Survival guide:

References

https://git-scm.com/

https://www.atlassian.com/

Why Git was created

The context: Linus Torvalds and the kernel

https://lkml.org/lkml/2005/4/6/121

2005-04-05: BitMover (Inc.) announces it will stop providing an open source version of their software BitKeeper (distributed version control system)

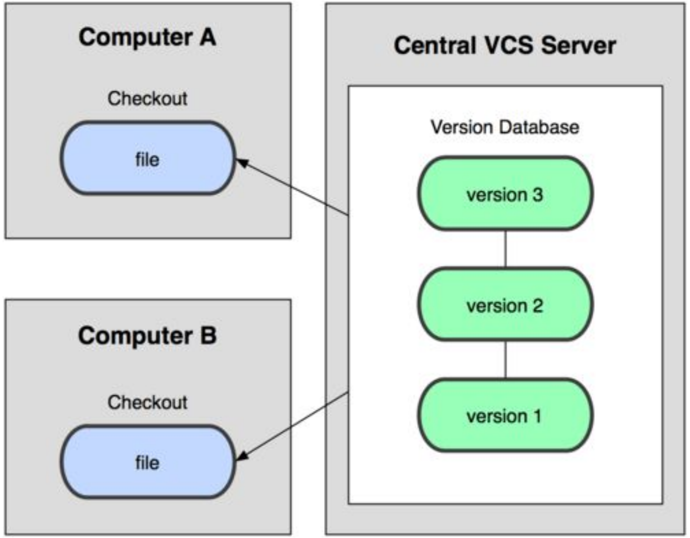
2005-04-06 : Linus decides to stop using BitKeeper (after 3 years of usage) for Linux development and starts looking for an alternative

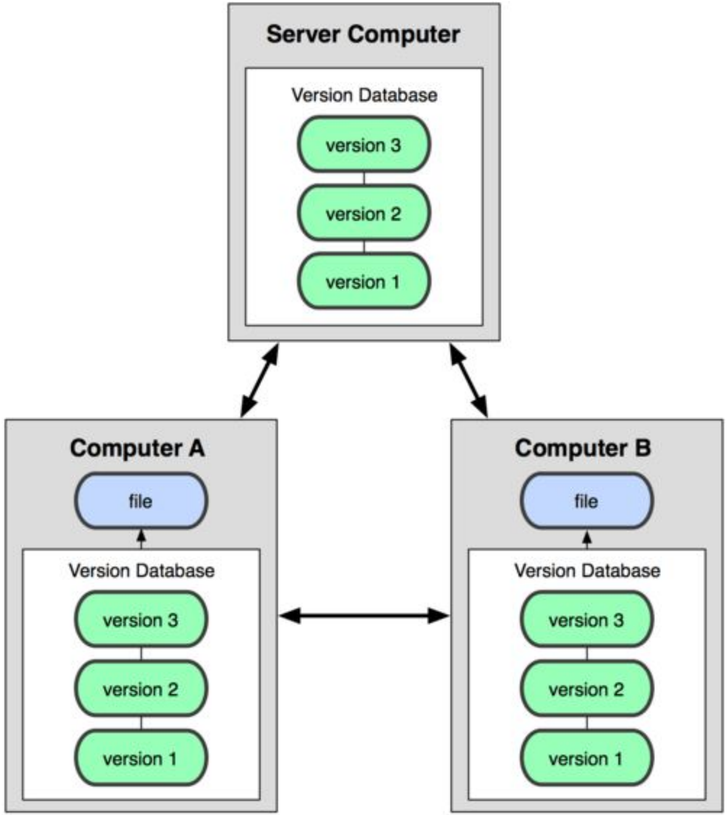
The context: Linus Torvalds and the kernel

An old Tech Talk at Google: https://www.youtube.com/watch?v=4XpnKHJAok8

- BitKeeper was not an option anymore because of licensing issues (caused by some kernel contributor trying to reverse engineer the BitKeeper client)
- They were no other suitable alternatives at the time that would fit the kernel development process
- Linus hates CVS and SVN
- Linus decided to make his own version control system… in two weeks

Centralised or distributed?





The context: Linus Torvalds and the kernel

“Yes, I still hate CVS with a passion, almost two decades after I had to use that horrid horrid thing. Some mental scars will not go away.”

-LinusTorvalds,24 May 2020

https://lkml.org/lkml/2020/5/24/384

The context: Linus Torvalds and the kernel

Git was designed with these objectives in mind:

- Reliability
- Consistency (resistant to file corruption) and security
- High performance
- Distributed
- Easy merge

Working locally

Repository

The repository is a (key-value) database. It stores Git objects, indexed by their SHA1 hash.

