Version Control Systems.

Version Control Systems

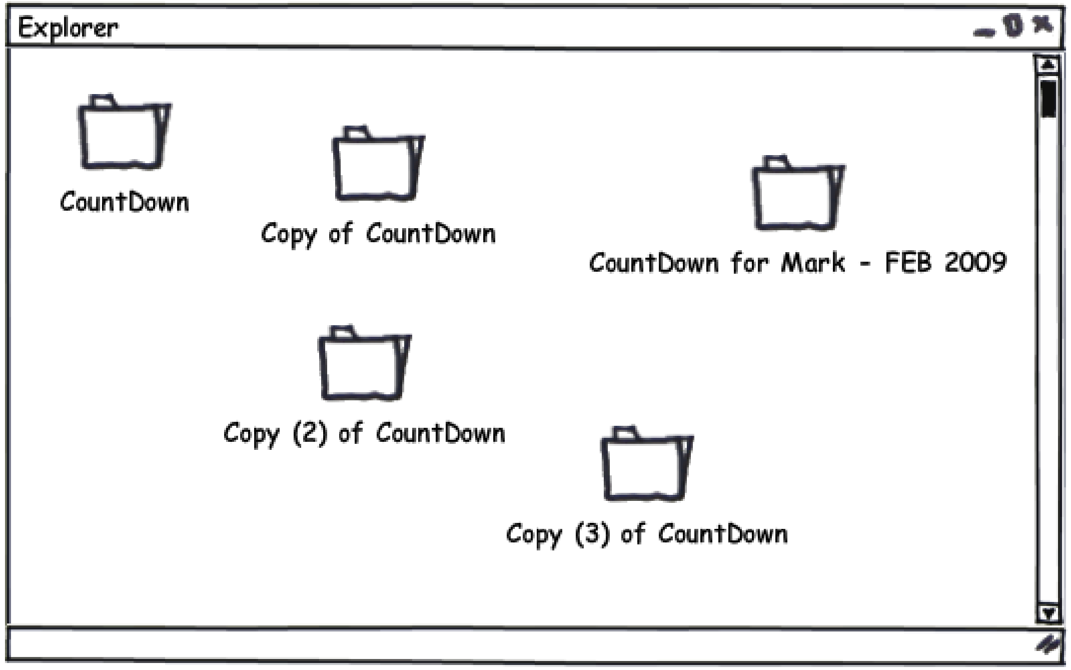
Version Control Systems is a system that records changes to a file or set of files over time so that you can recall specific versions later.

- revert selected 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 did what and when in the system
- Save yourself when things inevitably go wrong
- …

https://www.bmc.com/blogs/continuous-delivery-continuous-deployment-continuous-integration-whats-difference/

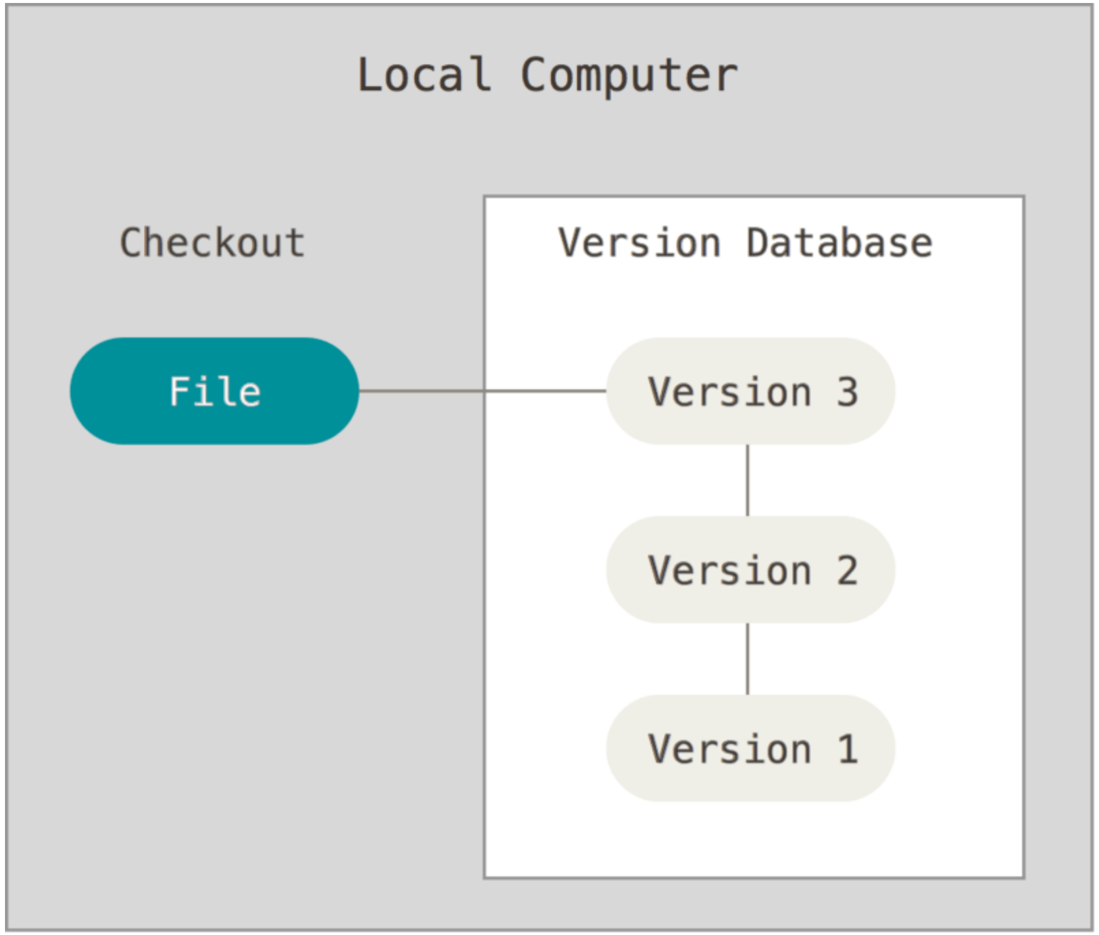
The Problem

- Maintaining group Projects.
- Patches are mostly sent via email
- Difficult to roll back
- Almost impossible to maintain if the number of people working in the project is large
- Testing new unstable features



Local Version Control Systems

- A method for recalling versions of a codebase
- Keeping a record of changes
- Who did what and when in the system
- Save yourself when things inevitably go wrong



Version Control: Why?

Individual

- Back-ups of the project
- Create a “checkpoint” in the project at any stage: Fearlessly modify code
- Tagging: Mark certain point in time
- Branching: Release versions and continue development

Team

- Everything in “Individual”
- Allow multiple developer to work on the same codebase
- Merge changes across same files: handle conflicts
- Check who made which change: blame/praise

Version Control: Types

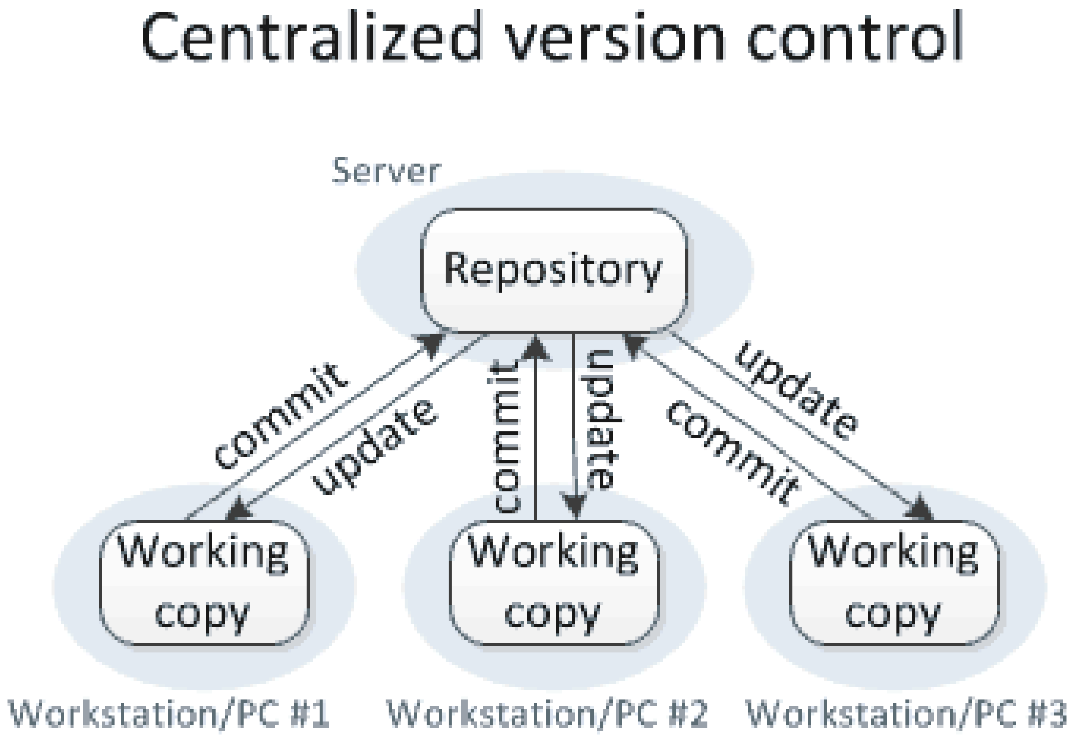
- Centralized Version Control Systems
- Distributed Version Control Systems

Centralised VCS

- A single authoritative data source (repository)
- Check-outs and check-ins are done with reference to this central repository.

Examples:

- Concurrent Version System (CVS)
- Subversion (SVN)



Drawbacks of Centralized Version Control

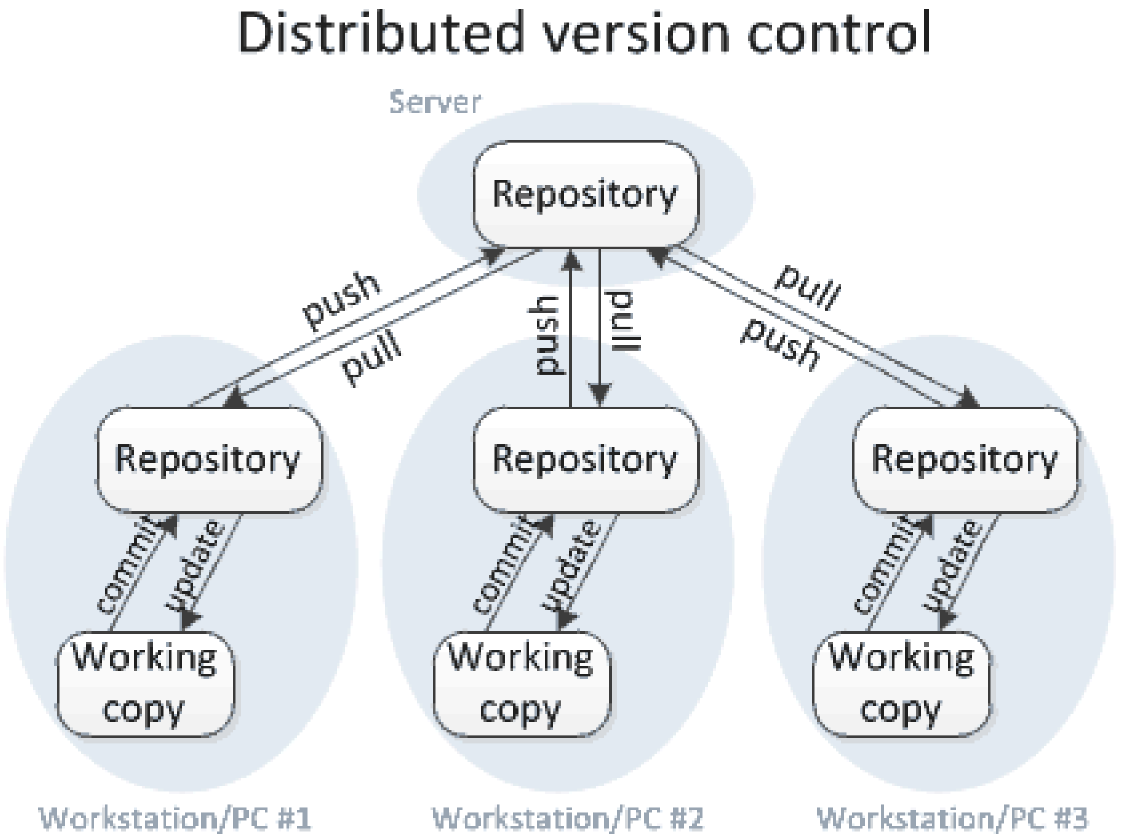
- Single point of failure
- Server downtime = no collaboration
- Risk of data loss
- Local VCS have the same issue

