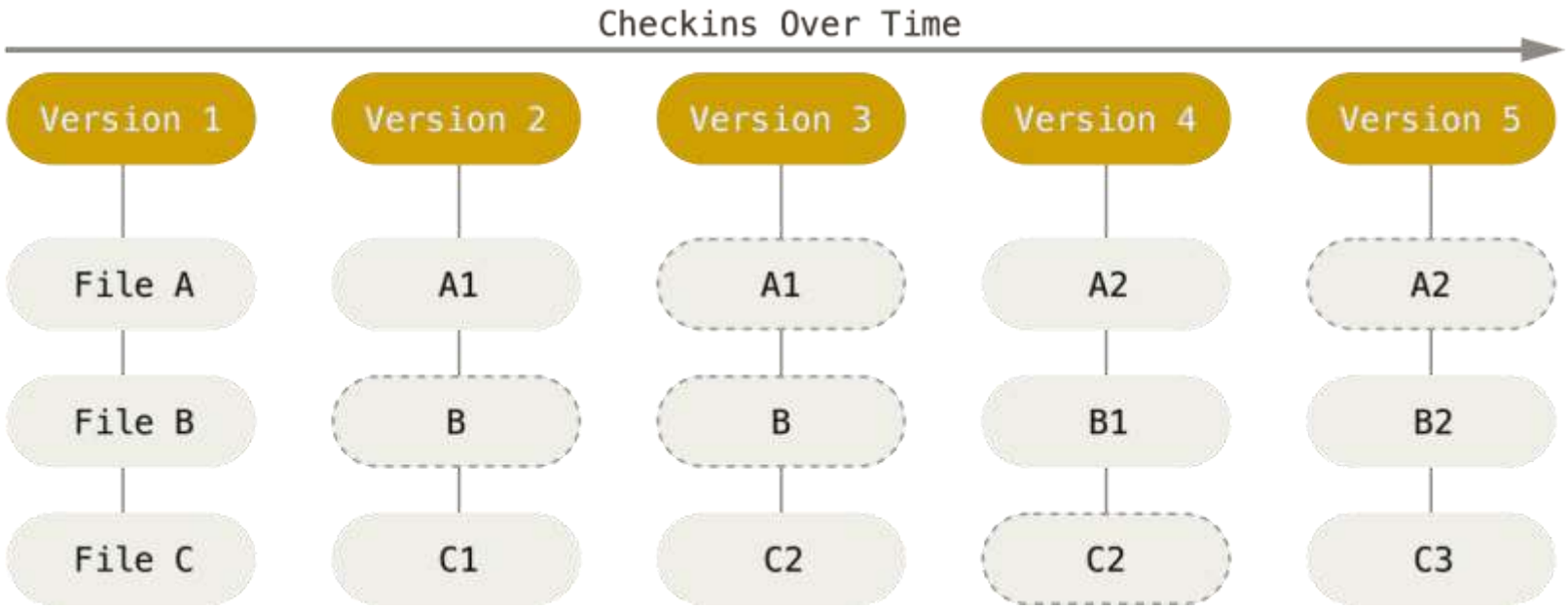
What is Git?



Eg: Git, Mercurial, Bazaar, DARCS etc.

- Clients check-out entire history of project from central server instead of just 1 snapshot like CVCS.
- Supports hierarchical models unlike CVCS.
- Allows users to work productively when not connected to a network.
- Makes most operations **faster** – No need to communicate with a central server.
- **Non-Linear Workflows** – Anyone can be anything since everyone has the complete repository.
- **Distributed** - Protection against data loss.

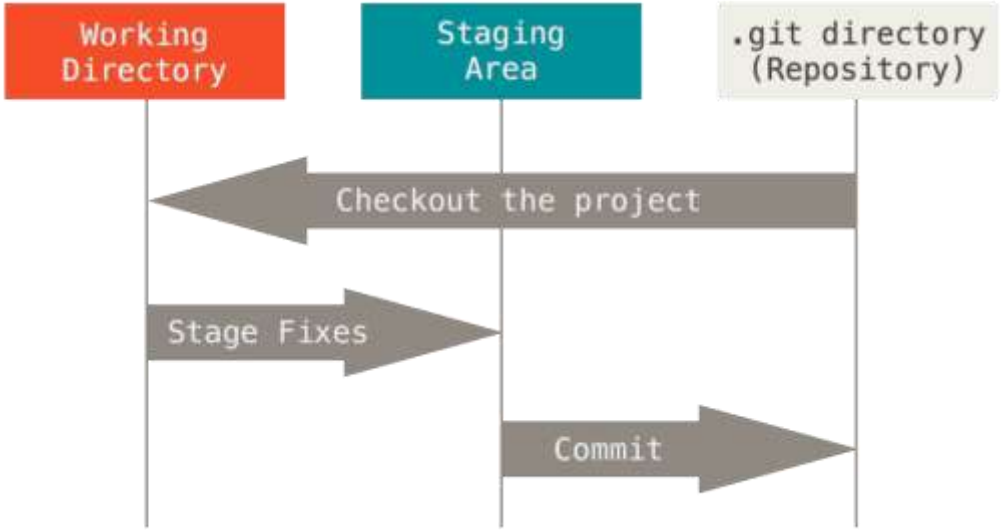
How does Git work?