Creating the database is done through one of these commands:

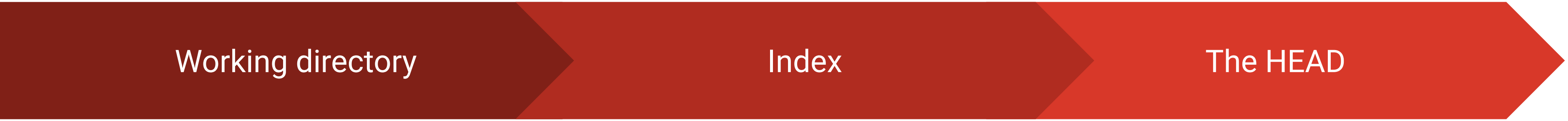
```
git init/git clone
```

Will create a new repository

Will copy a repository, given a URL

The 3-trees architecture

https://git-scm.com/book/en/v2/Git-Tools-Reset-Demystified



Working directory

Index

The HEAD

The OS directory containing the .git folder

Also called “staging area”.

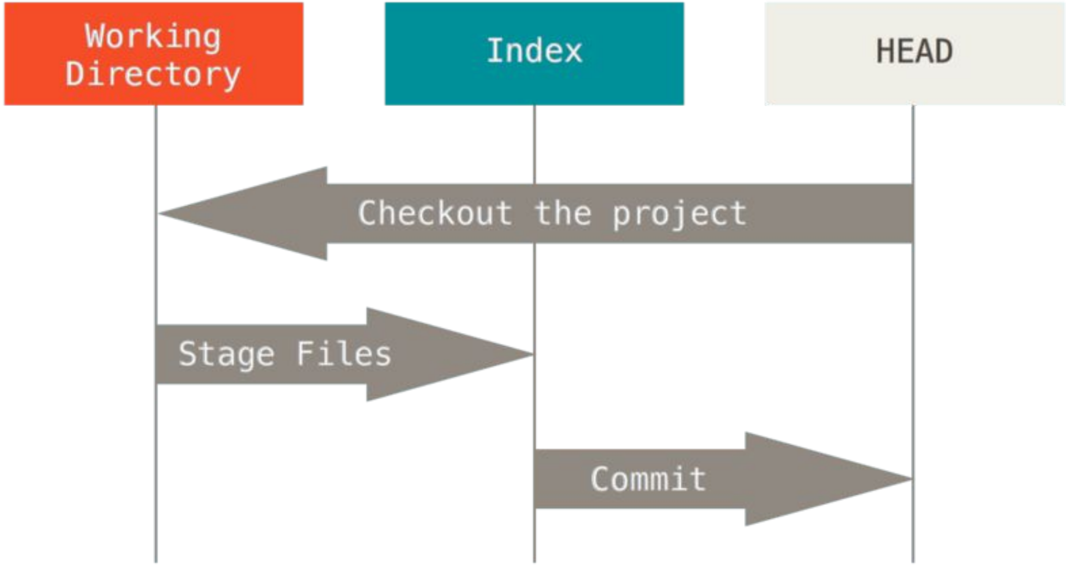
It contains a “commit in preparation”.

It is used as an intermediary level before committing file.

The HEAD is a pointer to the current branch reference.

The HEAD is a snapshot of your latest commit.