Distributed Version Control Systems

- No single repository is authoritative
- Data can be checked in and out from any repository

Examples

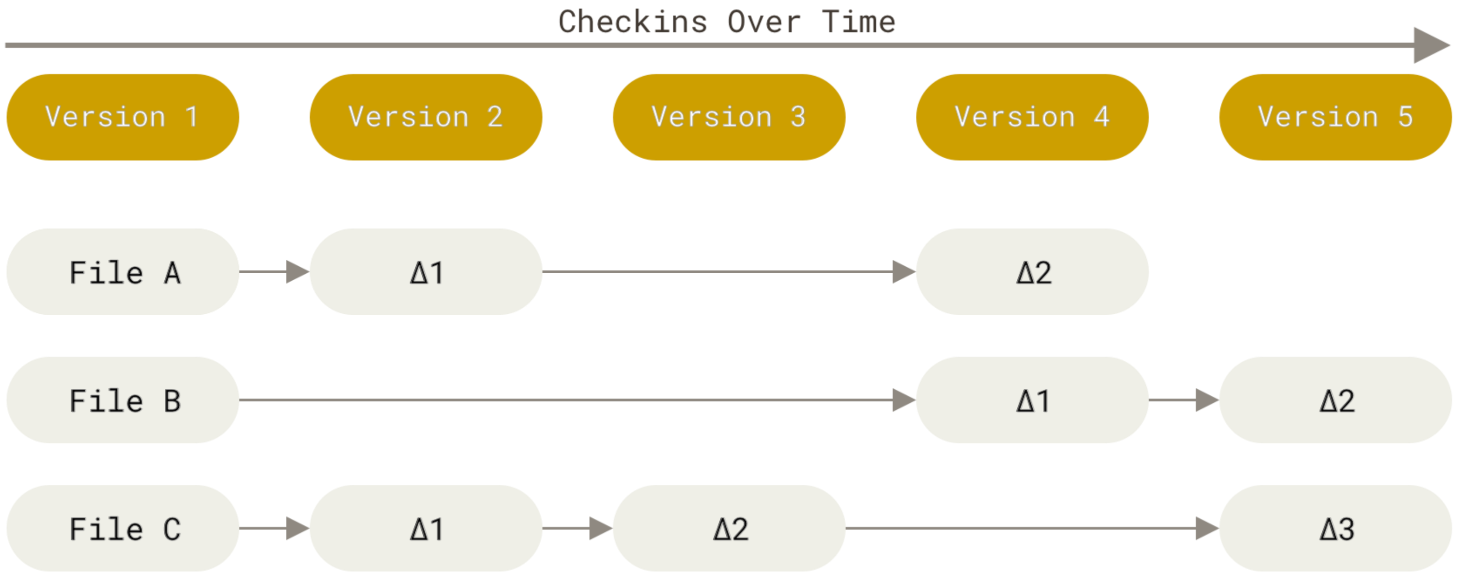
- Git
- Mercurial



What is Git?

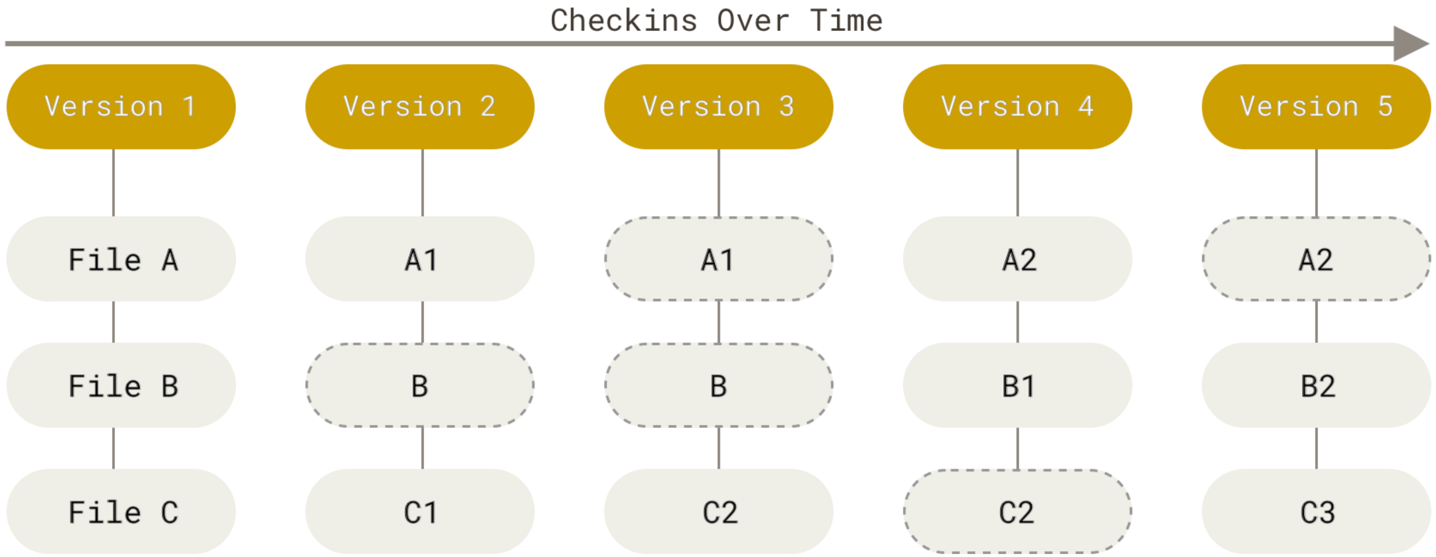
- Git is a popular version control system.
  - Free, open source
  - Fully distributed
  - Handle small files very effectively
  - Tracks contents, not files
- It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.
- It is used for:
  - Tracking code changes
  - Tracking who made changes
  - Coding collaboration

Snapshots, Not Differences



Storing data as changes to a base version of each file

Storing data as snapshots of the project over time



Nearly Every Operation Is Local

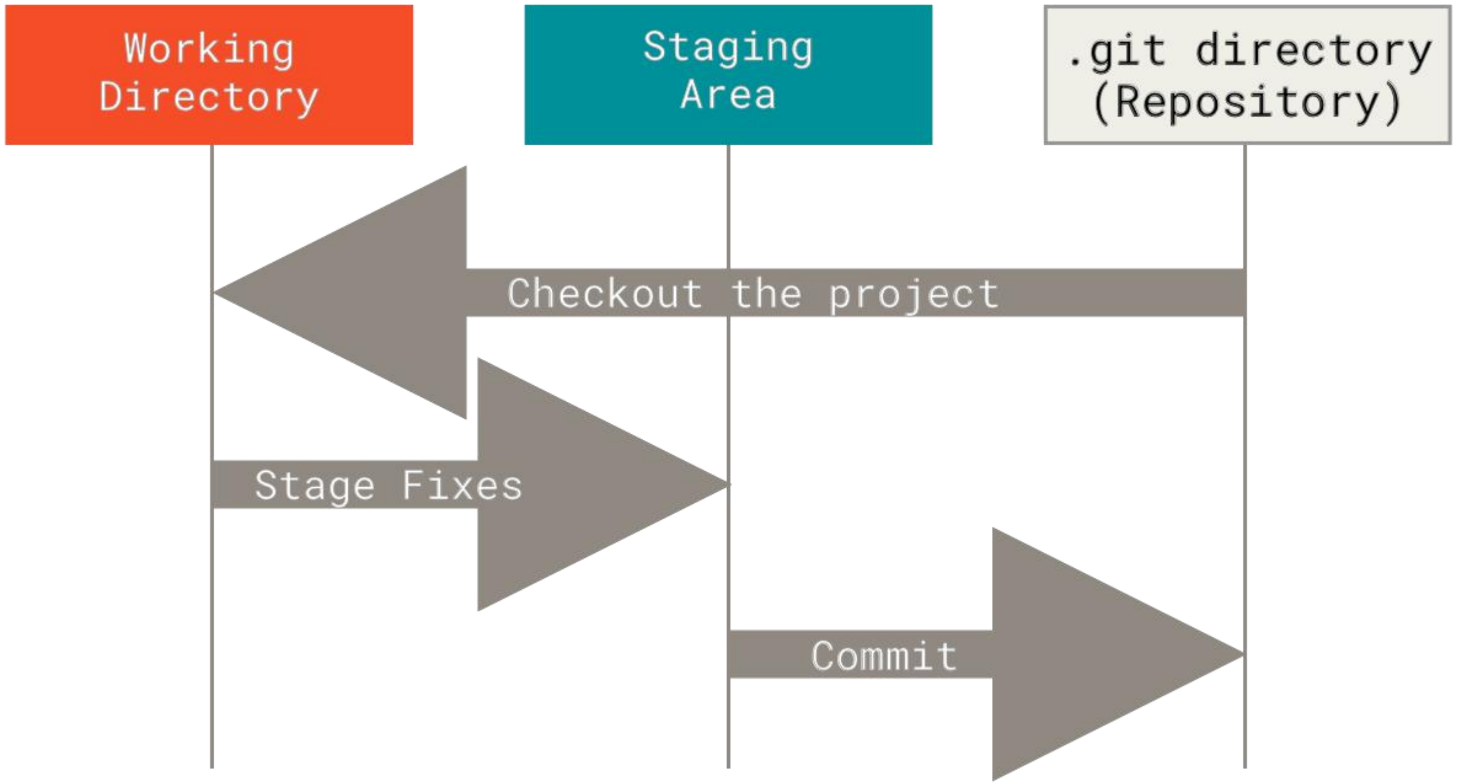
- Local operations = fast
- Full project history on your machine
- Instant history browsing
- Quick file comparisons
- Offline work
- No server dependency

Why Git?