- Git stores the snapshot of the file along with a reference to it.
- If file has not changed, Git does not store file again
- Everything is check-summed before storage using **SHA-1 Hash**
- Version numbers or commit numbers are first **7** characters of the SHA-1 Hash. Eg: 5610e3b

The Three States

- Git stores all its data in a **.git** directory in the root of the project.
- Working Directory – Latest checkout of the repository along with your changes.
- Staging Area – Stores information of what is to be committed next.
- Staging Area a.k.a Index



- **Git Workflow:-**
  1. Clone/Fork/Initialise a git repository and checkout a snapshot to **Working Directory**
  2. Modify Files in the **Working Directory**
  3. Stage the files you want to be committed by adding snapshots of them to staging area
  4. Commit your staged files.

Creating a Git Repository

git-init

Create an empty git repository or reinitialize an existing one

- Creates a new subdirectory named .git that contains all of your necessary repository files
- An initial **HEAD** file that references the HEAD of the master branch is also created.

Creating a Git Repository

Creating a project directory

$ mkdir my-awesome-project
$ cd my-awesome-project

Creating a Git Repository

Initialize a Git Repository using git-init

$ mkdir my-awesome-project
$ cd my-awesome-project

$ git init
Initialized empty Git repository in E:/my-awesome-project/.git/

Nothing here

Repo History

Starting with the Project

Start working on the project. In this example, create a file with some text in it.

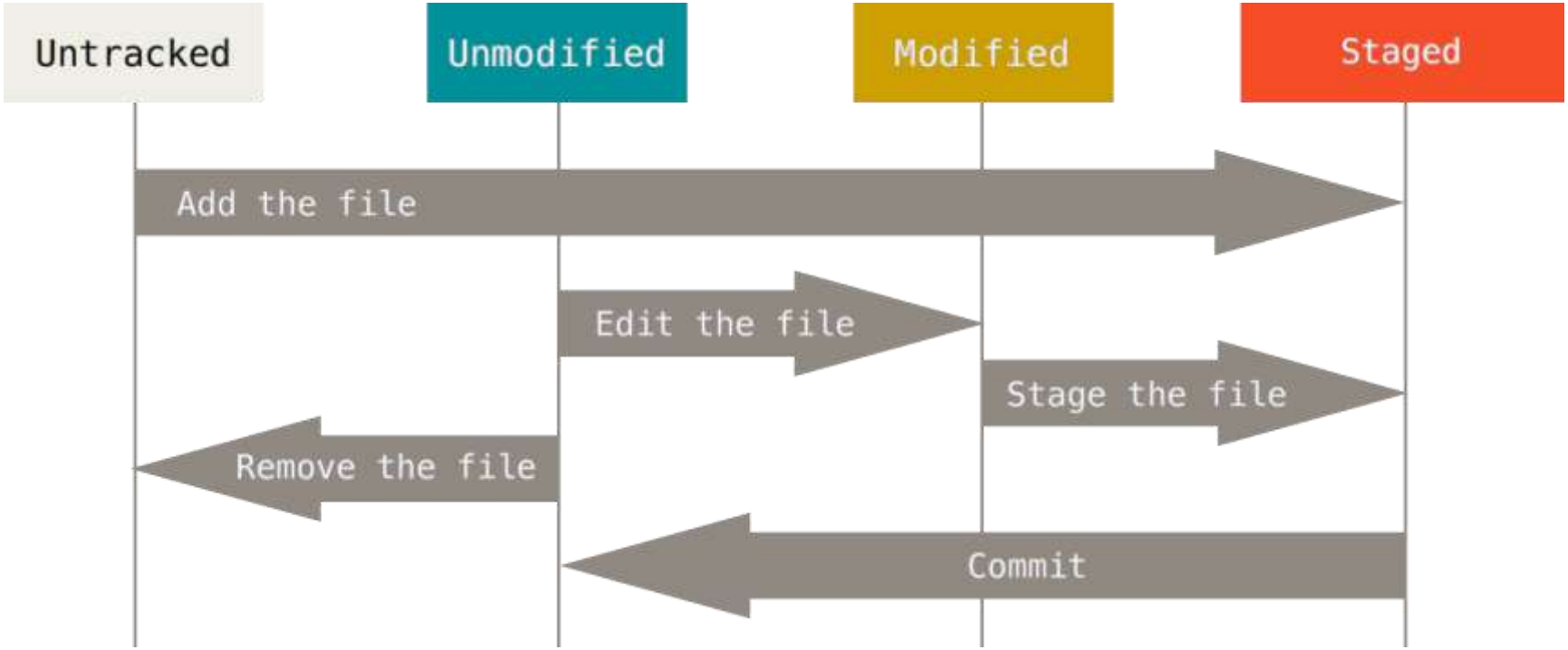
$ mkdir my-awesome-project
$ cd my-awesome-project
$ git init

$ echo “My First File” > first.txt

Nothing here

Repo History

Lifecycle of Status of Files



Check status of Git Repository

git-status

Show the working tree status

- Displays paths that have differences between
  - The index file and the current HEAD commit.
  - The working tree and the index file
  - Paths in the working tree that are not tracked by git