Typical workflow



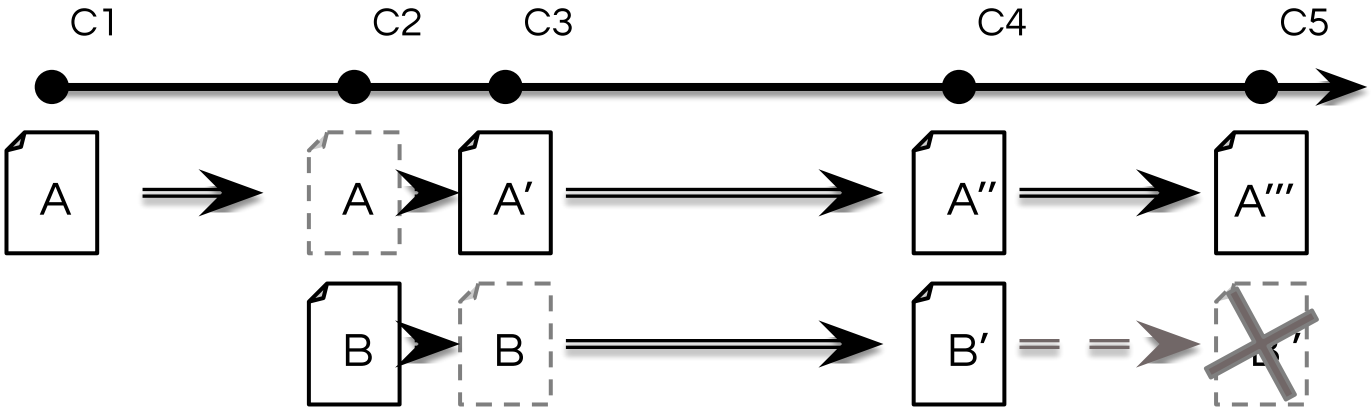
Working Directory

This is the directory containing the .git folder, which contains all git-related information, including the database.

Modifications in the working directory are not automatically saved in the repository.

Commit

A commit represents a state of the **working directory** at a given time.

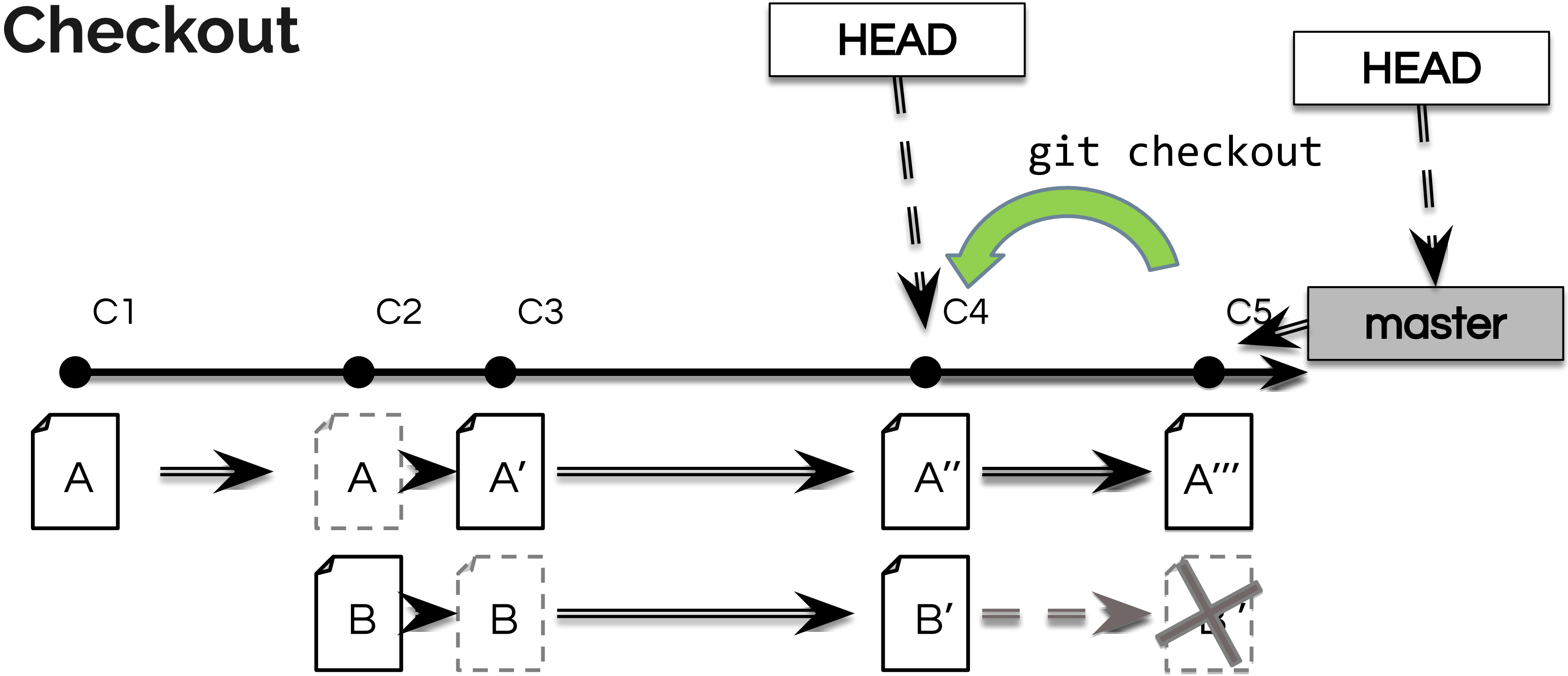


Committing in practice

1. Create/Modify a file within the working directory.
2. Add it to the index (staging area)
3. Commit the staging area

Note: Use “git status” before and after each step to witness the

Checkout



Inspecting history

`git log`

`--oneline`	: Display in one line the commit hash + message summary
`--path -p`	: Display the diff
`-n <number>`	: limit history to the last *n* commit
`--stat`	: Display changes/deletions/insertions in each file
`--graph`	: Display commit history as an ASCII graph