- Over 70% of developers use Git!
- Developers can work together from anywhere in the world.
- Developers can see the full history of the project.
- Developers can revert to earlier versions of a project.
- Git Has Integrity
- Git Generally Only Adds Data

Git: Stages

- Working directory
- Staging directory
- Git directory (repository)



Basic Git workflow

1) You modify files in your working tree.
2) You selectively stage just those changes you want to be part of your next commit, which adds *only* those changes to the staging area.
3) You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

Setting up Git

```
$ git --version

$ git config --global user.name “Sherlock Holmes”

$ git config --global user.email “Imsherlocked@gmail.com”

$ git config -h (list of commands)

$ git config --help (git manual)
```

Your default branch name

```
$ git config --global init.defaultBranch main
```

Checking Your Settings

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

Getting a Git Repository

1) You can take a local directory that is currently not under version control, and turn it into a Git repository, or

2) You can *clone* an existing Git repository from elsewhere.

Git: Development

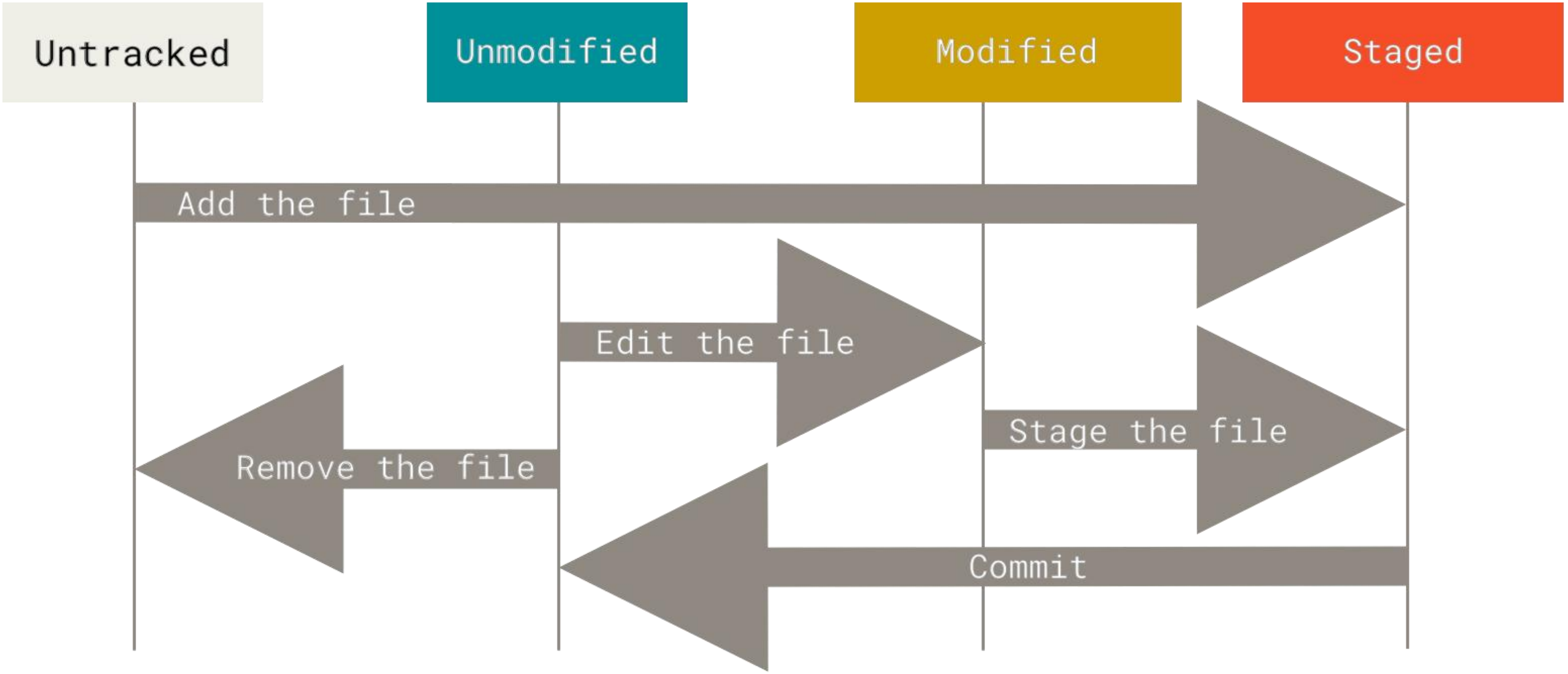
Initializing a Repository in an Existing Directory

```
$ cd <path>
$ git init
$ git add *.c
$ git add LICENSE
$ git commit -m 'Initial project version'
```

Cloning an Existing Repository

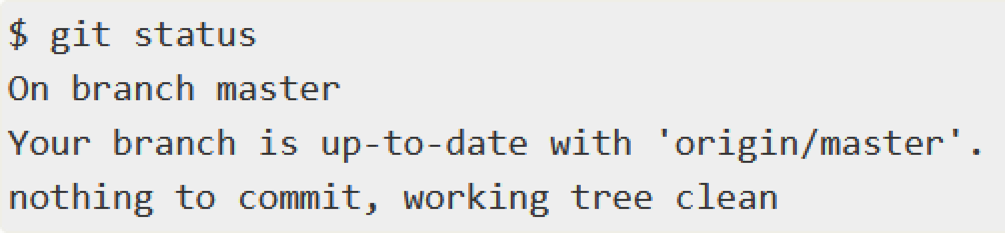
```
$ git clone <remote-url>
```

The lifecycle of the status of your files



Git: Development

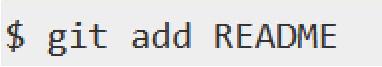
Checking the Status of Your Files



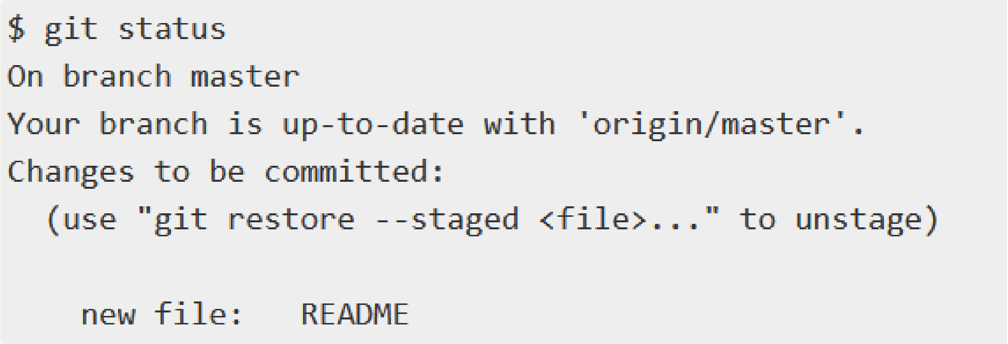
```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
```

Tracking New Files

$ git add <files>



```
$ git add README
```



```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)

    new file:   README
```