Check status of Git Repository

Check status of your git repository using git-status. See tracked/untracked files and status of modified files.

```
$ mkdir my-awesome-project
$ cd my-awesome-project
$ git init
$ echo “My First File” > first.txt
$ git status
# On branch master
# Initial commit
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#       first.txt
nothing added to commit but untracked files present (use "git add" to track)
```

Nothing here

Repo History

Staging a File

git-add

Add file contents to the index

- Updates the index using the current content found in the working tree, to prepare the content staged for the next commit
- Typically adds the current content of existing paths as a whole

Stage untracked file

Add the untracked file “first.txt” to the stage area using git-add.

```
$ mkdir my-awesome-project
$ cd my-awesome-project
$ git init
$ echo “My First File” > first.txt
$ git status
$ git add first.txt
```

Nothing here

Repo History

Check status again

Check the status of your git repository again and see the difference.

```
$ git init
$ echo “My First File” > first.txt
$ git status
$ git add first.txt
$ git status
# On branch master
# Initial commit
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#       new file:   first.txt
```

Nothing here

Repo History

Commit staged files

git-commit

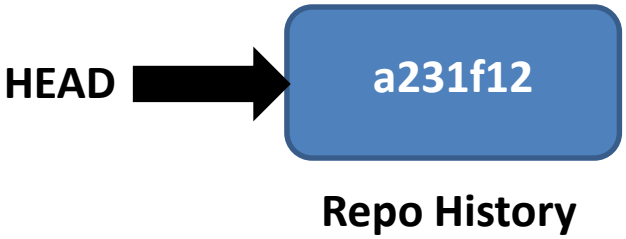
Record changes to the repository

- Stores the current contents of the index in a new commit along with a log message from the user describing the changes.

Stage untracked file

Commit your staged file(s) and check status of the git repository again to observe changes.

$ git add first.txt
$ git commit –m “First Commit”
[master (root-commit) a231f12] First Commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 first.txt
$ git status
 # On branch master
nothing to commit, working directory clean



Viewing Commit Log

git-log

Show commit logs

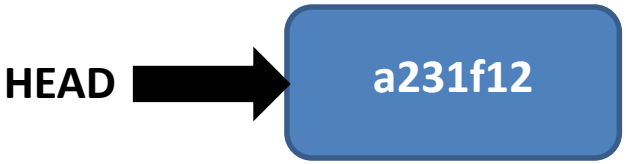
View Commit Log

Use git-log to view commits history and commit messages in chronological order.

$ git commit –m “First Commit”
$ git status

$ git log
commit a231f123102112daf8291894aa404f0c2b8fd5fb
Author: Nilay Binjola <nilaybinjola@gmail.com>
Date: Mon Nov 10 16:26:39 2014 +0530

First Commit

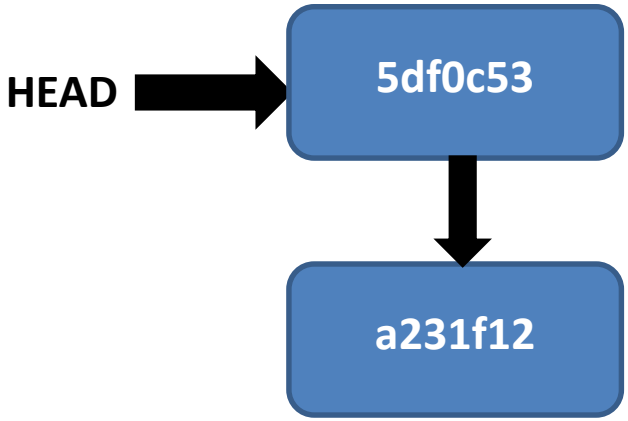


Repo History

Keep working on project

Keep working on project and keep committing work regularly.

```
$ echo “Second File” > second.txt
$ git add .
$ git commit –m “Second Commit”
$ git log
commit 5df0c533cfd8dac626fef959ee9c0b4560ea07c5
Author: Nilay Binjola <nilaybinjola@gmail.com>
Date:   Mon Nov 10 16:36:35 2014 +0530
    Second Commit
commit a231f123102112daf8291894aa404f0c2b8fd5fb
Author: Nilay Binjola <nilaybinjola@gmail.com>
Date:   Mon Nov 10 16:26:39 2014 +0530
    First Commit
```



Repo History

Frequently used commands

git-diff
Show changes between commits, commit and working tree, etc.

git-rm
Remove files from the working tree and from the index.

git-mv
Move or rename a file, a directory, or a symlink.

git-tag
Create, list, delete or verify a tag object signed with GPG.

git-clone
Clone a repository into a new directory.

Undoing things

$ git commit --amend

Used to amend the tip of the current branch. The commit you create replaces the current tip.

$ git reset HEAD <file name>

Unstage a staged file.