Inspecting the content of a commit

```
git show <object=HEAD>
```

Inspecting changes in a range

Between two commits

```
git diff <commit1> <commit2>
```

Between two commits on a file

```
git diff <commit1> <commit2> -- <path>
```

Interactive staging

In case you have lot of unstaged modifications, and want to separate them between multiple commits.

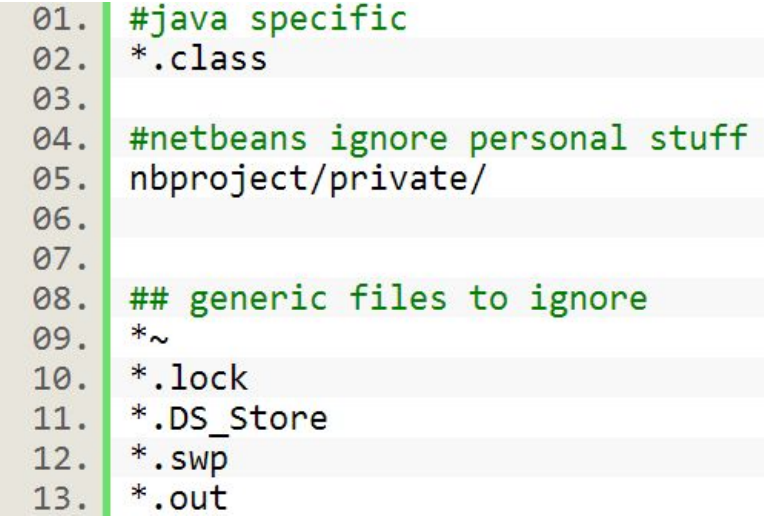
```
$ git add -i
```

.gitignore

To automatically exclude files from Git.

Typical candidates:

- Credentials
- Compiled binaries
- IDE configuration files
- Temporary files



```
#java specific
*.class

#netbeans ignore personal stuff
nbproject/private/


## generic files to ignore
*~
*.lock
*.DS_Store
*.swp
*.out
```

What to do if you mess up

Change the last commit

```
$ git commit --amend
```

git reflog

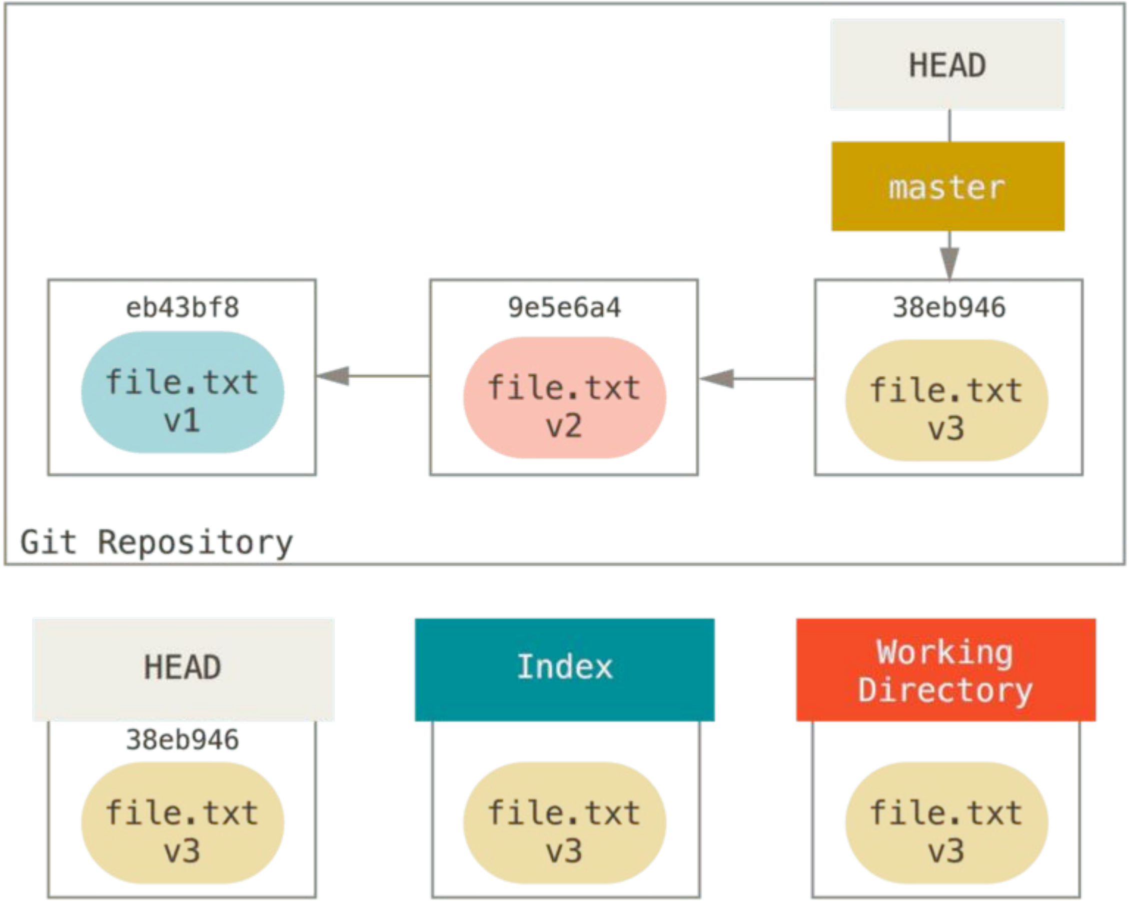
Literally, the “references log”.