Git: Development

View changes

- $ git diff
- $ git diff --staged

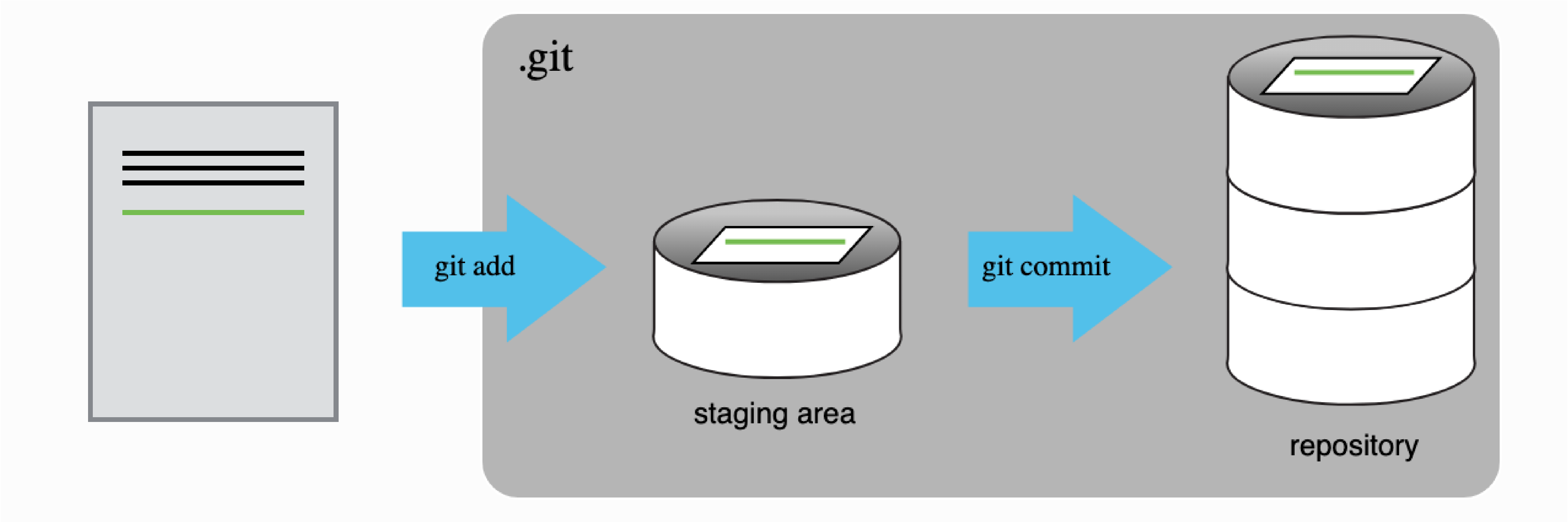
Git: Development

Create “snapshots” of your codebase

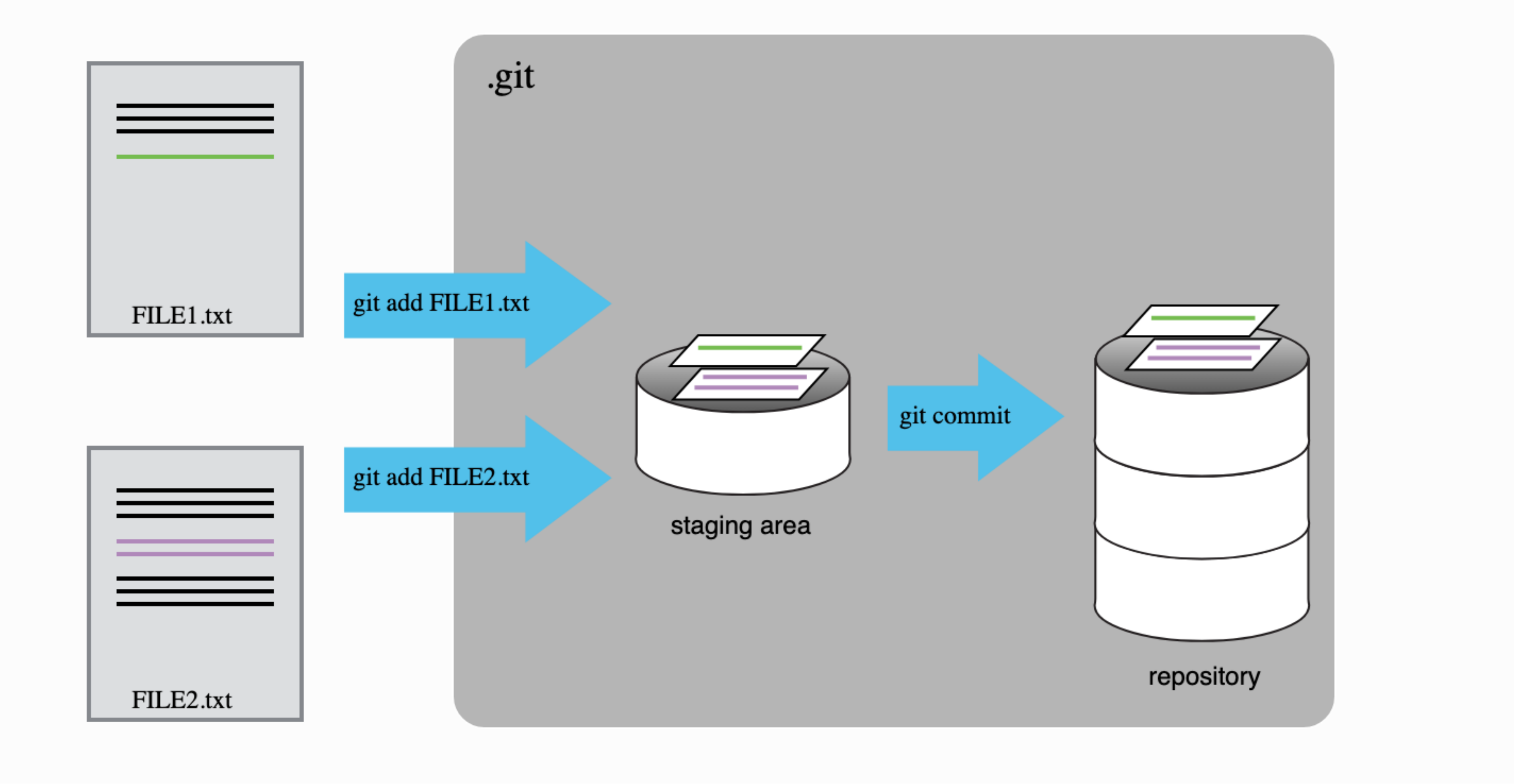
- git commit

Records changes to the repository

How does Git work?



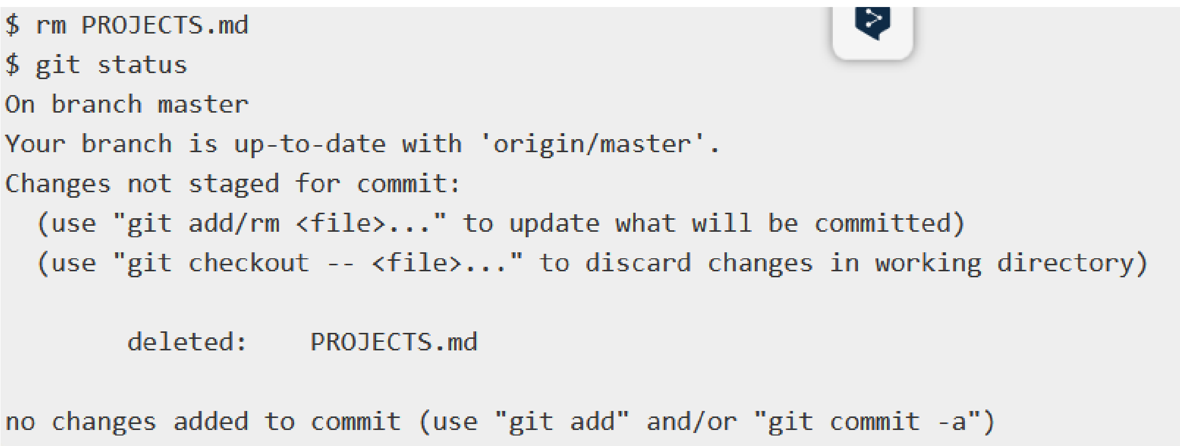
How does Git work?



Git: Development
Removing Files

Removing Files

$ rm <file>

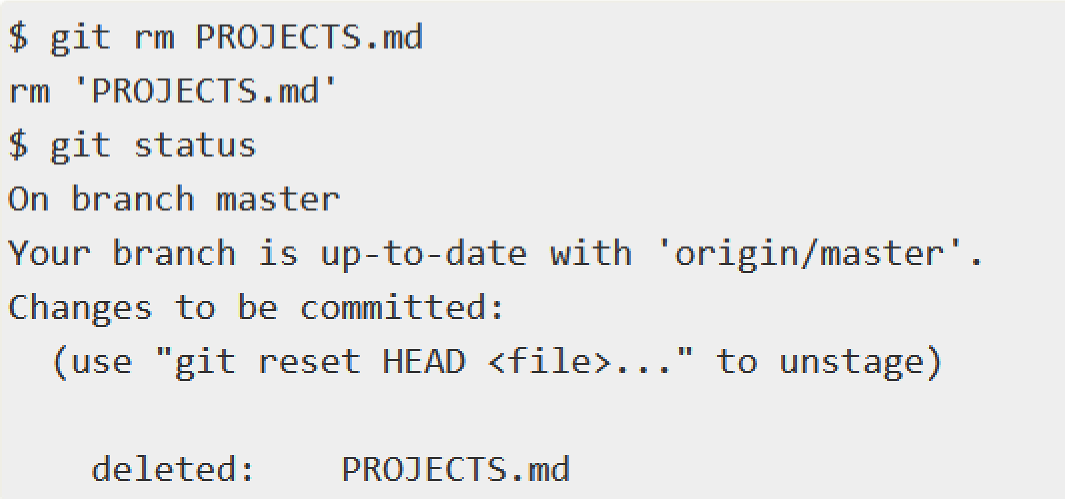


```
$ rm PROJECTS.md
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    PROJECTS.md

no changes added to commit (use "git add" and/or "git commit -a")
```

$ git rm PROJECTS.md



```
$ git rm PROJECTS.md
rm 'PROJECTS.md'
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:    PROJECTS.md
```

$ git rm –cached <file>
$ git rm log/*.log
$ git rm *~

Git: Development

Moving Files

```
$ git mv file_from file_to
```

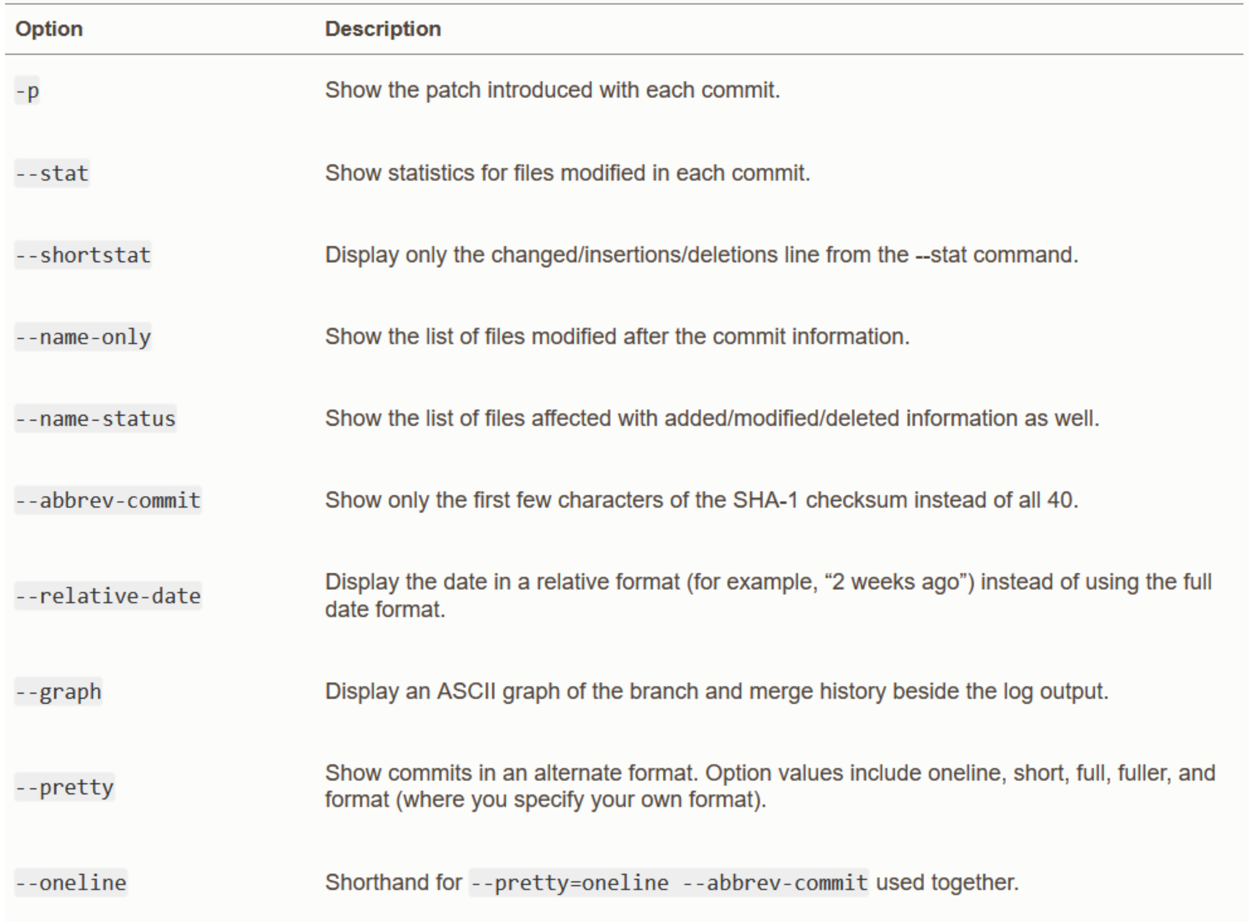
Viewing the Commit History

Check “snapshots” of the codebase

- git log

  Show commit logs

Common options to git log



Option	Description
`-p`	Show the patch introduced with each commit.
`--stat`	Show statistics for files modified in each commit.
`--shortstat`	Display only the changed/insertions/deletions line from the --stat command.
`--name-only`	Show the list of files modified after the commit information.
`--name-status`	Show the list of files affected with added/modified/deleted information as well.
`--abbrev-commit`	Show only the first few characters of the SHA-1 checksum instead of all 40.
`--relative-date`	Display the date in a relative format (for example, “2 weeks ago”) instead of using the full date format.
`--graph`	Display an ASCII graph of the branch and merge history beside the log output.
`--pretty`	Show commits in an alternate format. Option values include oneline, short, full, fuller, and format (where you specify your own format).
`--oneline`	Shorthand for `--pretty=oneline --abbrev-commit` used together.

Git Branching

Snapshots, not diffs — Git saves the full state of files at each commit, not just changes.