$ git checkout -- <file name>

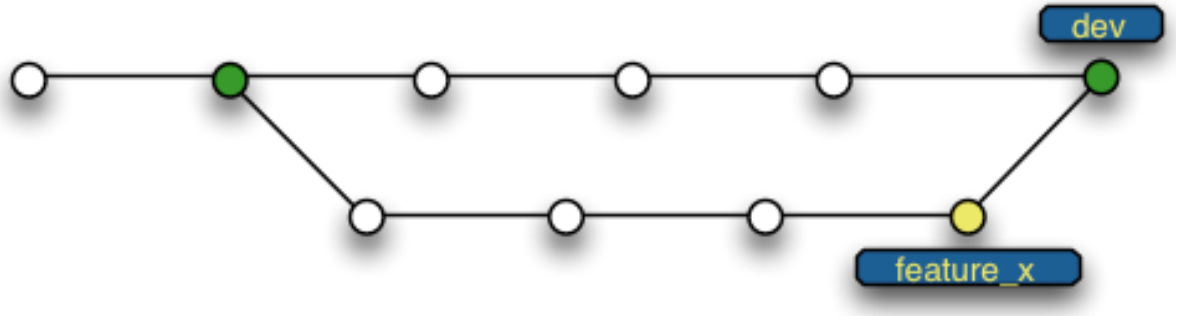
Revert file back to what it looked like when you last committed.

$ git reset –hard <commit>

Will destroy any local modifications. Will revert Working Directory to **commit** status.

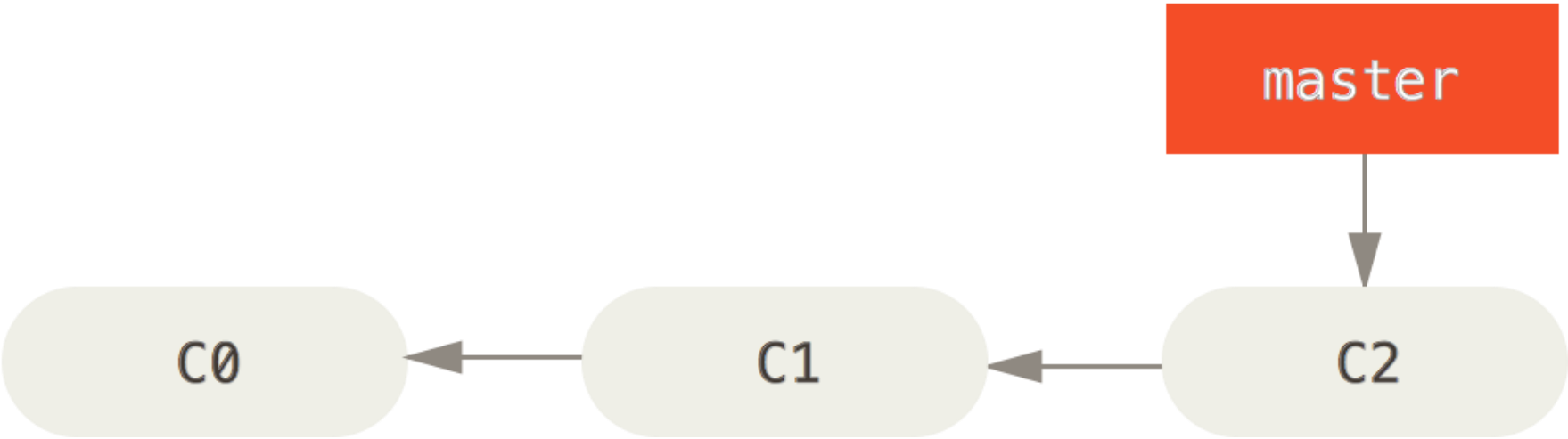
Branching – What is it?

- Diverge from the main line of development and continue to do work without messing with that main line.
- New commits are recorded in the history for the current branch, which results in a fork in the history of the project.
- The **git branch** command lets you create, list, rename, and delete branches.
  - It doesn't let you switch between branches or put a forked history back together again.



