Advance Git
by Rajesh Kumar

About me

Rajesh Kumar

DevOps Architect

@RajeshKumarIN | www.RajeshKumar.xyz

History

- Source Code Control Systems (SCCS)
  - 1972, Closed Source, free with Unix
- Revision Control System (RCS)
  - 1982, Open Source
- Concurrent Version System (CVS)
  - 1986-1990, open source
- Apache Subversion (SVN)
  - 2000, Open Source

BitKeeper SCM

- 2000, closed source, proprietry
- Distributed version control
- “community version” was free
- used for source code of the Linux Kernal from 2002-2005
- Controversial to use proprietary SCM for an open source project
- April – 2005 – The community version not free any more

Git is born

- April 2005
- Created by Linus Torvalds
- Replacement for bitKeeper to manage Linux Kernal source code

Git is Popular

- Distributed Version Control
- Open source and free software
- Compatible with Unix-like Systems (Linux, Mac OSX, and Solaris) and Windows
- Faster than other SCMs (100x in some cases)

Git is a hit

- Explosion in Popularity
- No official statics
- GitHub is launched in 2008 to host Git repositories
  - 2009: over 50,000 repositories, over 100,000 users
  - 2011: over 2 million repository's, over 1 million users

Programmers and Developers

- HTML, CSS, JavaScript
  - PHP, Ruby, Ruby on railes, Perl Python, ASP
  - Java, C, C++, C#, Objective C
  - Action Script, Coffee Script, Haskell, Scala, Shell Scripts
- Not as useful for tracking non-text files
  - Image, movies, music, fonts
  - Word processing files, spreadsheets, PDFs

The Git Repository

- .git directory
  - **Config – Repo private configuration file (.ini style)**
  - Description – Repo description
  - Hooks/* - hooking scripts
  - **Objects/* - The object repository**
  - **Refs/heads/* - branches (like “master”)**
  - **Refs/tags/* - tags**
  - **Refs/remotes/* - tracking others**
  - Logs/* - logs
  - Index – changes to commit
  - HEAD – points to one of the branches (the “current branch”, where commits go)

Objects

- Every object has a SHA1 to uniquely identify it
- Objects consist of 4 types:
  - Blobs (the contents of a file)
  - Trees (directories of blobs or other trees)
  - Commits
    - A Tree
    - Plus zero or more parent commits
  - Tags
    - An object (usually a commit)

A SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long.SHA-1 produces a 160-bit (20-byte) hash value

Configuring Git

- System
  - /etc/gitconfig
  - Program file\git\etc\gotconfig
- User
  - ~/.gitconfig
  - $HOME\.gitconfig
- Project
  - my_project/.git/config

Configuring Git…

- System
  - git config --system
- User
  - git config --global
- Project
  - git config

Configuring Git…

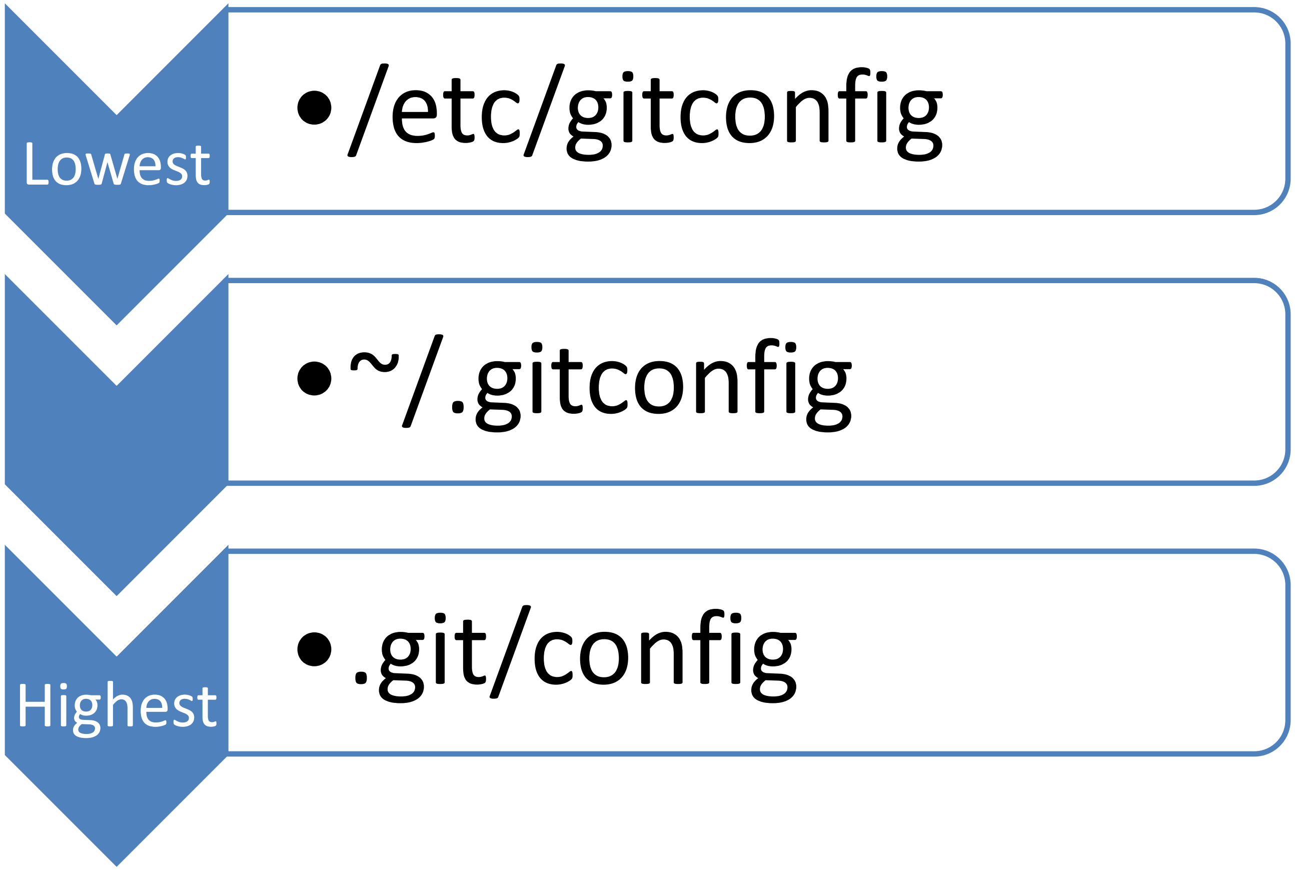
- git config --global user.name "rajesh kumar"
- git config --global user.email someon@nowehre.com
- git config --list

- more .gitconfig
- git config --global core.editor "vim“
- git config --global color.ui true

Configuring Git…

- git config --list
- git config --global section.key
- git config --global section.subsection.key
- git config --global user.name "rajesh kumar"
- git config --global user.email
- git config --global core.editor "vim“
- git config --global color.ui true

git Config Priorities

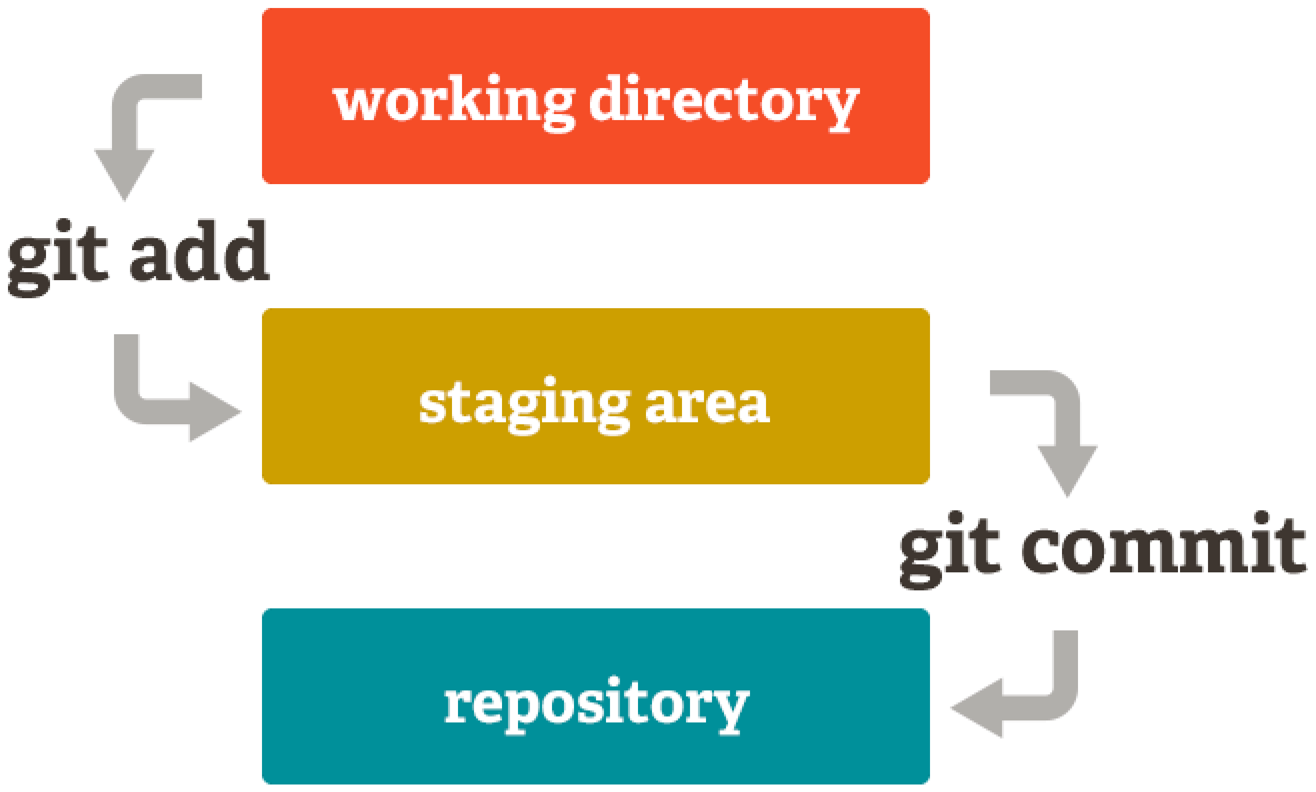


- /etc/gitconfig
- ~/.gitconfig
- .git/config

git workflow

- Working with local repos
  - git init
    - Creates a .git in the current dir
  - git add <directory tree>
    - Adds all files (except .git)
  - git commit
    - Commits the changes (in this case initial commit)
    - Creates a branch named **master**
    - HEAD points at master
- Working with Remote Repos
  - git clone
    - Creates a git repo from an existing repo
    - All remote branches are tracked
    - Remote HEAD branch checked out as your initial master branch as well
  - git pull
  - git push

Git workflow



Head

- Pointer to “tip” of the current branch in repository
- Last state of repository, what was last checked out
- Points to parent next commit
  - Where writing commits takes place

SCM Operations

- Bootstrap
  - Init
  - Checkout
  - Switch branch
- Modify
  - Add, delete, rename
  - Commit
- Information
  - Status
  - Diff
  - Log
- Reference
  - Tag
  - Branch
- Collaborate
  - Clone
  - Pull, fetch
  - push

Git Help

git help   (list of common commands)

git <command> -h

- Manual page

```
man git <command>
git help <command>
git <command> --help
```

git add .

git add . only adds what is there, not what has been deleted (if tracked).

git add .