Commit object — stores:

- pointer to the snapshot
- author info (name & email)
- commit message
- links to parent commit(s)

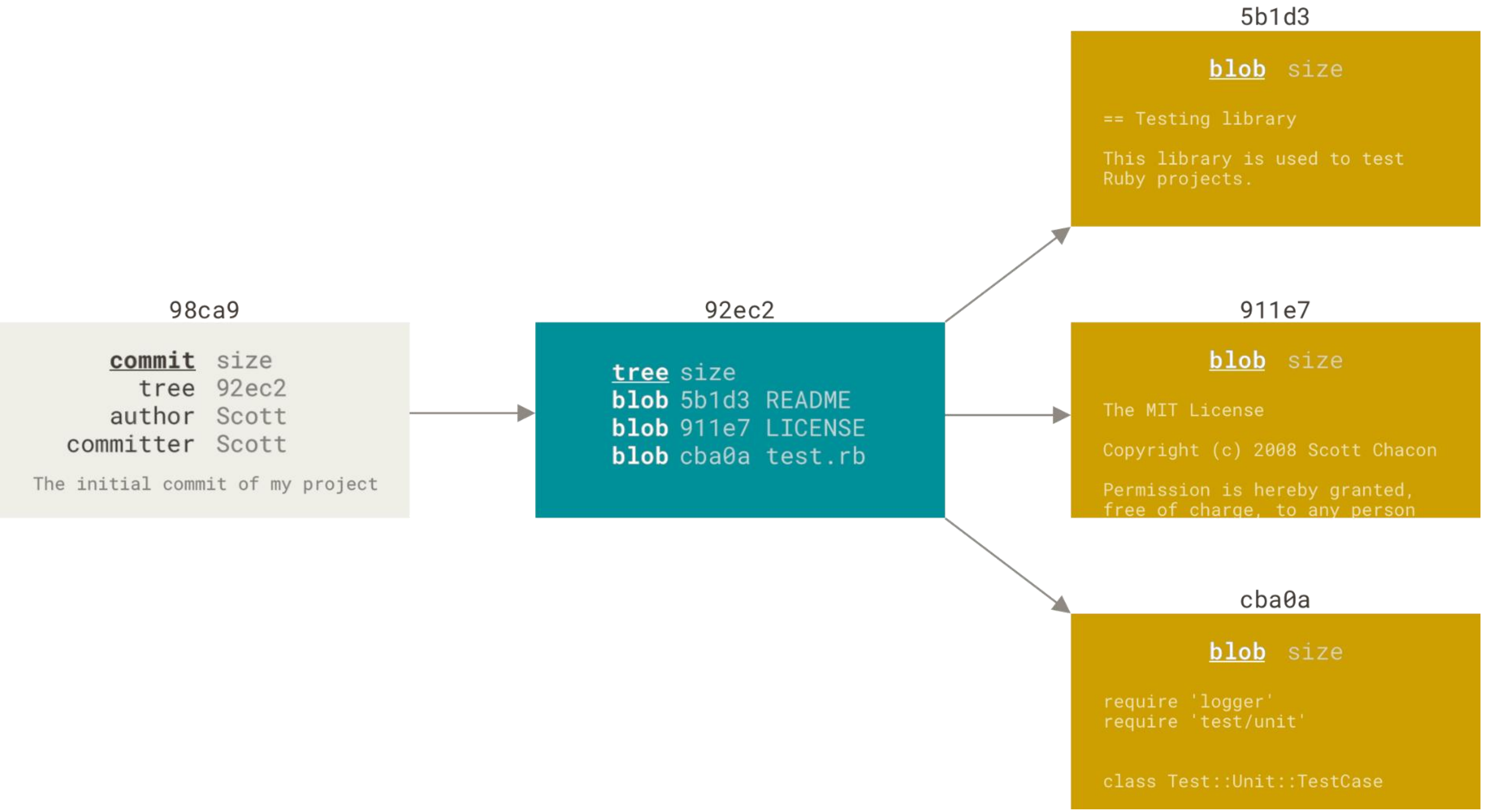
Staging process — files are checksummed (SHA-1) and saved as blobs in the repository.

Tree object — organizes files and directories for the commit.

Repository structure example — after one commit you have:

- 3 blobs (file contents)
- 1 tree (directory structure)
- 1 commit (metadata + pointer to the tree)

A commit and its tree



98ca9

```
   commit  size
     tree  92ec2
   author  Scott
committer  Scott

The initial commit of my project
```

92ec2

```
tree size
blob 5b1d3 README
blob 911e7 LICENSE
blob cba0a test.rb
```

5b1d3

```
blob  size

== Testing library

This library is used to test
Ruby projects.
```

911e7

```
blob  size

The MIT License

Copyright (c) 2008 Scott Chacon

Permission is hereby granted,
free of charge, to any person
```

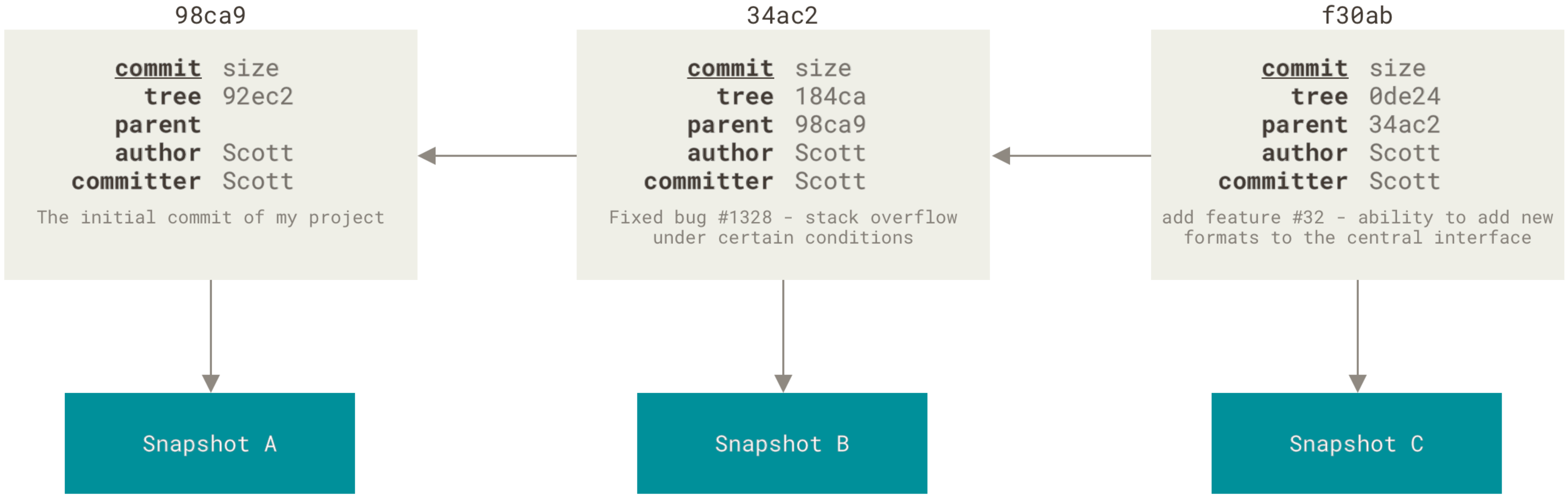
cba0a

```
blob  size

require 'logger'
require 'test/unit'


class Test::Unit::TestCase
```

Commits and their parents



98ca9

```
   commit  size
     tree  92ec2
   parent
   author  Scott
committer  Scott
```

The initial commit of my project

34ac2

```
   commit  size
     tree  184ca
   parent  98ca9
   author  Scott
committer  Scott
```

Fixed bug #1328 - stack overflow under certain conditions

f30ab

```
   commit  size
     tree  0de24
   parent  34ac2
   author  Scott
committer  Scott
```

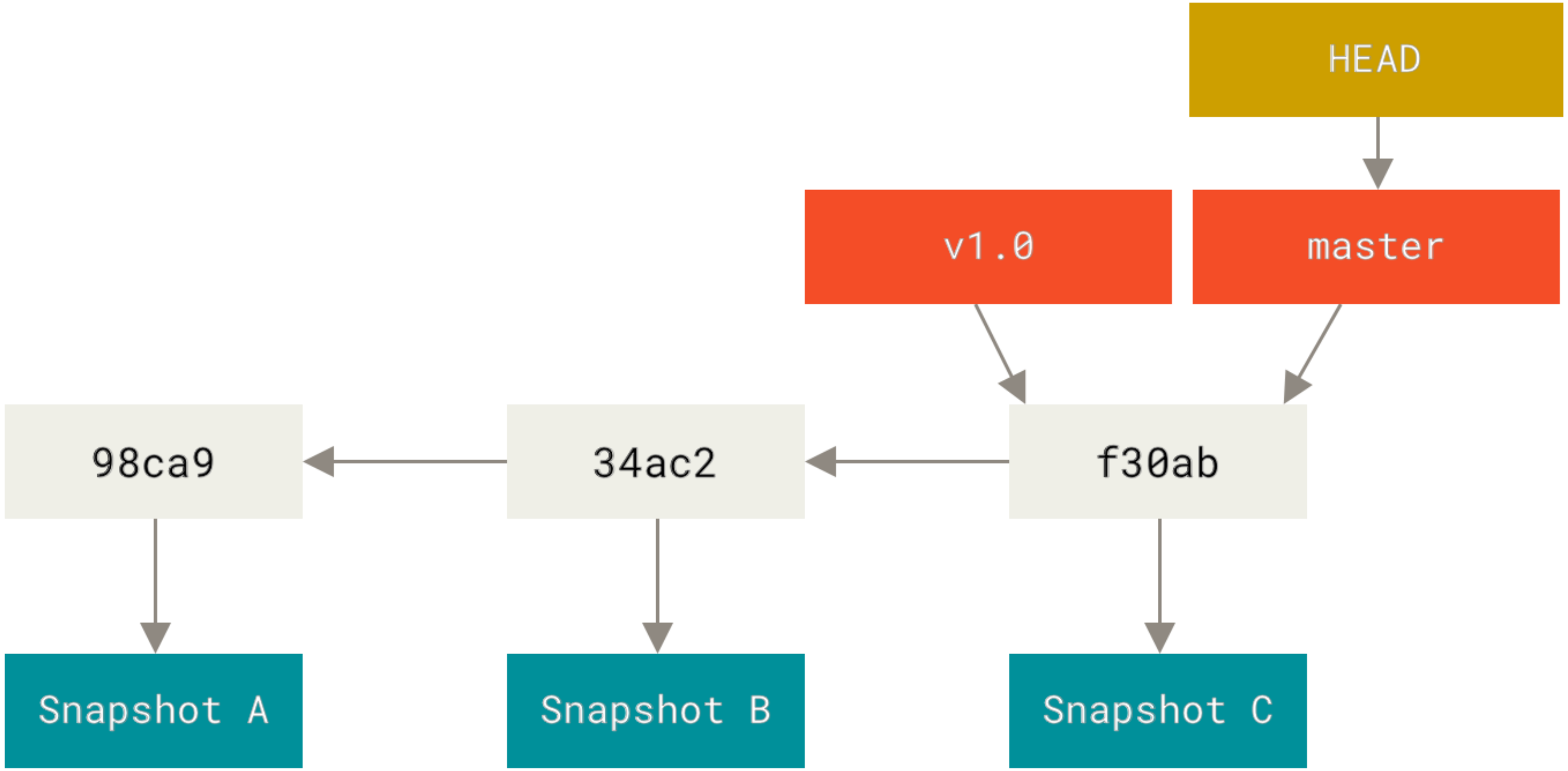
add feature #32 - ability to add new formats to the central interface