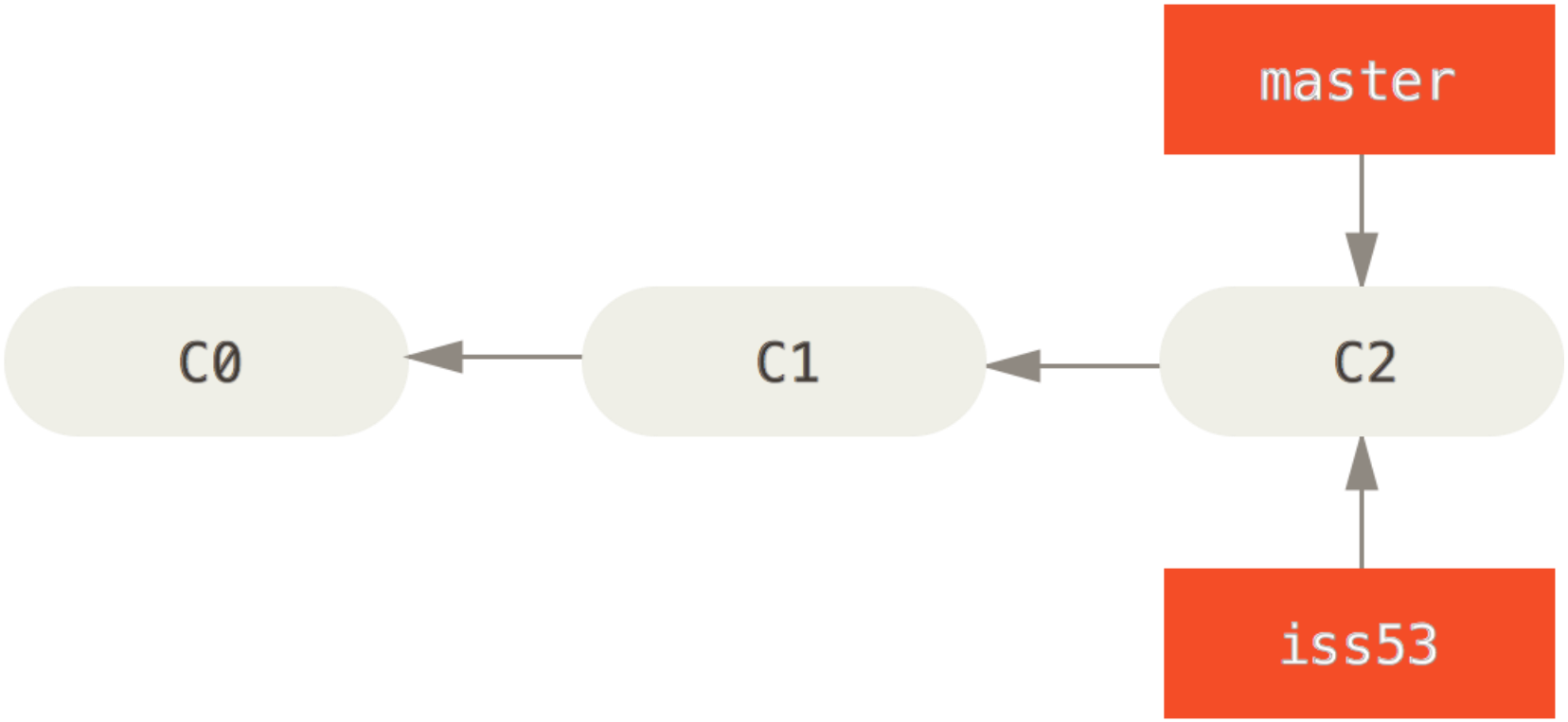
When you want to add a new feature or fix a bug—no matter how big or how small—you spawn a new branch to encapsulate your changes.

Branching



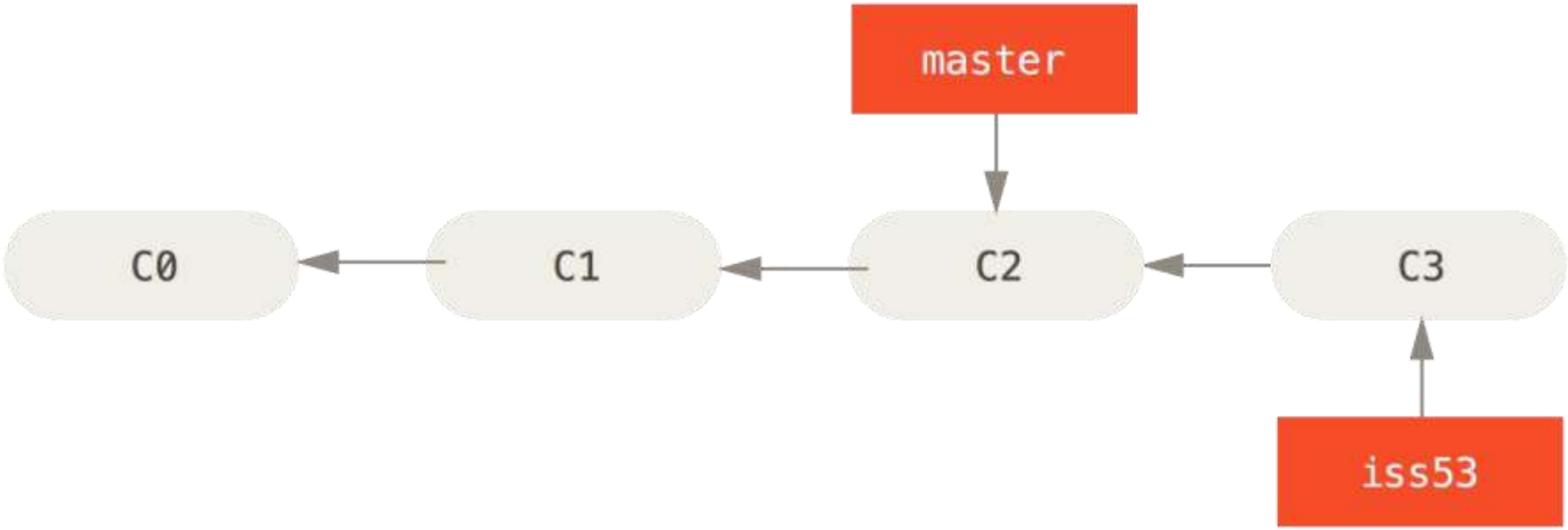
A basic project repository commit tree.