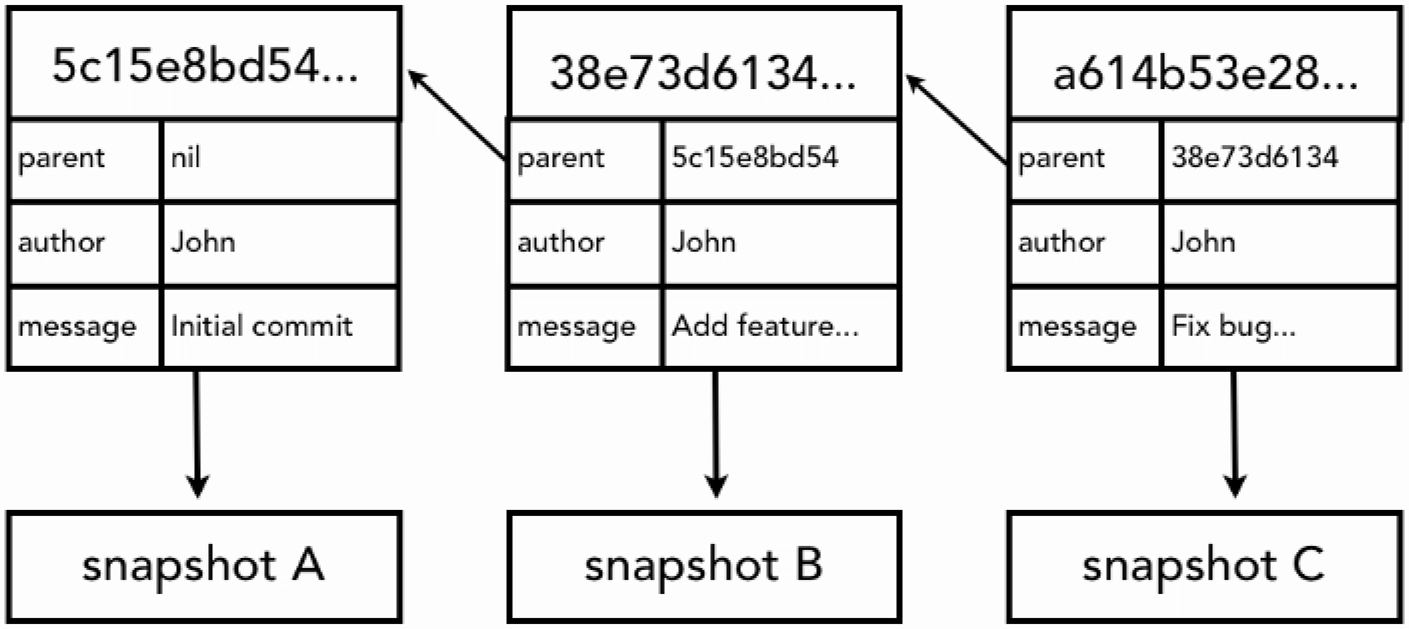
git add -A would take care of both steps...

Referring to commits

When we submit any changes to the repository,

- Git generate a checksum for each change set
  - Checksum algorithm convert data into a simple number
  - Same data always equals same checksum
- Data Integrity is fundamental
  - Changing data would change checksum
- Git uses SHA-1 hash algorithm to create checksums
  - 40 character hexadecimal string (0-9,a-f)
  - Example: 1837f0b7056c64cef103210b07539b6313612ba3

referring to commits



Commit Message best practices

- Short single-line summary (less than 50 characters)
- Optionally followed by a blank line and a more complete description
- Keep each line to less than 72 characters
- Write a commit messages in present tense, not past tense
  - “fix bugs” or “fixes bug”, not “fixed bug”

Commit Message best practices

- Bullets points are usually asterisks or hypens
- Can add “ticket tracking numbers” from bugs or support requests
- Can develop shorthand for your organization
  - “[css,js] ”
  - “bugfix: ”
  - “#24223 - ”

Commit Message best practices

- Be clear and descriptive
  - Bad: “Fix typo”
  - Good: “Add missing > in project section of HTML
  - Bad: “Update login code”
  - Good: “Change user authentication to use Blowfish”
  - Bad: “Updates member report, we should discuss if this is right next week”

git log

```
commit 1837f0b7056c64cef103210b07539b6313612ba3
Author: rajesh kumar <someon@nowehre.com>
Date:   Thu Dec 6 01:16:03 2012 -0800

    first commit
```

```
git log --oneline
git log --oneline --graph
git log --format=short == git shortlog
git log file1 file2 dir3
```

Show changes only for listed files or subdirs

git log

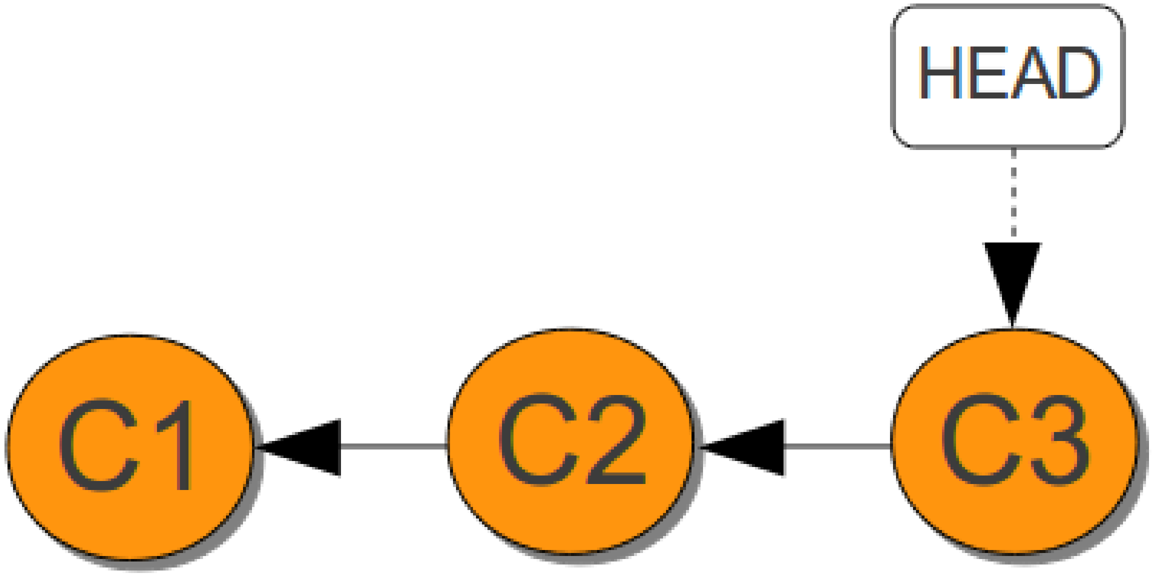
- git log –n 1/2/3/0
- git log --since=2012-05-05
- git log --until=2012-04-23
- git log --grep=“init”

See the log pof remote repos
git fetch origin
git log origin/master
Eq. to
git log HEAD..origin/master

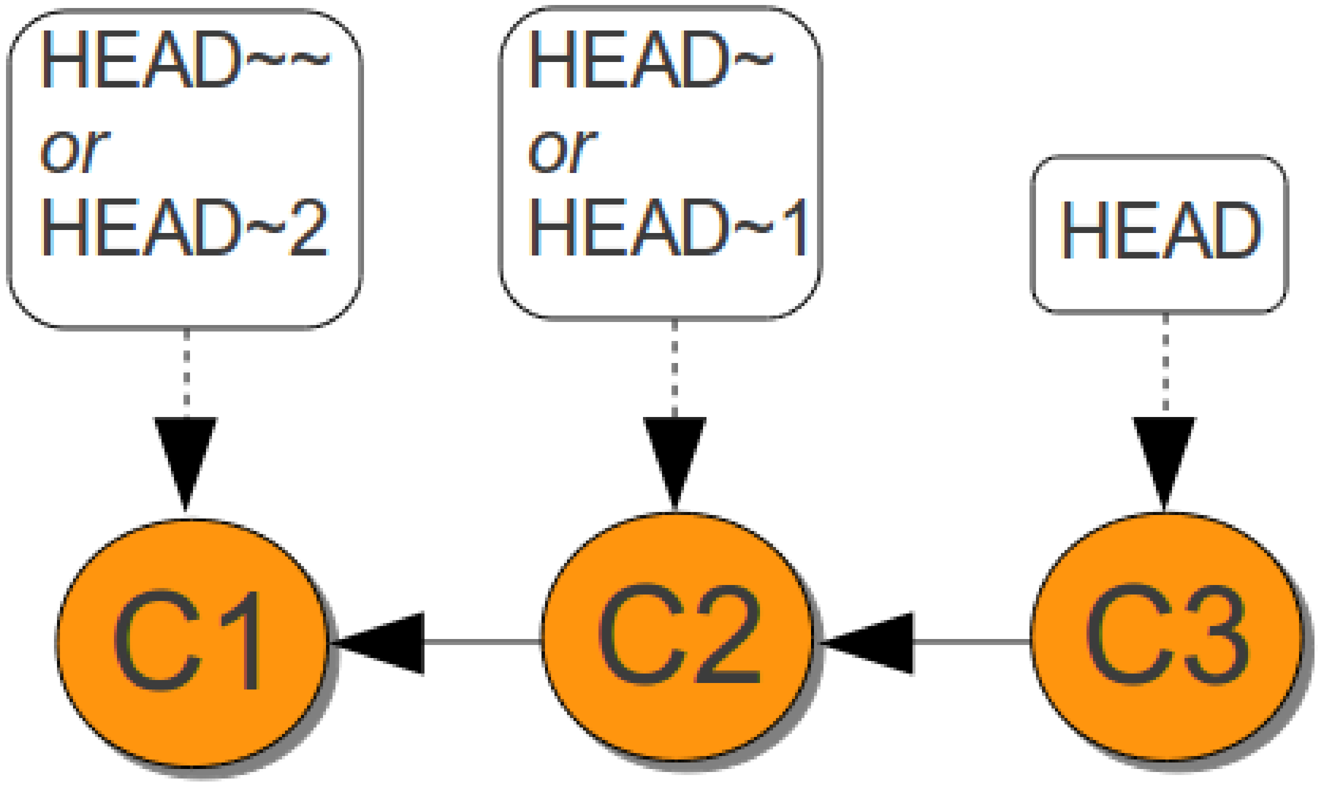
Head

- Pointer to “tip” of the current branch in repository
- Last state of repository, what was last checked out
- Points to parent next commit
  - Where writing commits takes place

HEAD



HEAD



HEAD Commands

```
> git log HEAD
```

shows all the commits reachable from the current HEAD

```
> git show
```

The git show command reports the changes introduced by the most recent commit:

```
> git show HEAD~
> git show HEAD~2
```

Behind The Scenes: Where Is The HEAD?