This history keeps track of all the actions performed on references (branches/HEAD), along with the SHA1 indexes.

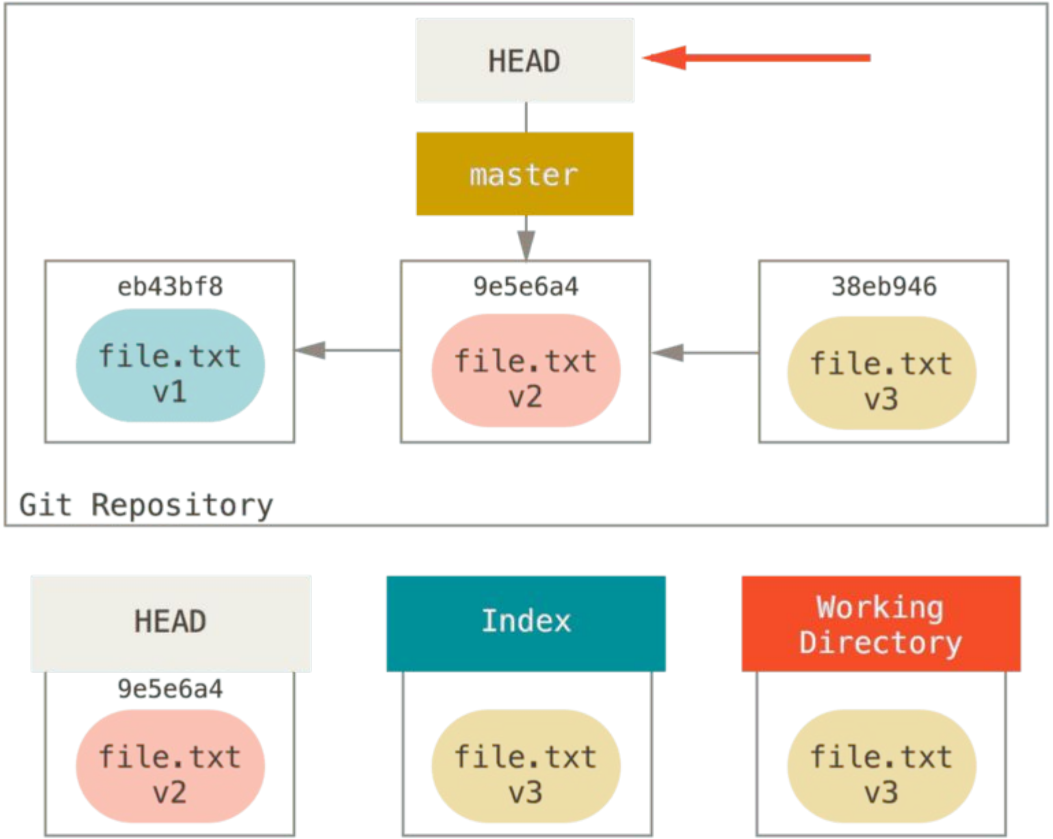
This is on method to reach orphan commits (ones that are not under a branch)