Branching



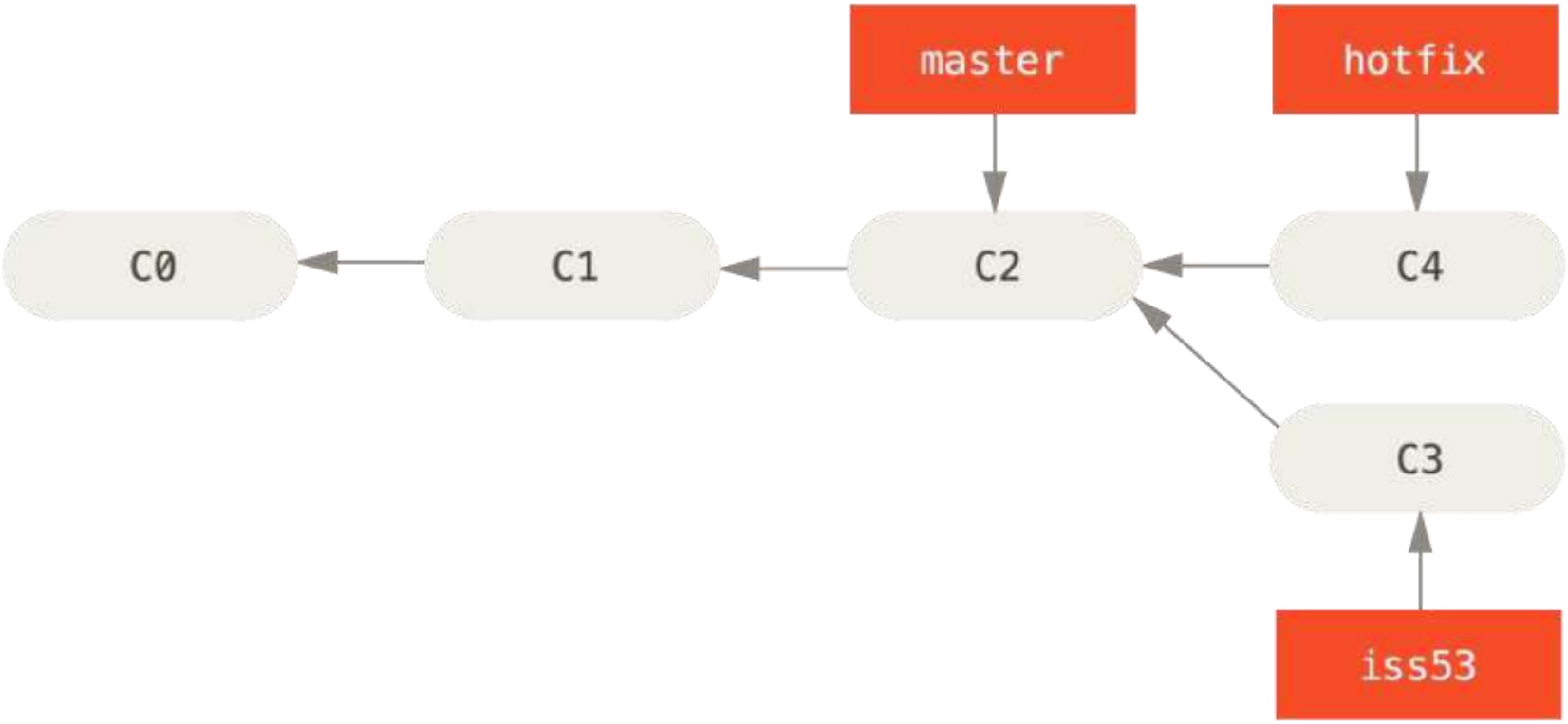
Creating a new branch.

Branching



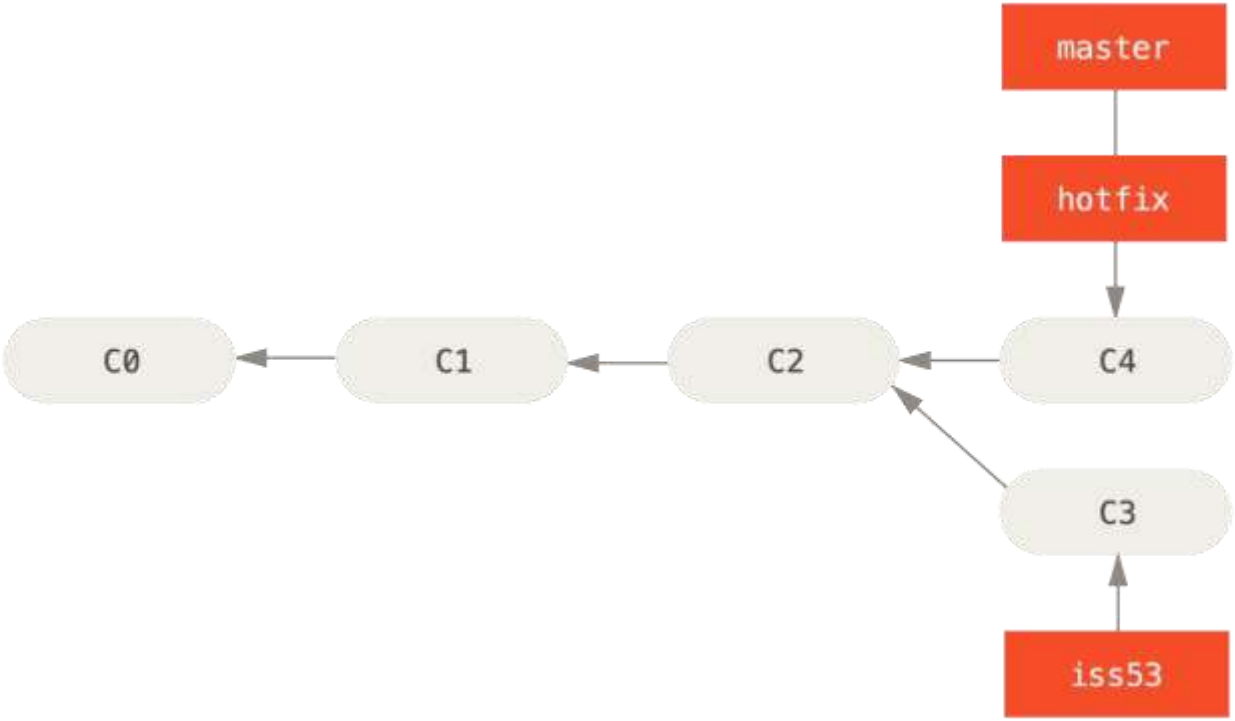
Committing changes to the new branch “iss53”.