The contents of the git HEAD variable is stored in a text file in the .git/HEAD:

```
> cat .git/HEAD
ref: refs/heads/master
```

That is telling us that we need to look at the file refs/heads/master in the .git directory to find out where HEAD points:

Undoing the changes

- git checkout
- git revert – Undo the commits with new commits.
- git clean - removes untracked files from your **working directory**.
- git reset - It modifies the index (the so-called "**staging area**") Or it changes which commit a branch head is currently pointing at.

git checkout

The git checkout command serves three distinct functions:

1. checking out files,
2. checking out commits, and
3. checking out branches.

git checkout

```
> git checkout <commit> <file>
```

Check out a previous version of a file. This turns the <file> that resides in the working directory into an exact copy of the one from <commit> and adds it to the **staging area**.

git checkout

```
> git checkout <commit>
```

Update all files in the working directory to match the specified commit.

git checkout

```
> git checkout master
```

git revert

The git revert command undoes a committed snapshot. But, instead of removing the commit from the project history, it figures out how to undo the changes introduced by the commit and appends a new commit with the resulting content.

This prevents Git from losing history, which is important for the integrity of your revision history and for reliable collaboration.

git clean

```
> git clean -n
```

Perform a “dry run” of git clean.

```
> git clean -f
```

Remove untracked files from the current directory.

```
> git clean -df
```

Remove untracked files *and* untracked directories from the current directory.

git reset

git revert is a “safe” way to undo changes, whereas git reset as the dangerous method.

When you undo with git reset(and the commits are no longer referenced by any ref or the reflog), there is no way to retrieve the original copy—it is a permanent undo.

In Practice

It can be used to remove committed snapshots, although it’s more often used to undo changes in the **staging area** and the **working directory**. In either case, it should only be used to undo *local* changes—you should **never** reset snapshots that have been shared with other developers.

git reset

```
> git reset <file>
```

Remove the specified file from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.

git reset

> git reset

Reset the staging area to match the most recent commit, but leave the working directory unchanged. This unstages *all* files without overwriting any changes, giving you the opportunity to re-build the staged snapshot from scratch.

git reset

```
> git reset --hard
```

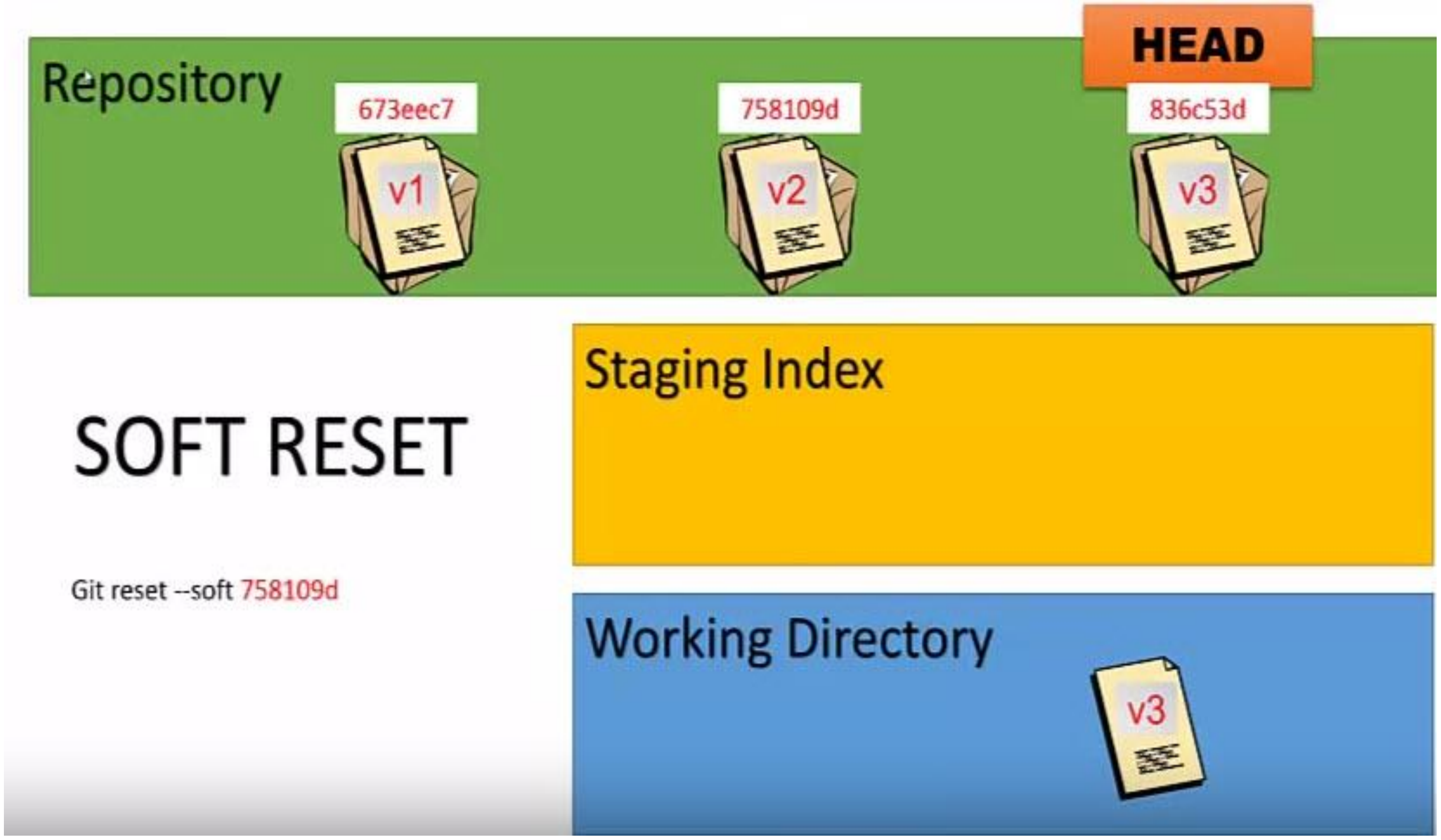
Reset the staging area and the working directory to match the most recent commit. In addition to unstaging changes, the --hard flag tells Git to overwrite all changes in the working directory, too.

git reset
A dangerous method

Three kinds of reset

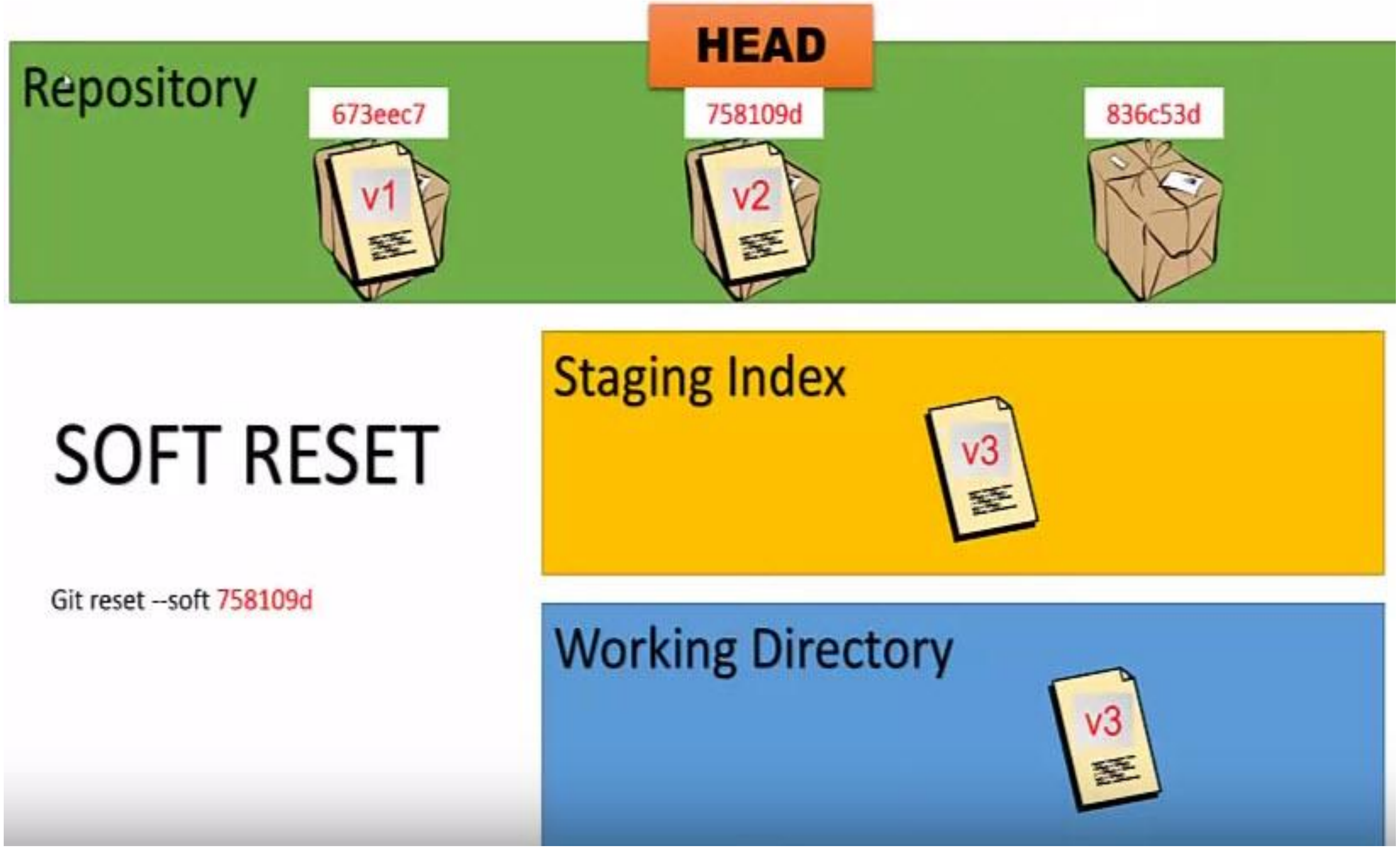
1. soft
   - Moves the head
   - git reset --soft 34dghsd
2. Mixed
   - Changes HEAD and Staging Index
   - git reset –mixed eb45juh
3. Hard
   - Reverts completely back to commit
   - git reset --hard fg4533df