Warning: the history is garbage collected and will only keep track of change to the configured limit (see git-gc)

git reset

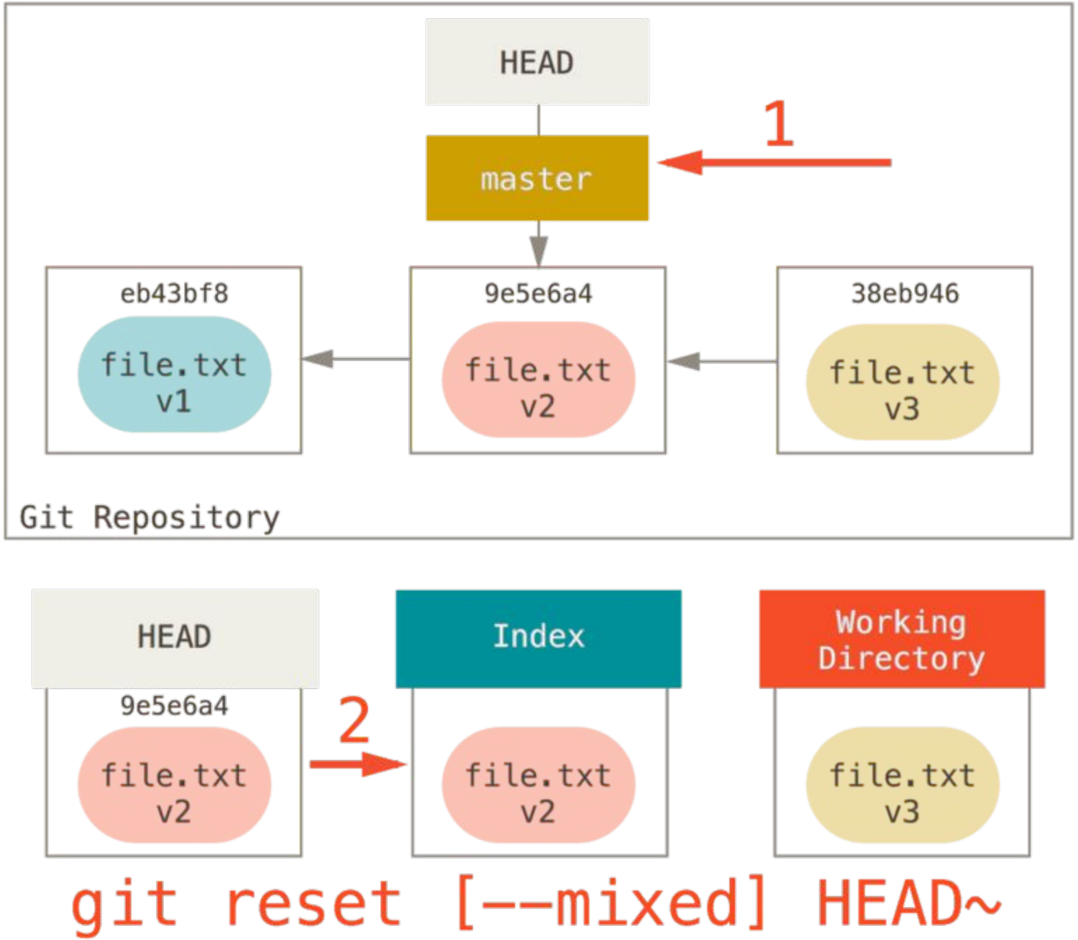

HEAD
master
eb43bf8
file.txt
v1
9e5e6a4
file.txt
v2
38eb946
file.txt
v3
Git Repository
HEAD
38eb946
file.txt
v3
Index
file.txt
v3
Working Directory
file.txt
v3