Branching



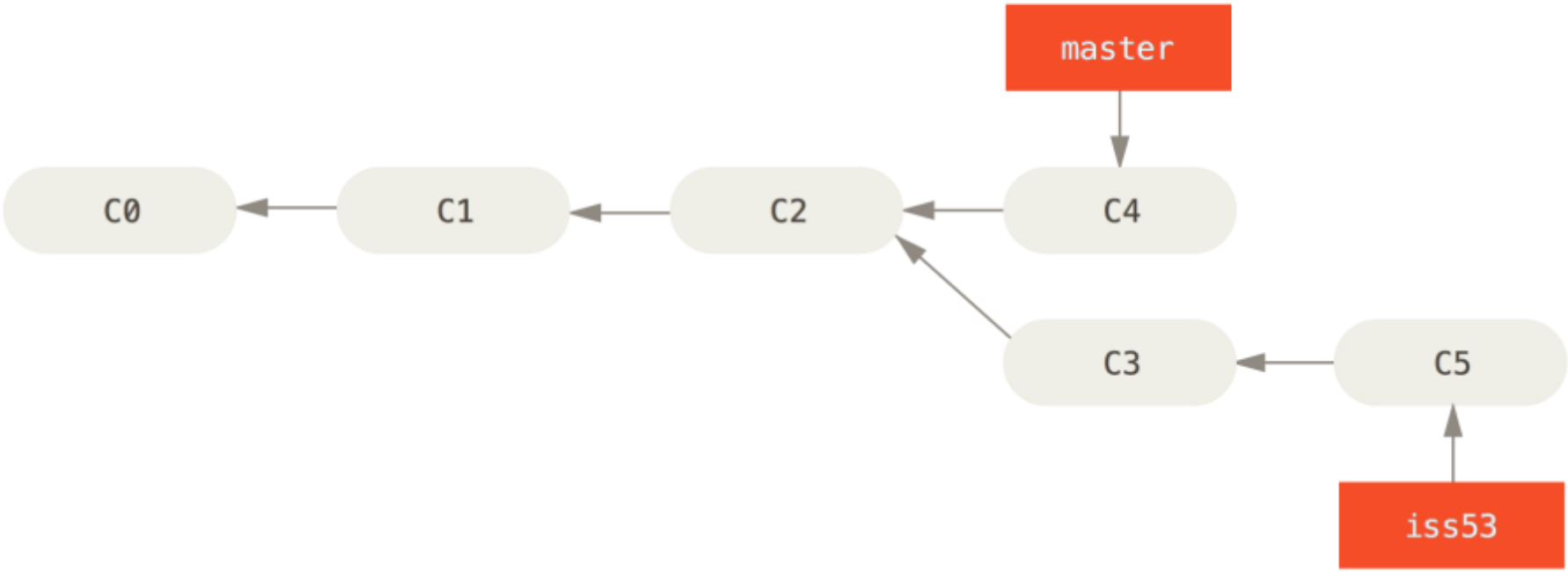
Checking out a new branch from master (after switching to master) and working on it.