Snapshot A

Snapshot B

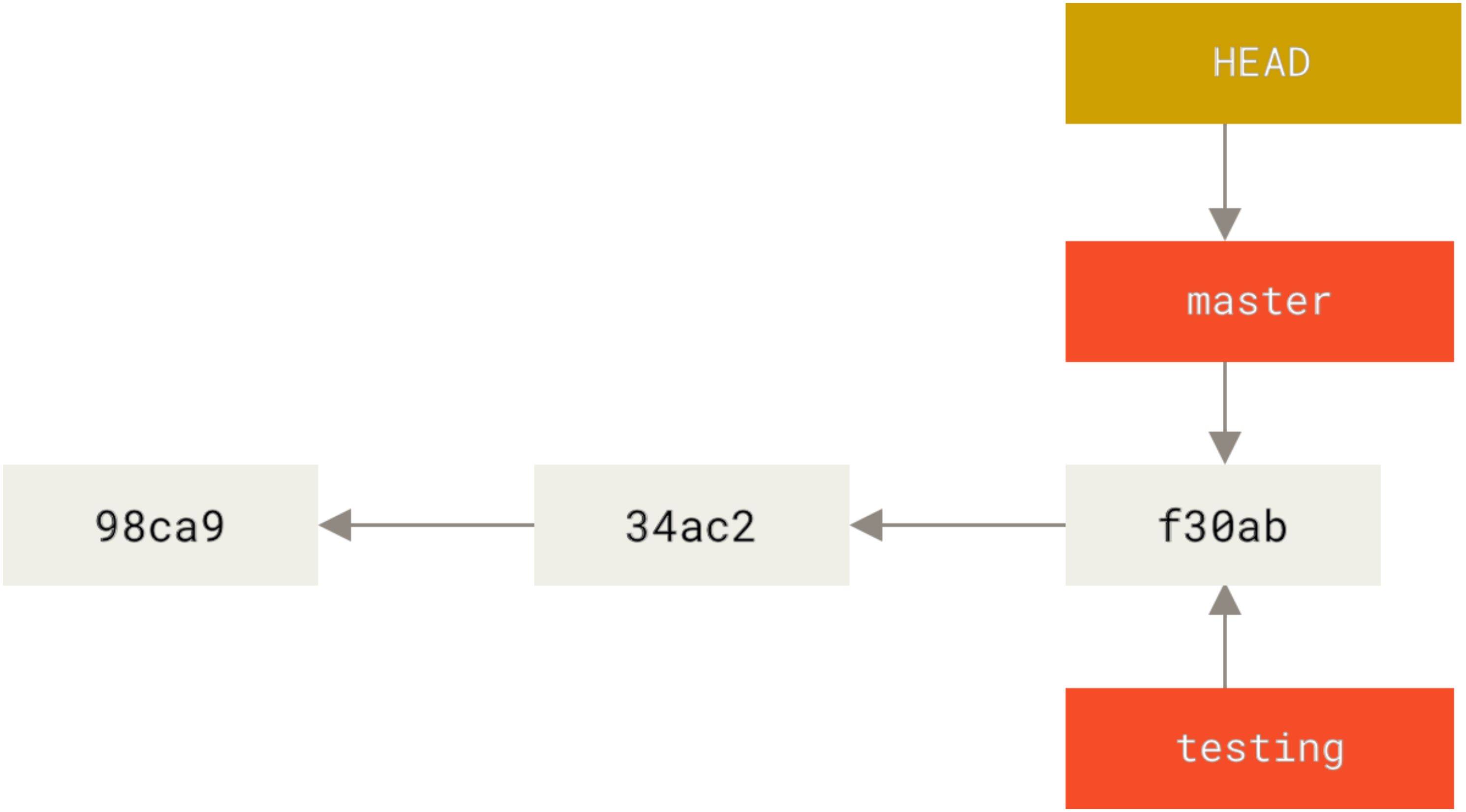
Snapshot C

A branch and its commit history



Creating a New Branch

```
$ git branch testing
```

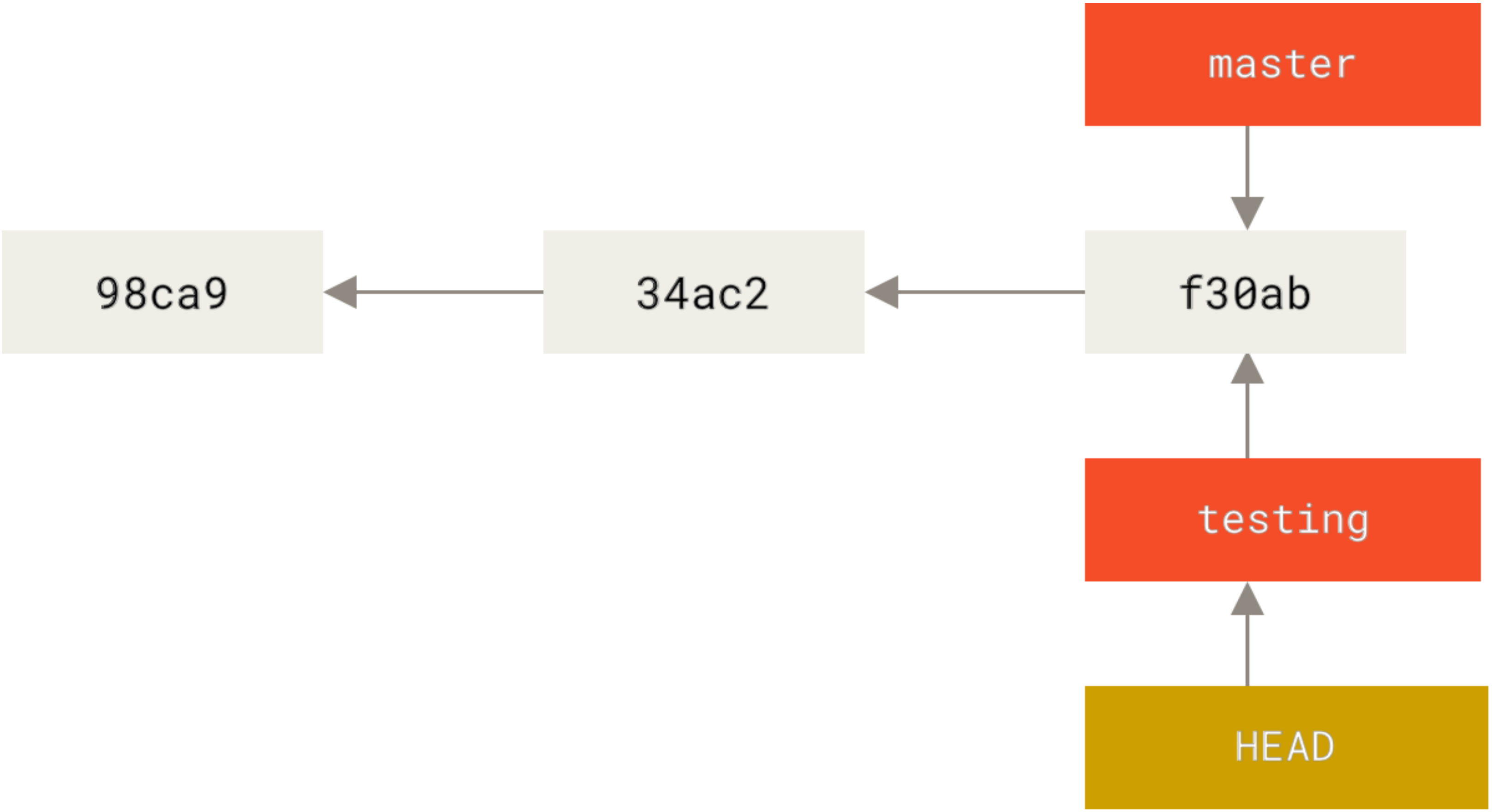


Switching Branches

$ git checkout

Example:

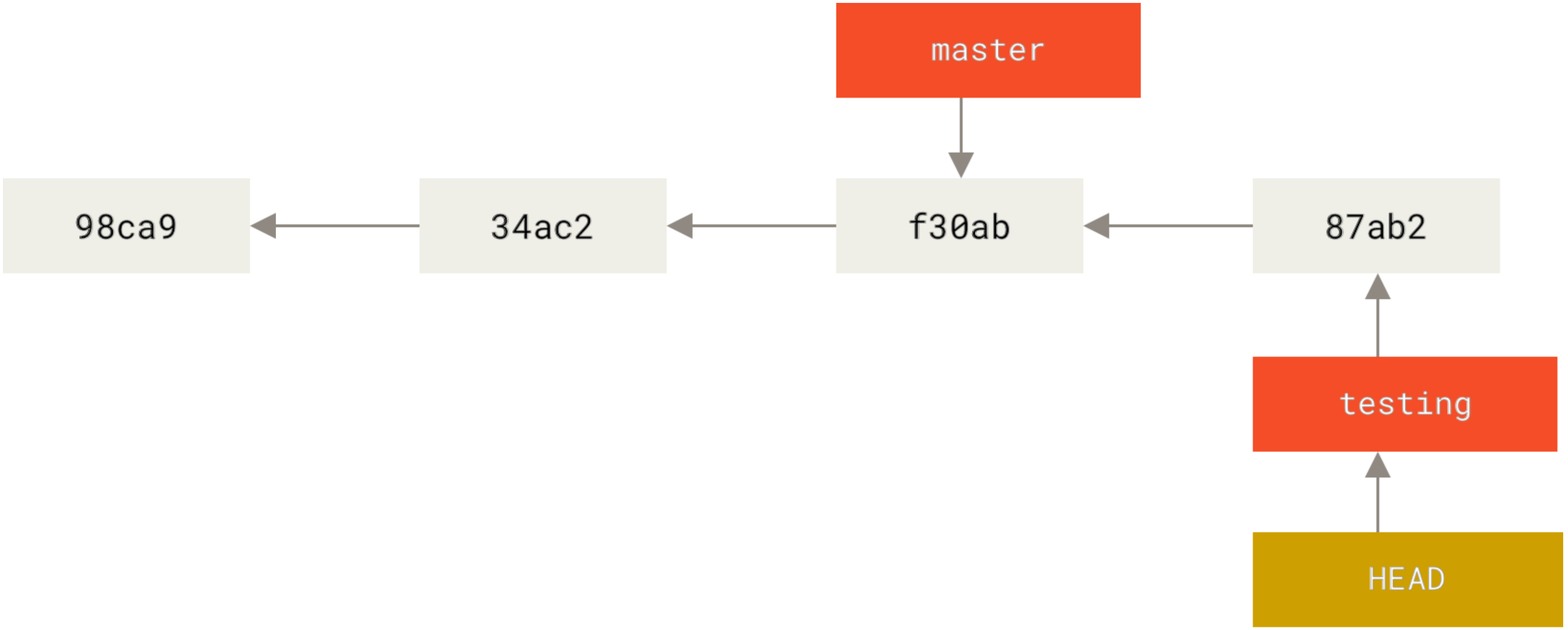
$ git checkout testing



The HEAD branch moves forward when a commit is made

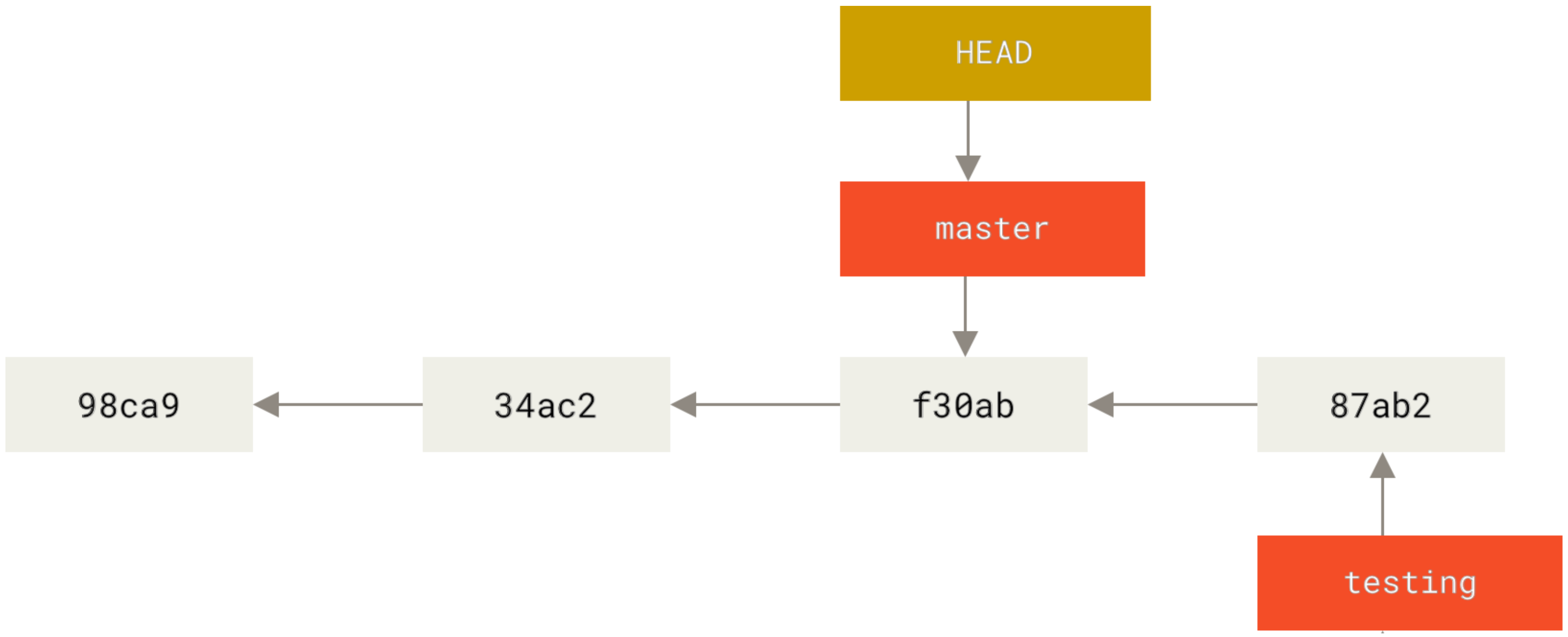
$ vim test.rb

$ git commit -a -m 'made a change'



HEAD moves when you checkout

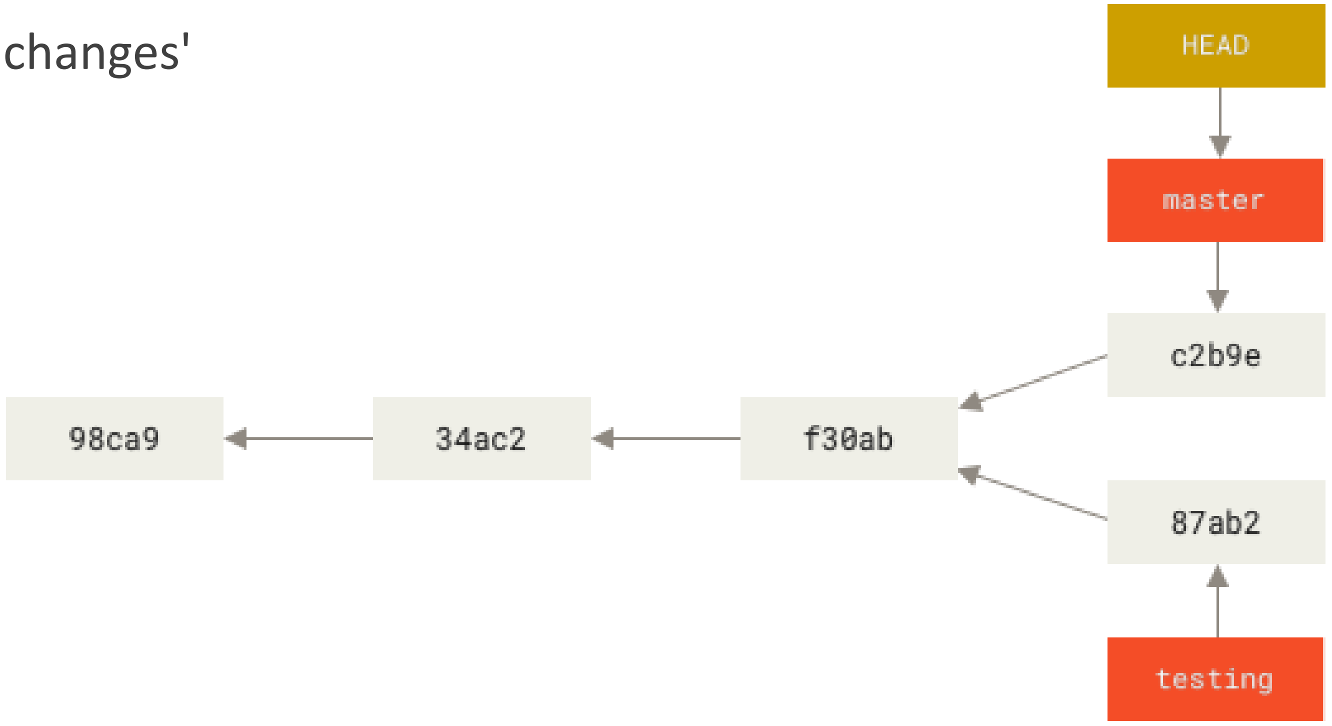
$ git checkout master



Divergent history

```
$ vim test.rb
```

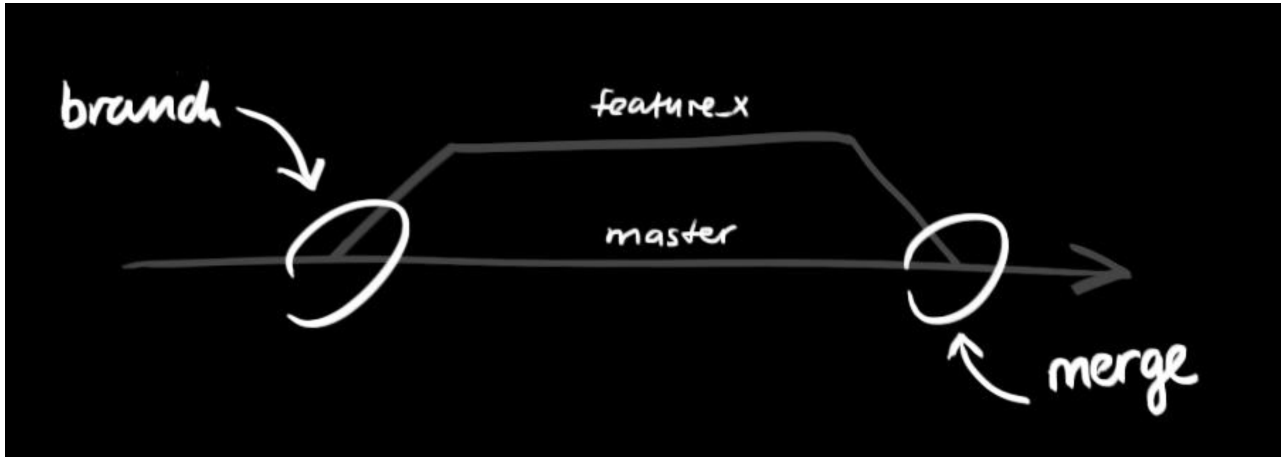
```
$ git commit -a -m 'made other changes'
```



Git: Development

Branches

- git checkout –b <branch-name>



Git: Development

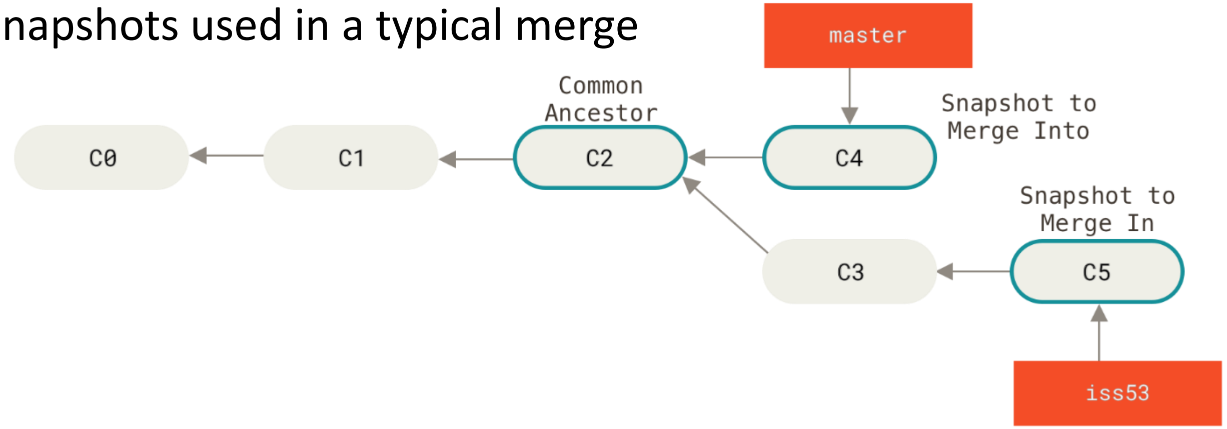
Merge other branches

- git merge

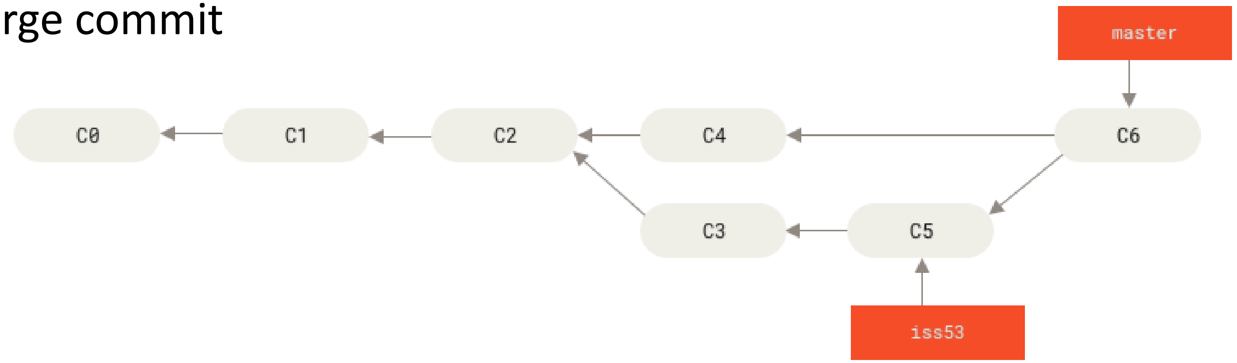
Example:

```
$ git checkout master
Switched to branch 'master'
$ git merge iss53
$ git branch -d iss53
```