SOFT RESET



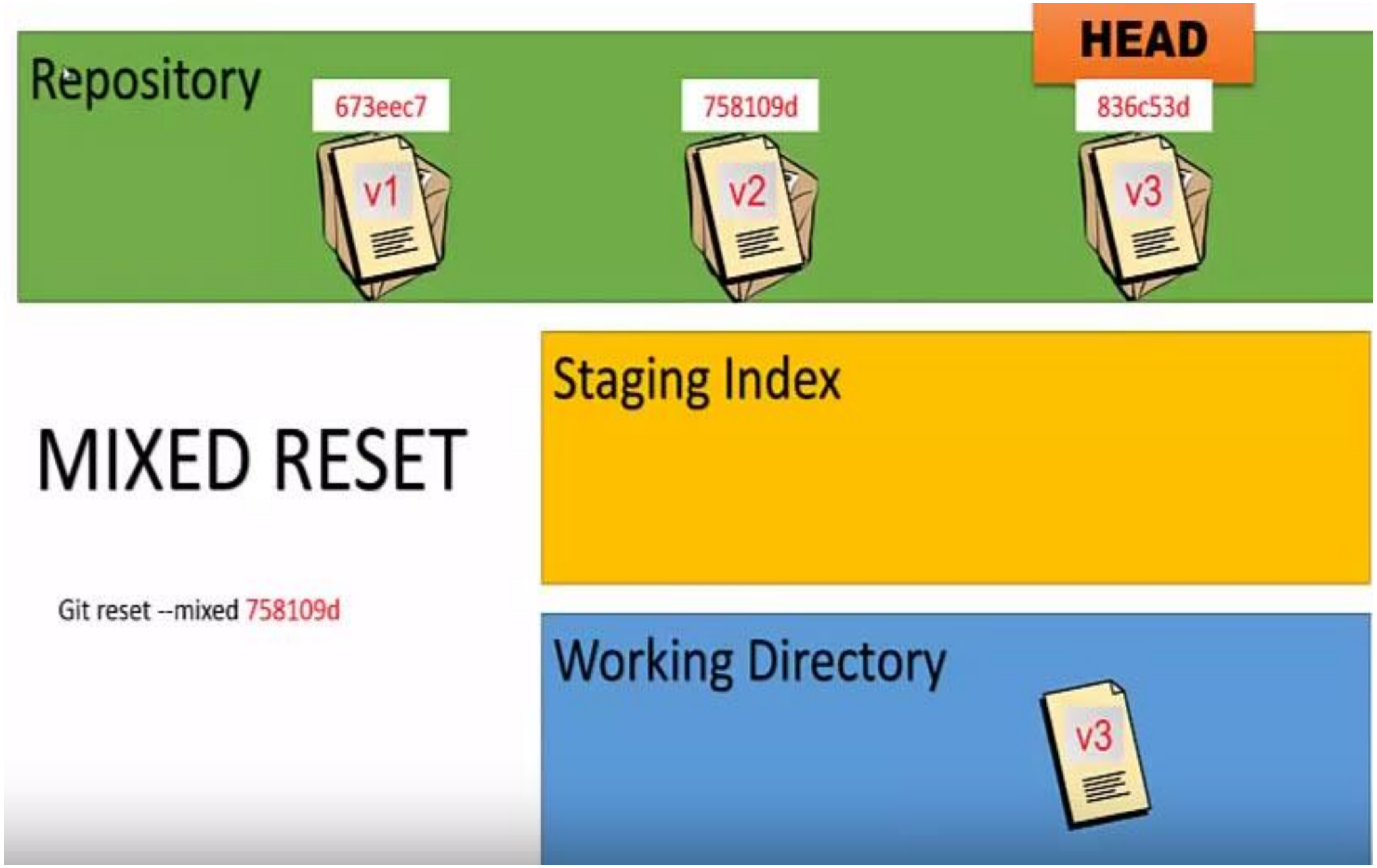
```
Git reset --soft 758109d
```

SOFT RESET



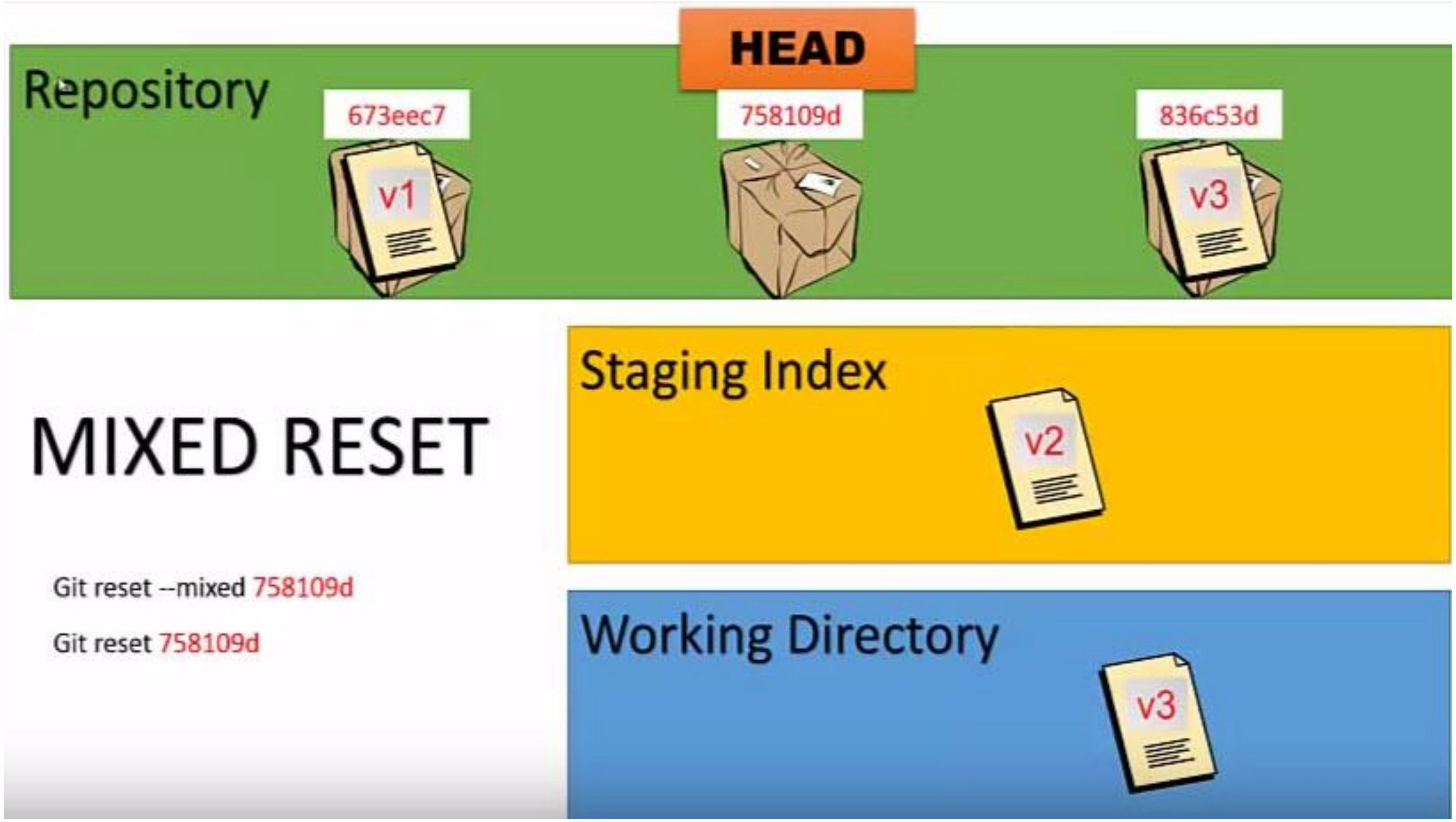
```
Git reset --soft 758109d
```

MIXED RESET (Default)



Git reset --mixed 758109d

MIXED RESET (Default)