Merging



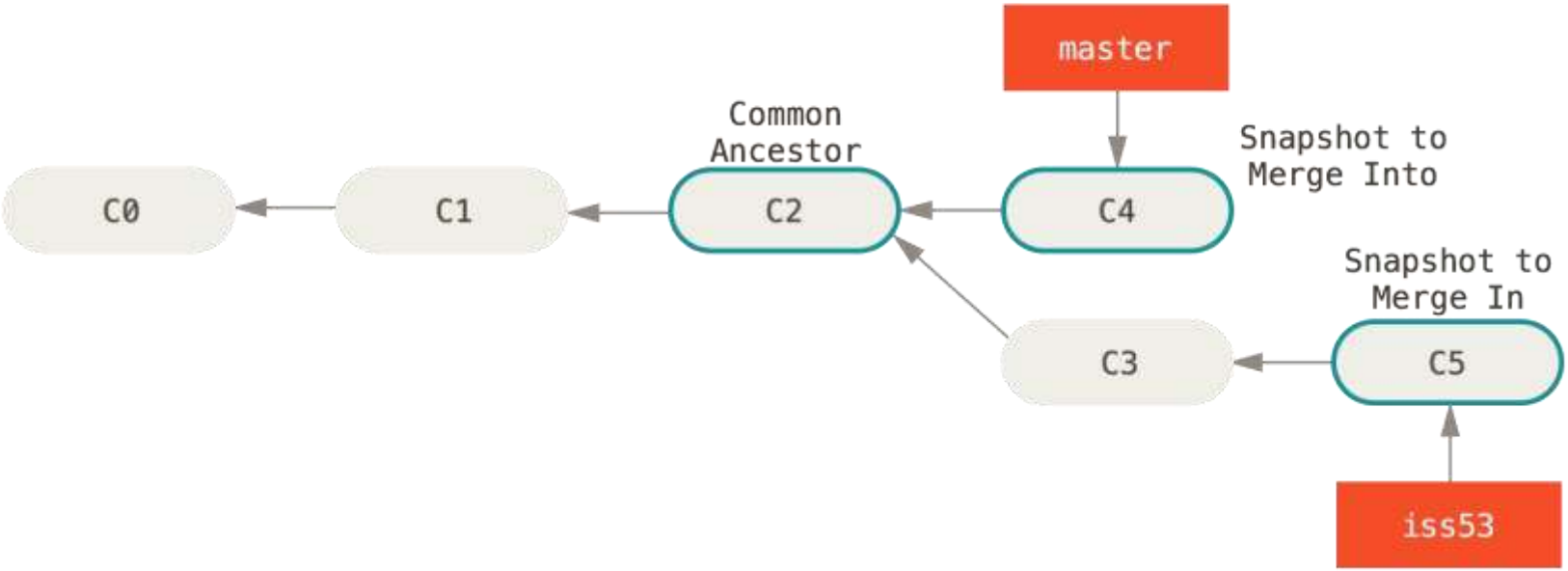
Merging branch “hotfix” with master.

Merging



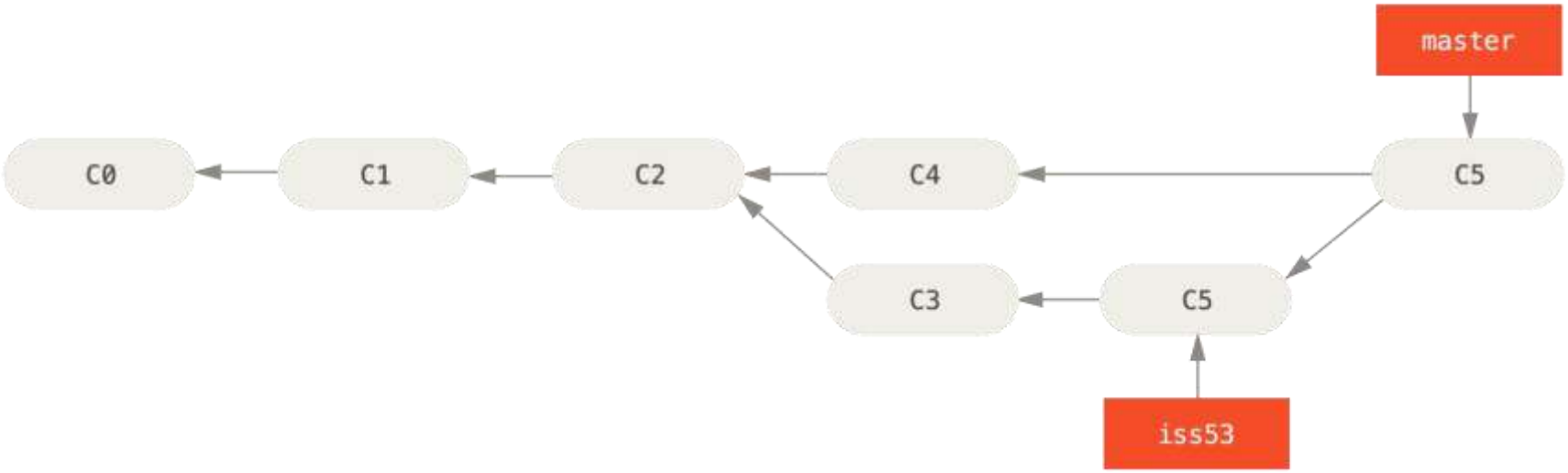
Checking out branch “iss53” and working on it.

Merging



Merging branch “iss53” to branch “master”

Merged



All branches merged. New Commit created.

Merge Conflicts

- **Edit Collision -** If you changed the same part of the same file differently in the two branches you're merging together.
  - **Choose** either one side of the merger or the other.

- **Removed File Conflict -** one person edits a file, and another person deletes that file in their branch.

- Use **git mergetool** for better visualization etc.

Where to go now?

- ✓ Practice using dummy repositories
- ✓ Read on:-
  - ✓ Branching and Merging
  - ✓ Working with Remotes
  - ✓ Tagging
  - ✓ Git Hooks
  - ✓ Distributed Workflows
- ✓ Read the Pro Git Community book on their main website
- ✓ Fun online git tutorial - https://try.github.io/levels/1/challenges/1
- ✓ Start using Git for your projects hosted on Github, BitBucket.

"What Would CVS Not Do?“ – Linus Torvalds