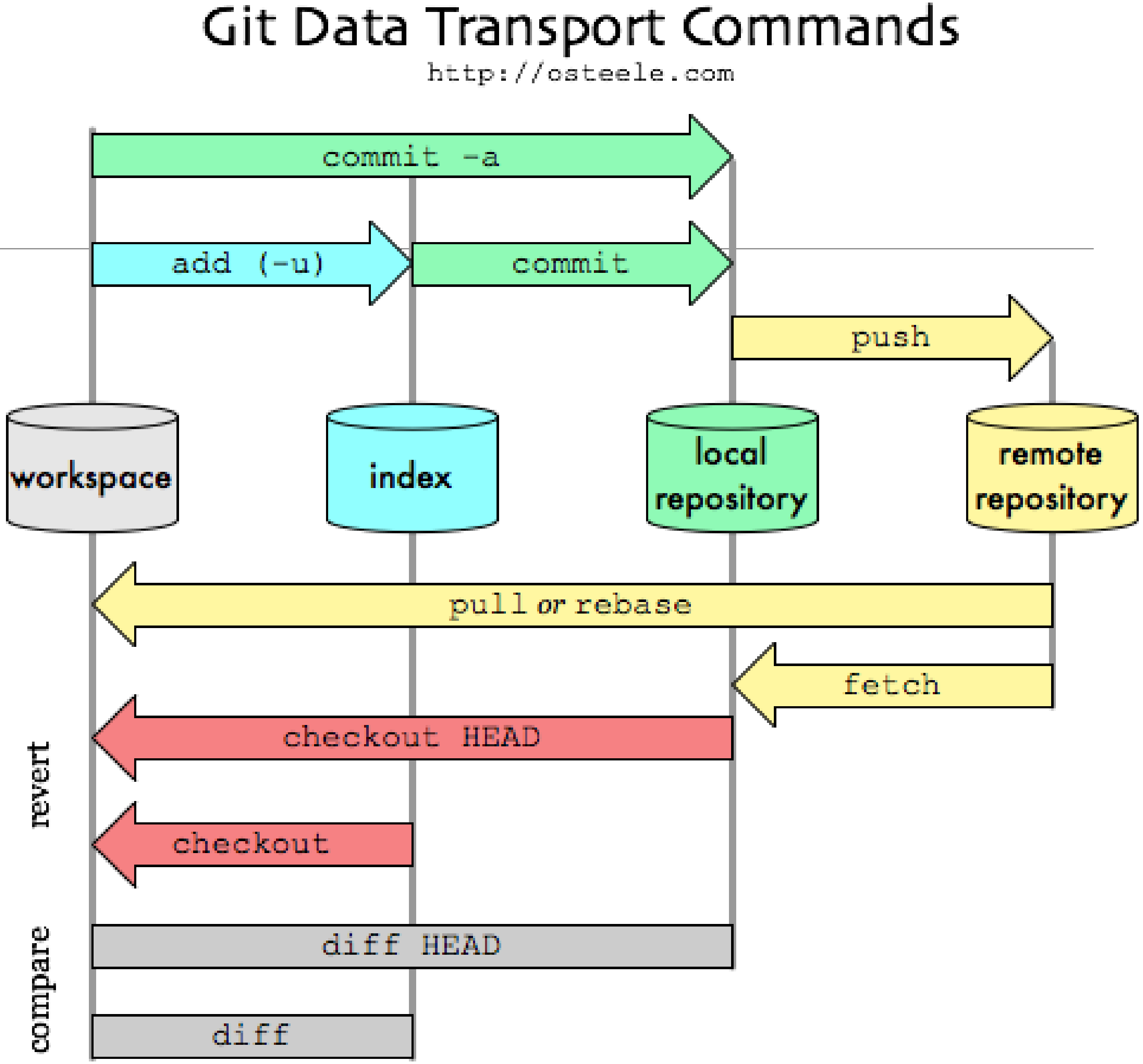
Three snapshots used in a typical merge



A merge commit

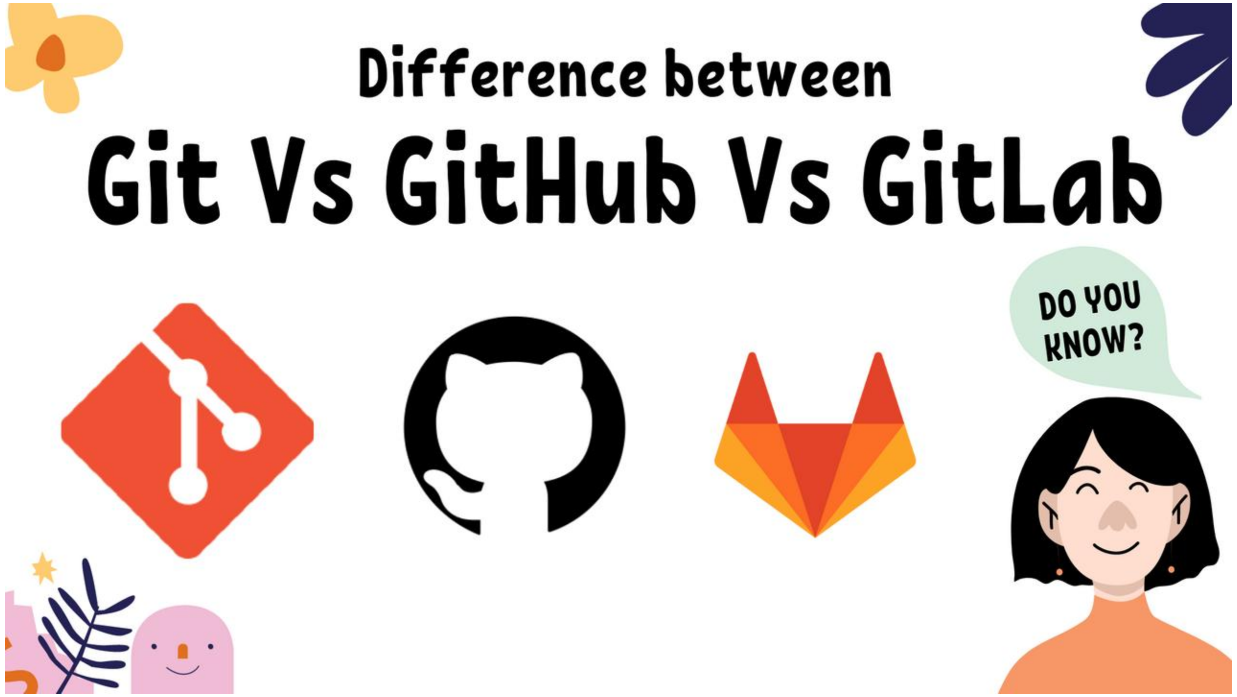


Git: Development



What is Git/GitLab/GitHub?

- Git: Version Control System.
- GitLab/GitHub: Source Code management System/platform used to host git repositories and to share it with others enabling collaboration.
- GitLab has a built-in CI/CD whereas GitHub doesn't.



Creating a Repository

$ mkdir gitlab (make a directory - repository)

$ cd gitlab

$ git init (to initialise git)

$ vim readme.md (Press ‘i’ to go to insert mode, press ’escape’ to come out of insert mode. Press ‘:x’ to save and exit)

$ ls (to list items)

$ ls -a (to show hidden directory)

$ git status (to show on which branch are you)

$ git add . (sends file to staging area)

$ git commit -m “my initial commit” (sends file to local repo)

Git Unstage files

```
vim index.html
vim readme.md (”Hello! I am learning git”)
git status
git add .
git commit -m “Added index.html and readme”
git status
git reset HEAD readme.md (Note: Removes from the staging area and it doesn’t mean that the changes have gone away. Need to commit)
git status
git commit -m “added first page” (Commits only index.html)
git status
git add .
git commit -m “added readme”
git status
```

Track Changes

$ git status

$ git log (shows the time and date of each commit)

$ git log --patch (shows the details of the file)

$ git diff (helps to review changes)

$ git diff --staged (Press q to quit)

Committing a folder

```
mkdir temp

ls -a

git status

touch temp/.gitkeep (Create an empty file inside an empty folder)

git status

git add .

git commit -m “Added a temp folder”

git status
```

Delete Files

```
touch newfile.txt

ls -a

git status

rm newfile.txt

git status
```

Question: Will it be removed?

Git Branch

```
git checkout -b feature/new-table

git status

vim index.html (Make some changes in the branch feature/new-table)

git add .

git commit -m “Added a table”

cat index.html

git checkout master

cat index.html

git checkout feature/new-table

git branch -d feature/new-table
```

Git Merge

Fast-forward merge

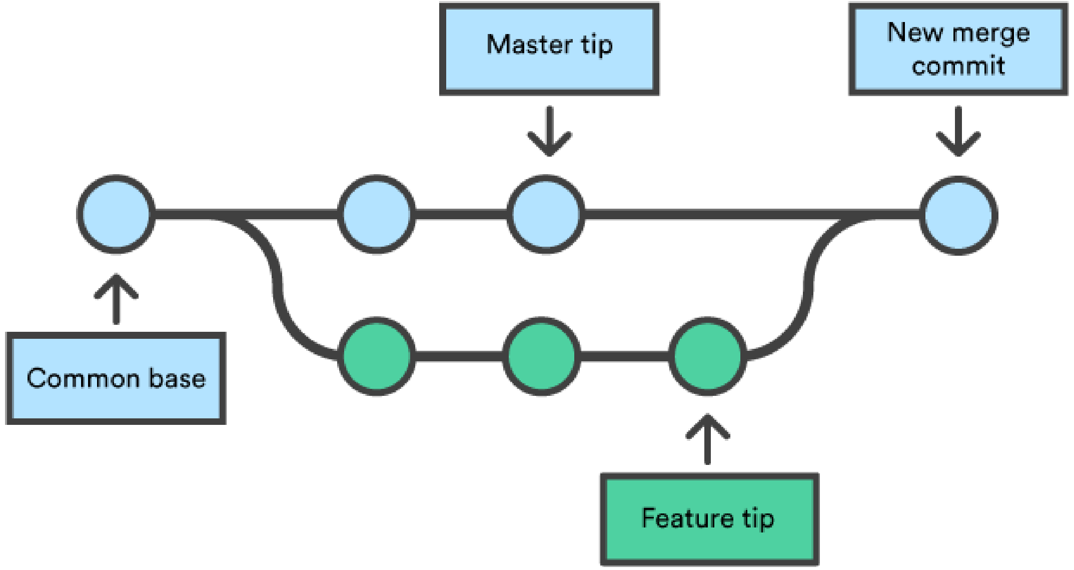
git checkout master

git merge feature/new-table

git log

git branch

git branch -d feature/new-table



Resources

https://git-scm.com/book/ms/v2/Getting-Started-About-Version-Control

https://www.w3schools.com/git/git_intro.asp?remote=github

https://docs.github.com/en/get-started/using-git/about-git