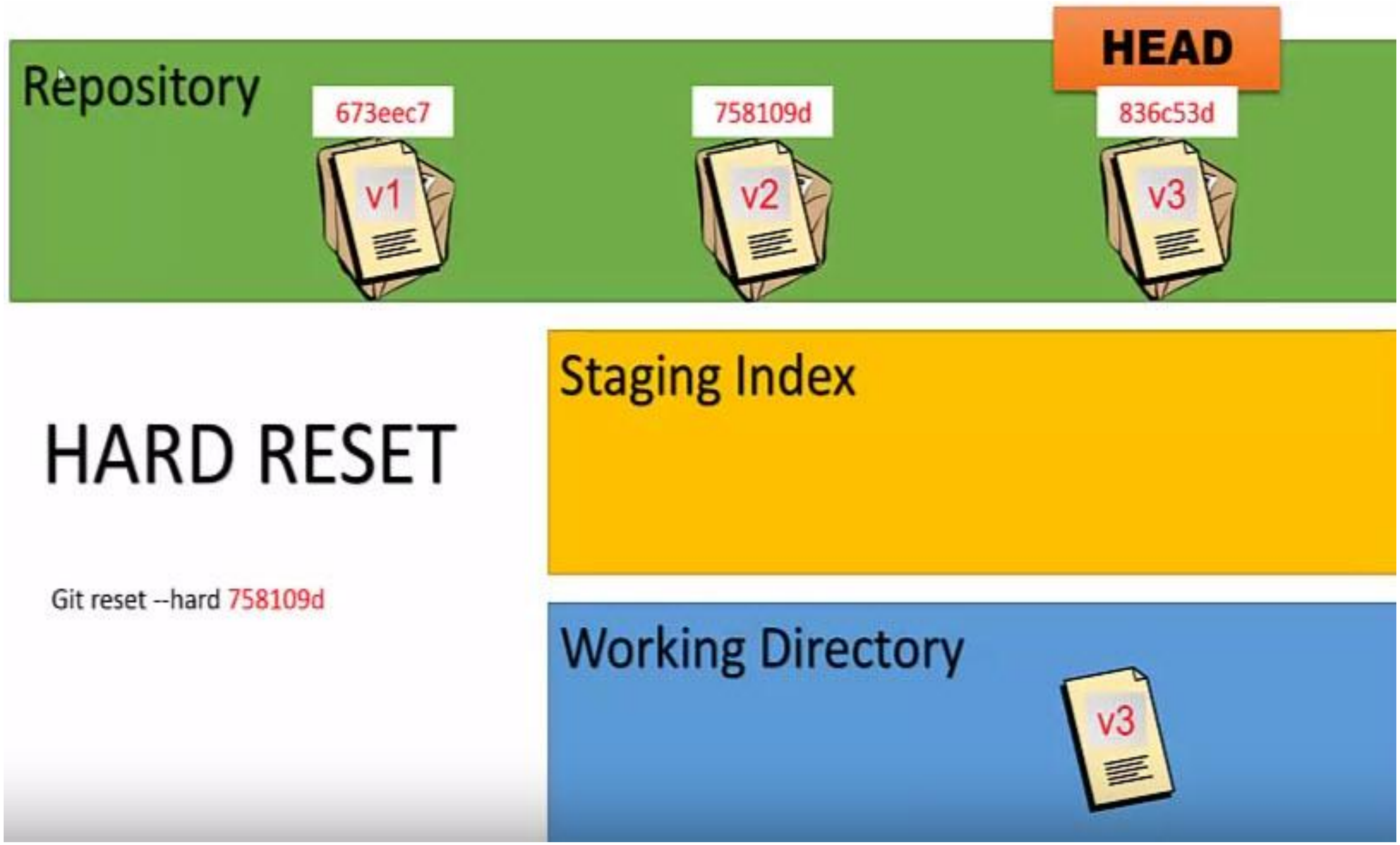


MIXED RESET

Git reset --mixed 758109d

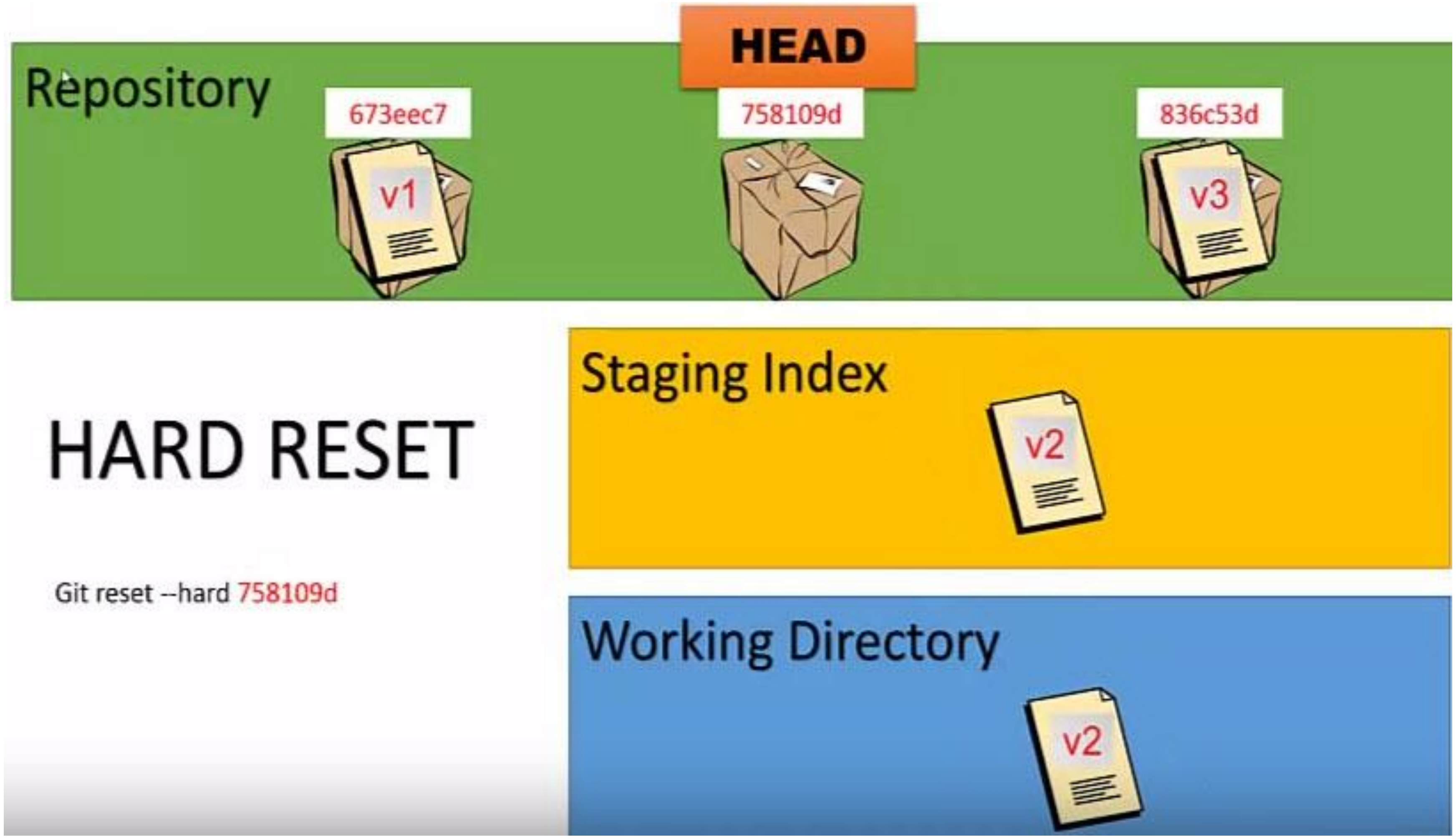
Git reset 758109d

HARD Reset



Git reset --hard 758109d

HARD Reset



```
Git reset --hard 758109d
```

Renaming the file

```
> git mv <filename>
> git commit -m"renaming file"
```

Deleting the file

```
git rm file1.txt
git commit -m "remove file1.txt"
```

git diff

- git diff SHA1…SHA2
- git diff HEAD~1..HEAD
- git diff HEAD~1..
- git diff branch1 branch2
- git diff anotherbranch
- git diff --staged

git branching

- git checkout –b
- git branch <branchname>
- Find .git/refs – To knw the branches
- Delete Branch
  - git branch -D <branchname>

git branching

- git checkout –b = git branch + git checkout

git merge

```
> git merge <branch>
```

Working with remote repository

- Scenario 1 – Start from Local repository
- Scenario 2 – Start from Remote repository
- Scenario 3 – Mixed approach

Remotes Repository

- git remote – List of remotes connections
- git remove -v “
- git remote add <alias> <url>
- git remote rm <alias>
- git remote rename <old-name> <new-name>

Start from Existing repository

- git init
- git remote add <name> url
- git push

Start from Remote repository

- git clone
- git push
- git pull (or git fetch + git merge)

Mixed approach

- git init
- git remote add <name> url
- git pull (or git fetch + git merge)
- git push

get fetch

git-fetch - Download objects and refs from another repository

- Fetch before work
- Fetch before you push
- Fetch often

- git fetch
- git fecth origin

git pull

== git fetch + git merge

Pull Vs Clone

- **Clone**: Get a working copy of the remote repository.
- **Pull**: I am working on this, please get me the new changes that may be updated by others and merge it.

Appendix - Delete remote depots

- git push origin nontracking:nontracking
- git push origin :nontracking
- git push origin –delete nontracking

.gitignore

Appendix - git reflog

Questions

- How Git store the changes?
- How to Commit Single file?
- I could not understand git log output?
- How Git workflow works?
- what is git reset?
- How to undo the changes?
- Clone Vs Pull Vs Fetch
- How to delete Local branch?
- How to delete Remote Branch?
- What is bare git repo

Delete remote depots

- git push origin nontracking:nontracking
- git push origin :nontracking
- git push origin –delete nontracking

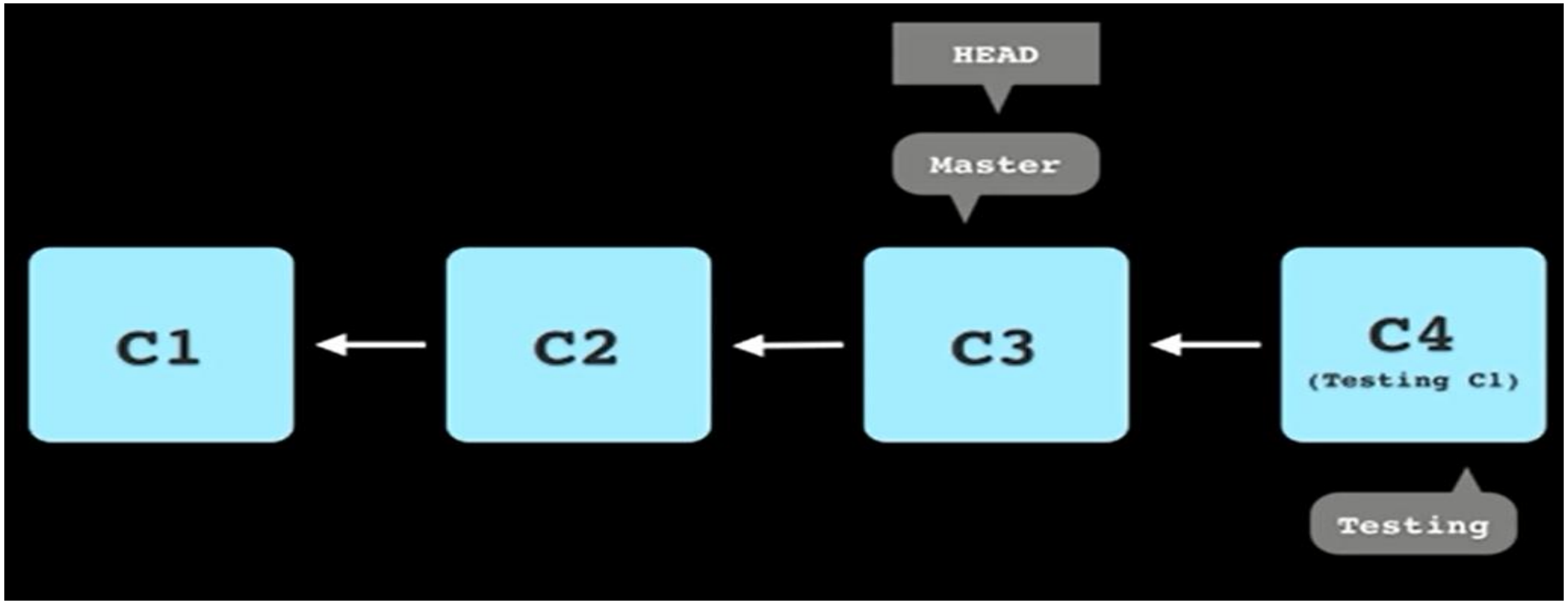
Appendix - Pull Vs Rebase

- Pull → fetch + merge with master
- Rebase -> fecth + Checkout and make master of master + merge with local

Types of Merge

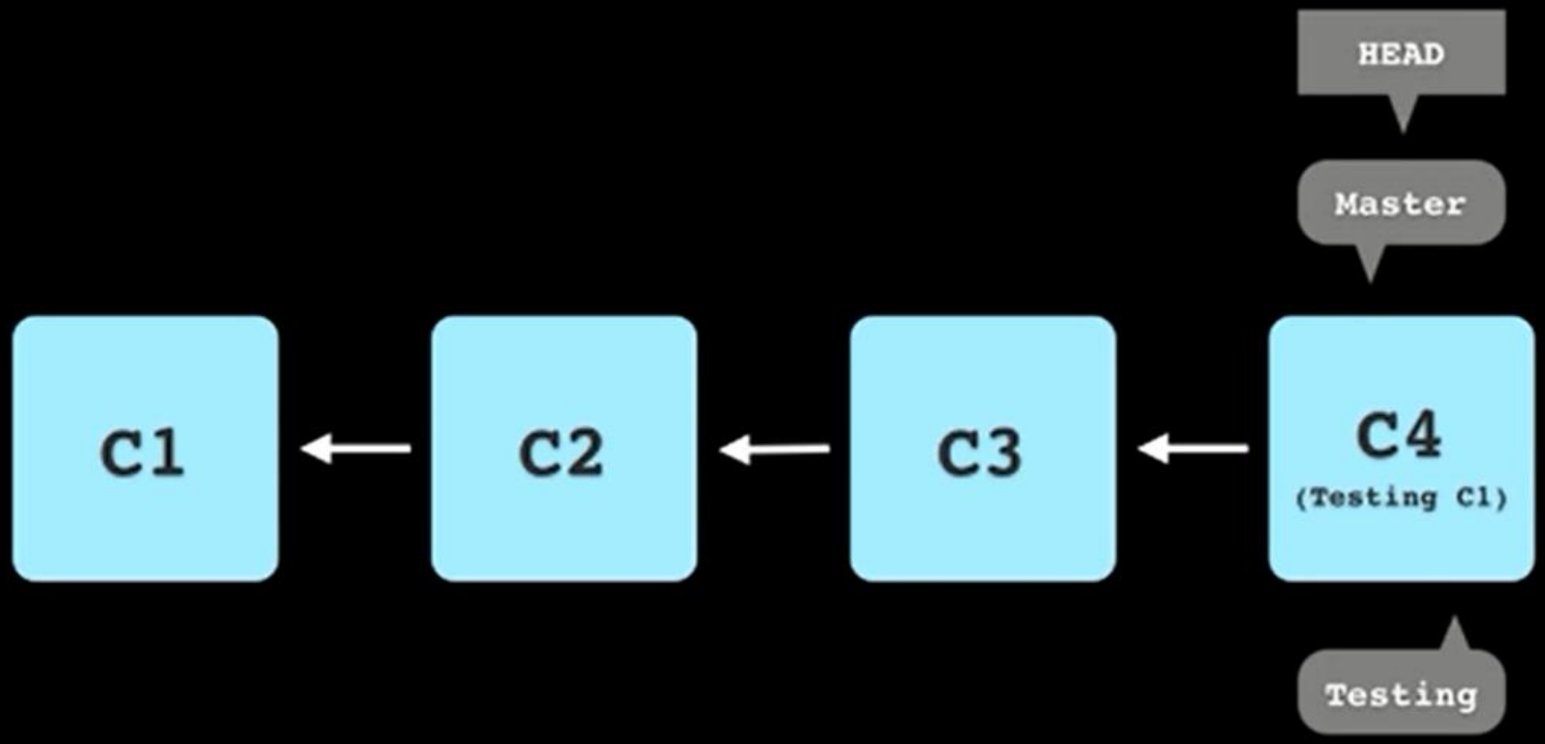
- Fast Forward Merge
- Recursive Merge aka 3 Way merge

Fast Forward Merge

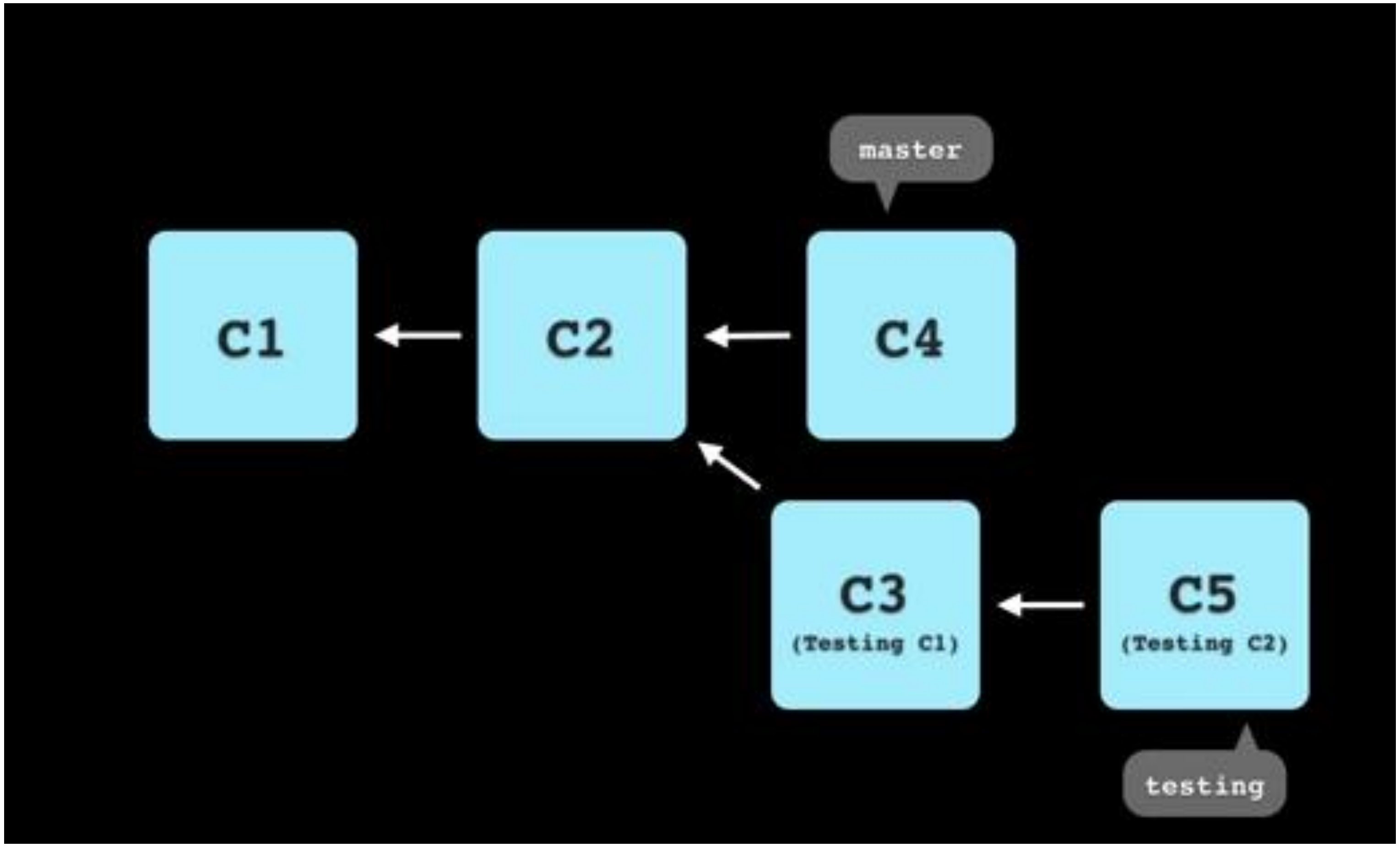


> No New Commits on Master

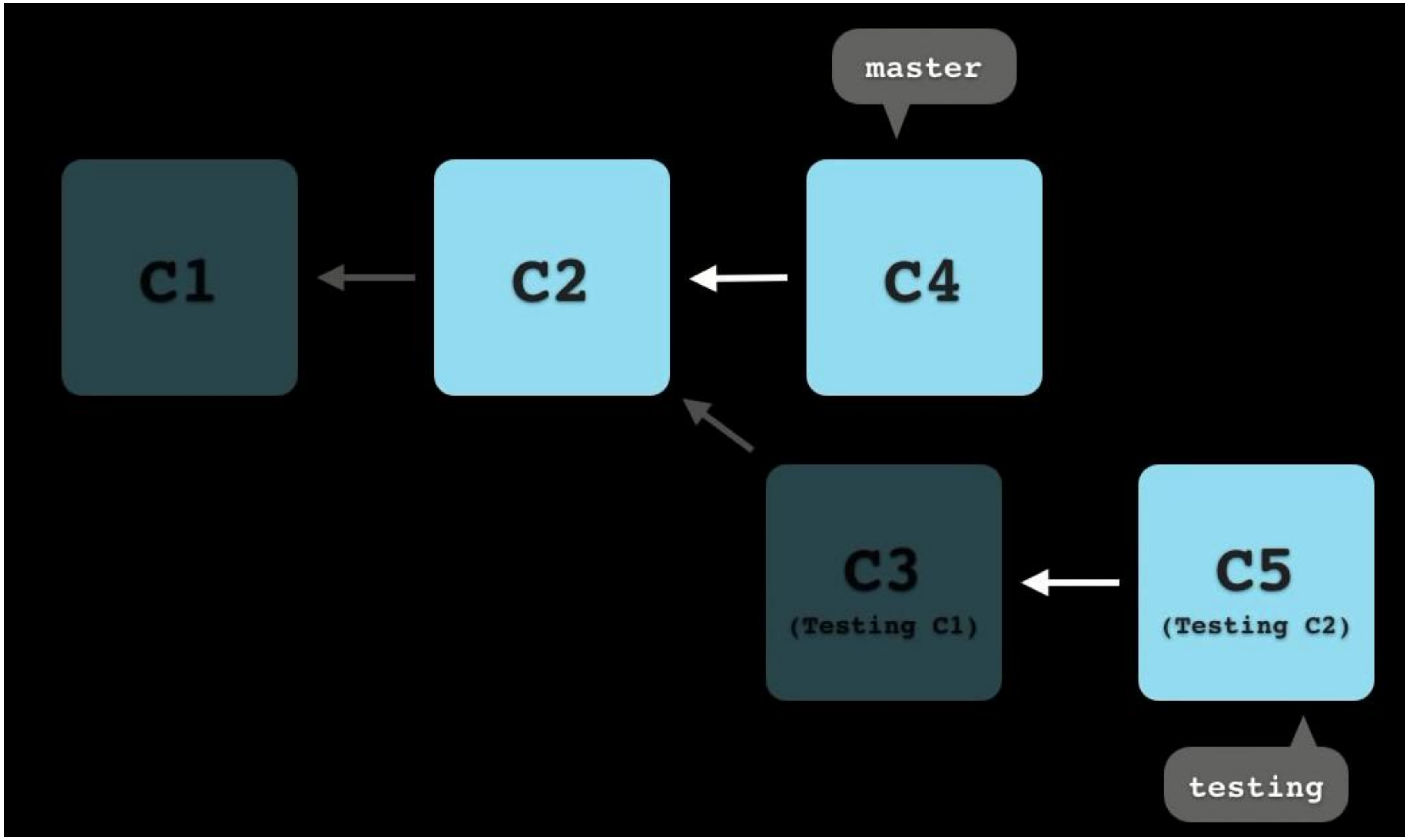
Fast Forward Merge



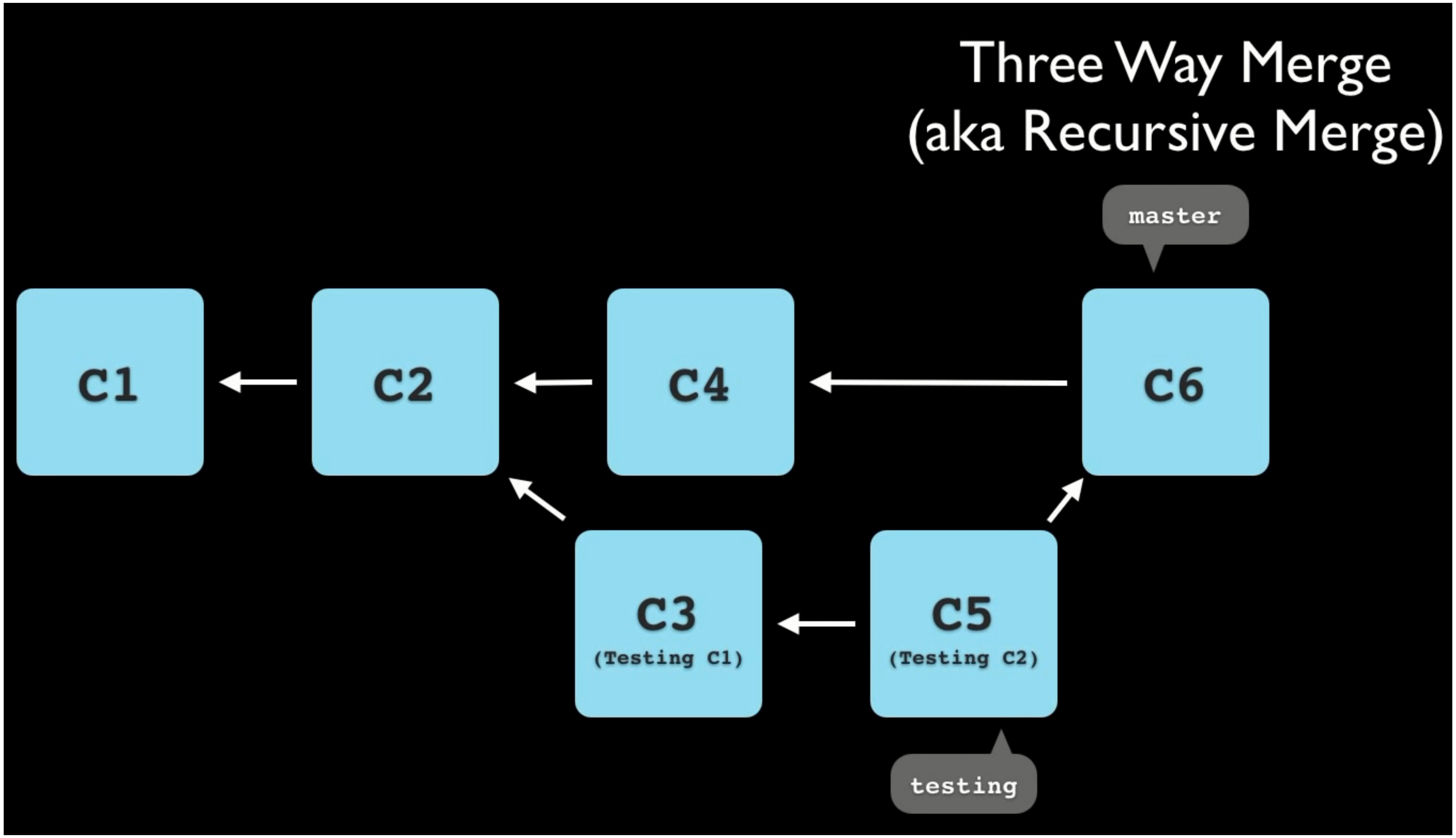
Three way merge=recursive merge



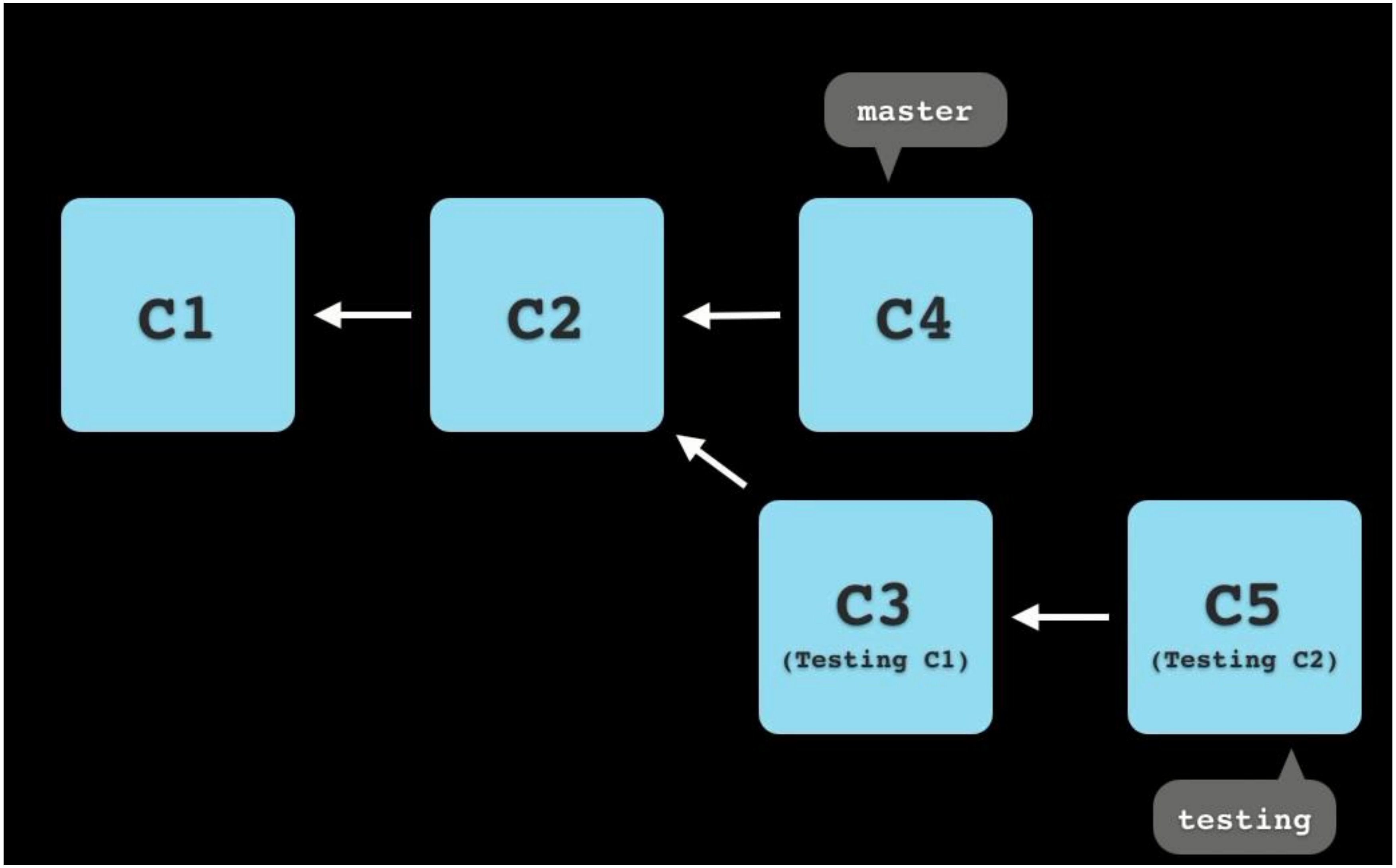
Three way merge=recursive merge



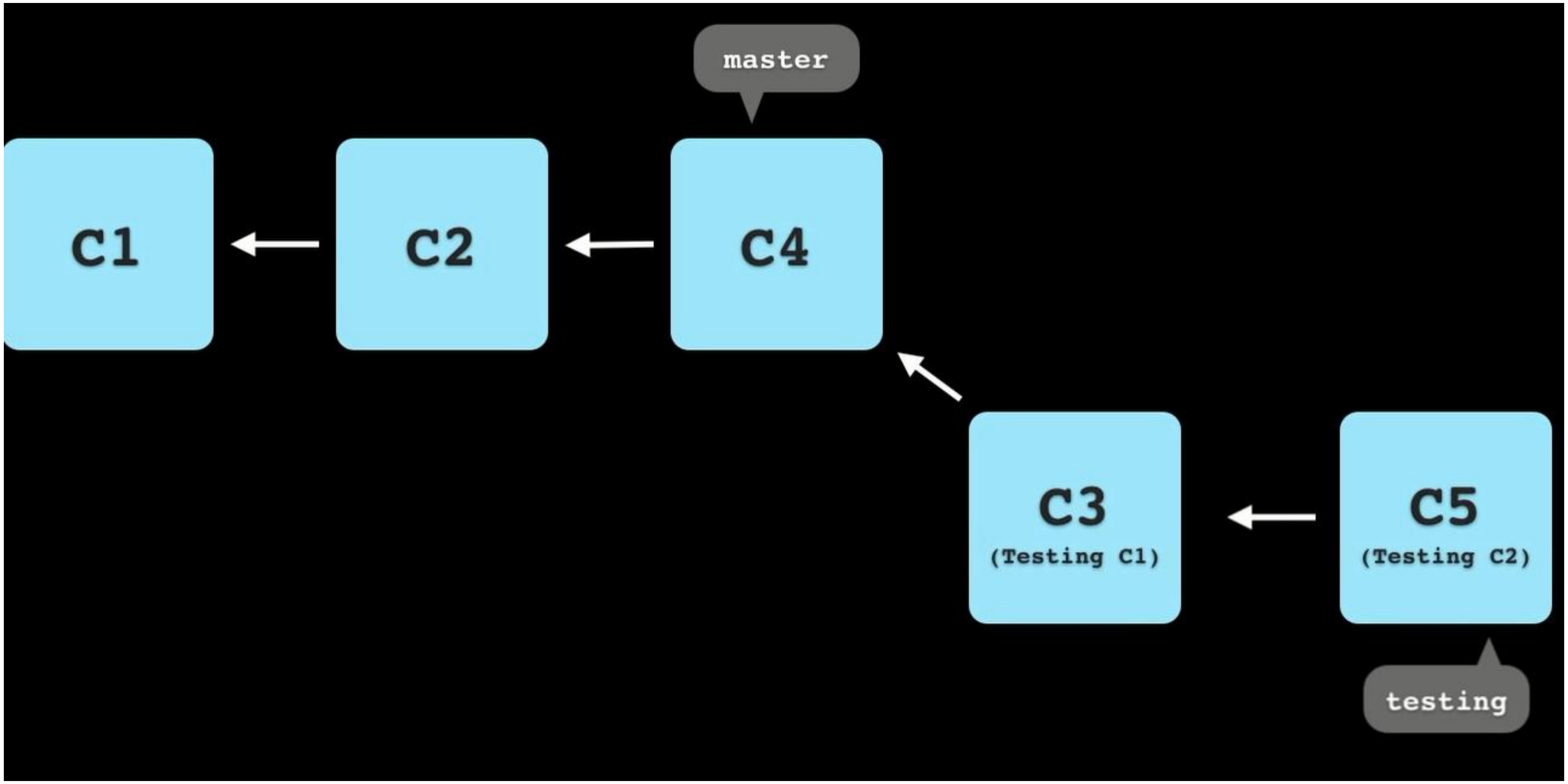
Three way merge

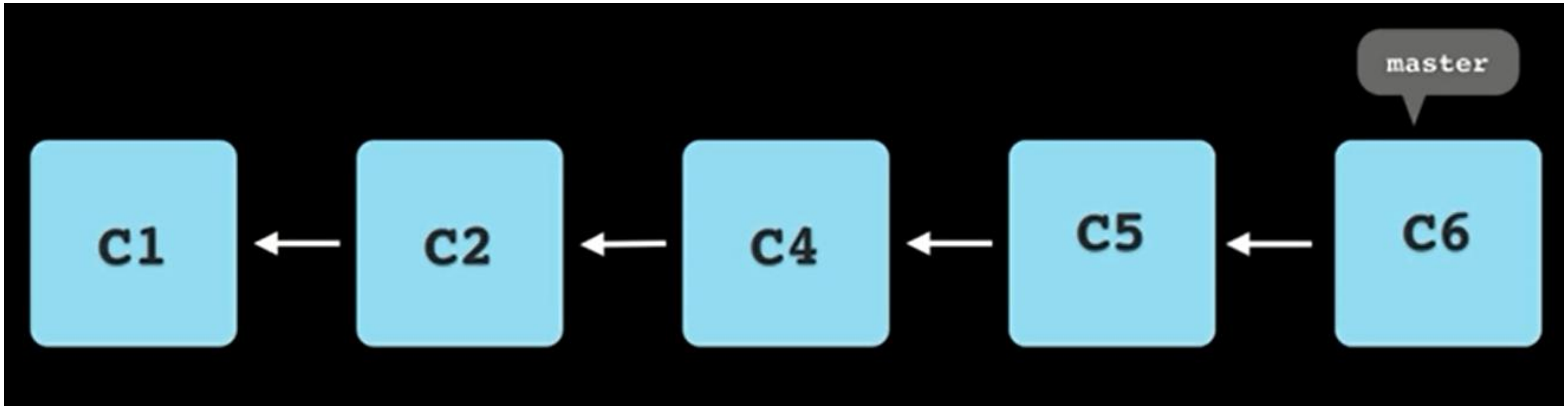


Rebasing

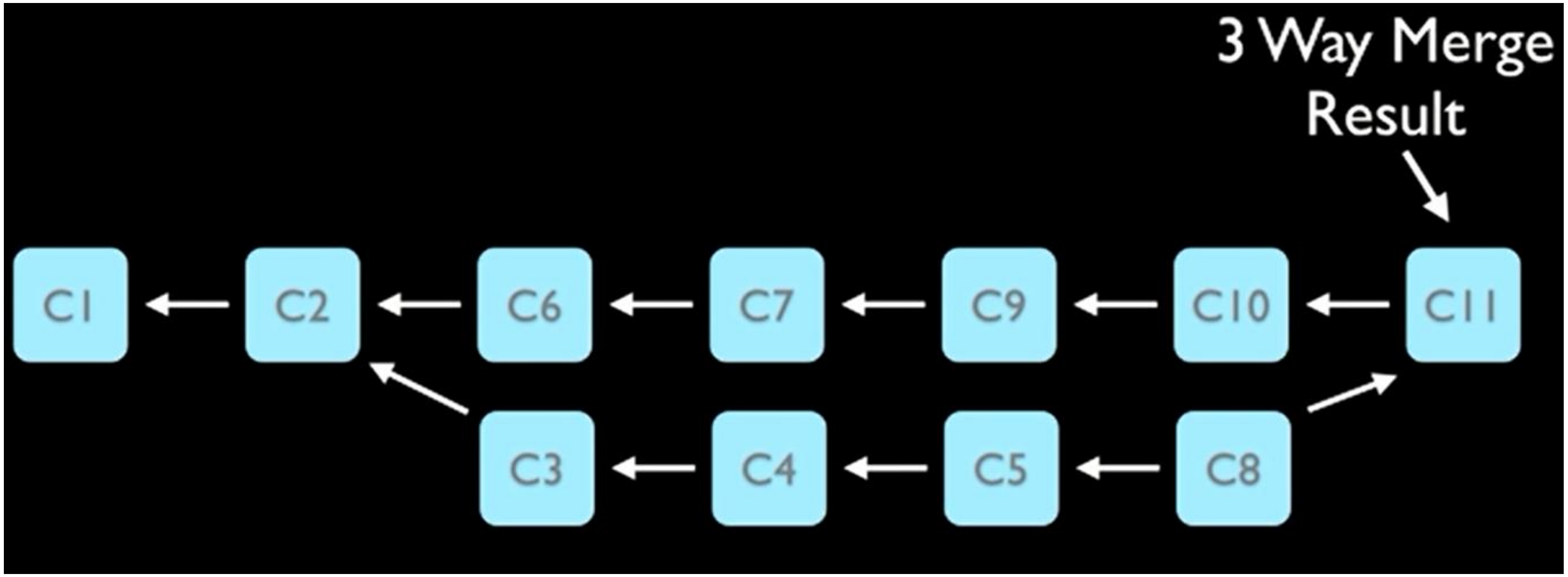


Rebase

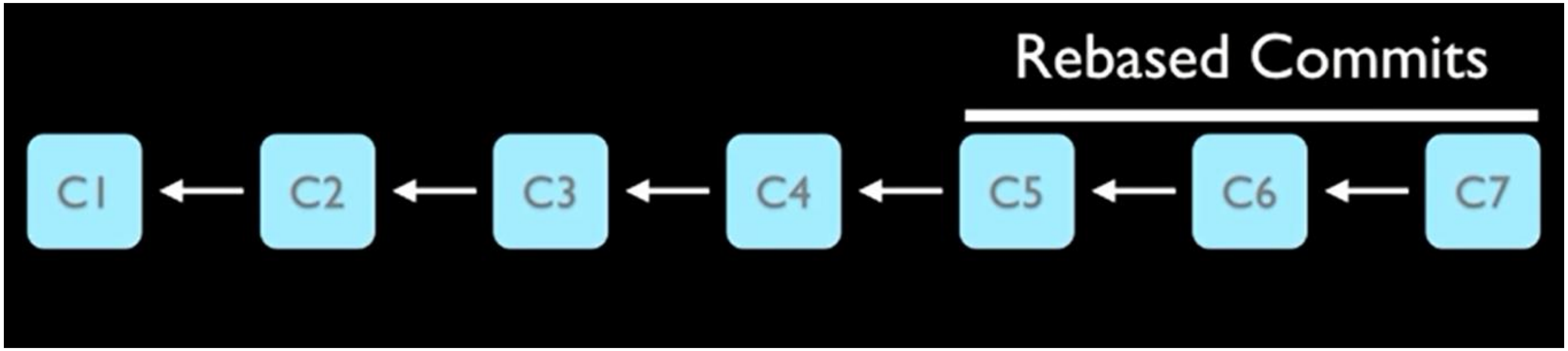



master
C1
C2
C4
C5
C6

Merged without rebase



Merged with rebase



Git rebase testing branch

- git rebase master
- git checkout master
- git merge testing

DO NOT

- Never rebase a branch which you made public available

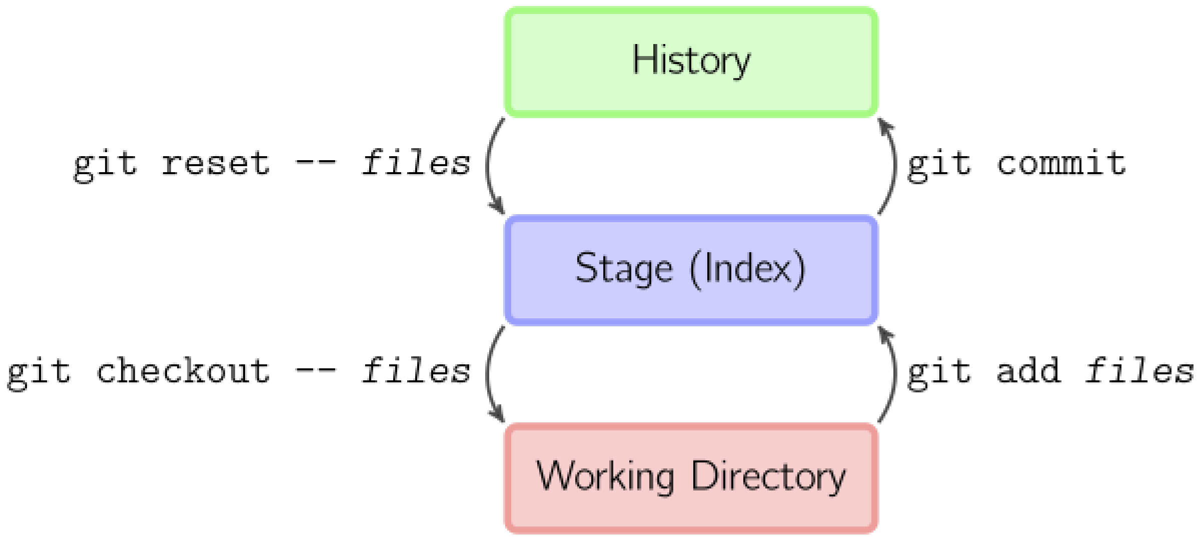
Rebase

- Rewrites commits as if they started in a different place
- Breaks SHA1s: commits are lost
  - Don’t rebase if you have published commits
- git rebase master topic2
  - All changes in topic2 but not in master are rewritten as changes to the new master
- May result in merge conflicts
  - git rebase –continue or –abort or –skip
- git rebase –i (interactive) is helpful
- When rebased, merge is a fast forward
  - git checkout master; git merge topic2

Reference

- http://git-scm.com/
- http://gitref.org/
- www.git-tower.com
- https://github.com/
- http://www.git-tower.com/
- http://git-scm.com/book
  https://www.youtube.com/watch?v=ZDR433b0HJY

Appendix



History
git reset -- *files*
git commit
Stage (Index)
git checkout -- *files*
git add *files*
Working Directory