git reset

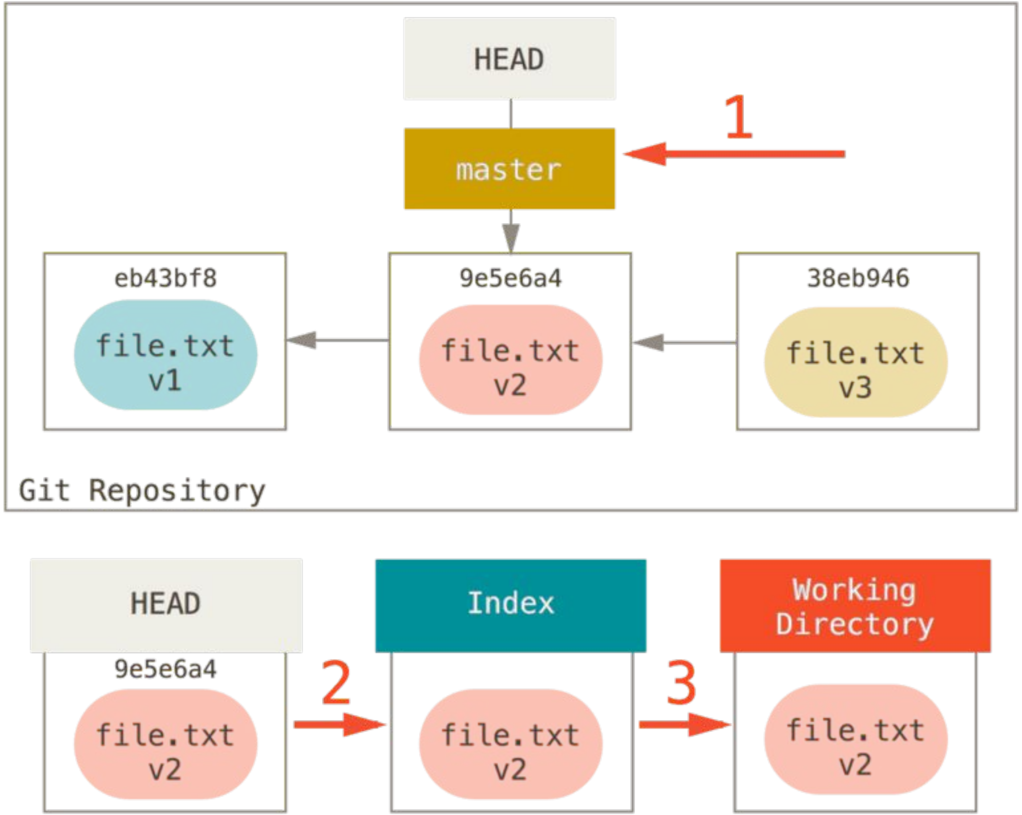


git reset --soft HEAD~

git reset


HEAD
master
1
eb43bf8
file.txt v1
9e5e6a4
file.txt v2
38eb946
file.txt v3
Git Repository
HEAD
9e5e6a4
file.txt v2
2
Index
file.txt v2
Working Directory
file.txt v3
git reset [--mixed] HEAD~

git reset



```
git reset --hard HEAD~
```

A final tip - the BFG

If you accidentally commit some big file (or private data like credentials) in you repo.

Here is your solution:

https://rtyley.github.io/bfg-repo-cleaner/

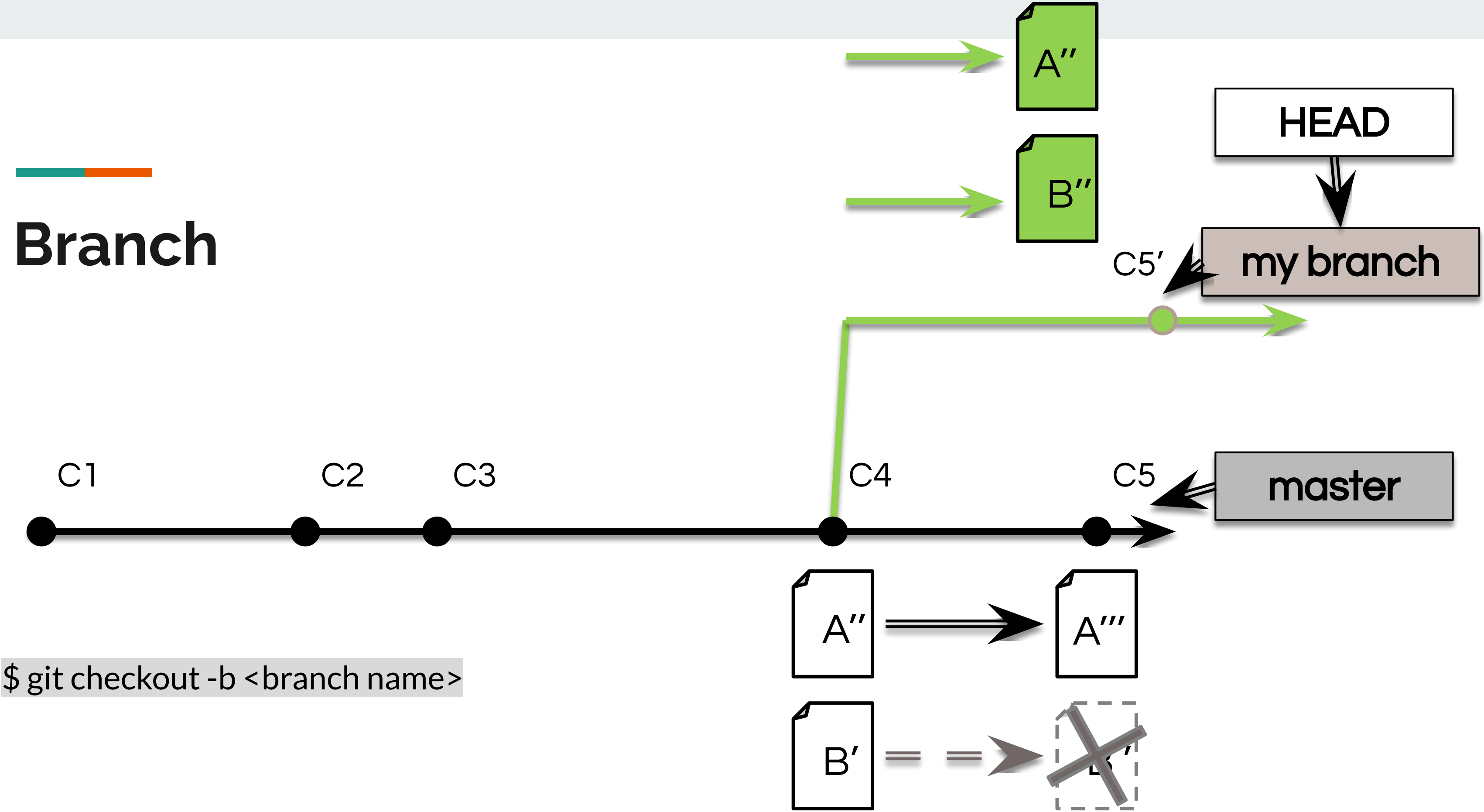
Branching

What is a branch?

A branch is a reference to a commit.

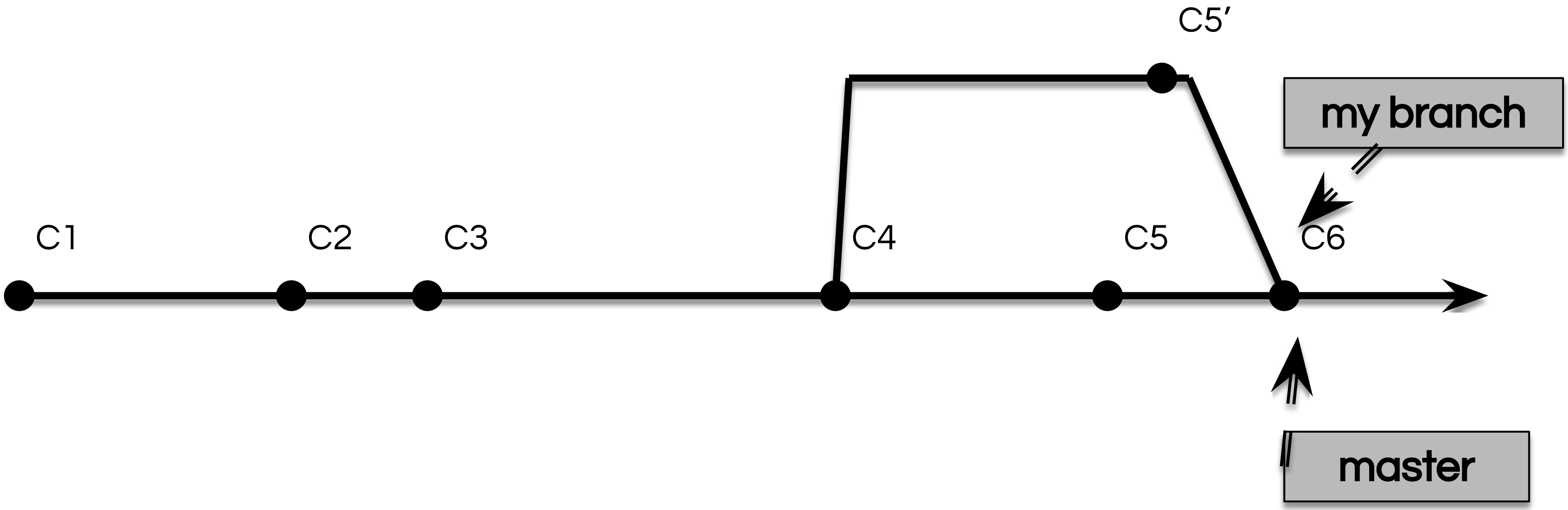
The reference may be moved, typically when a commit is made on a branch reference pointed by HEAD.

Branch

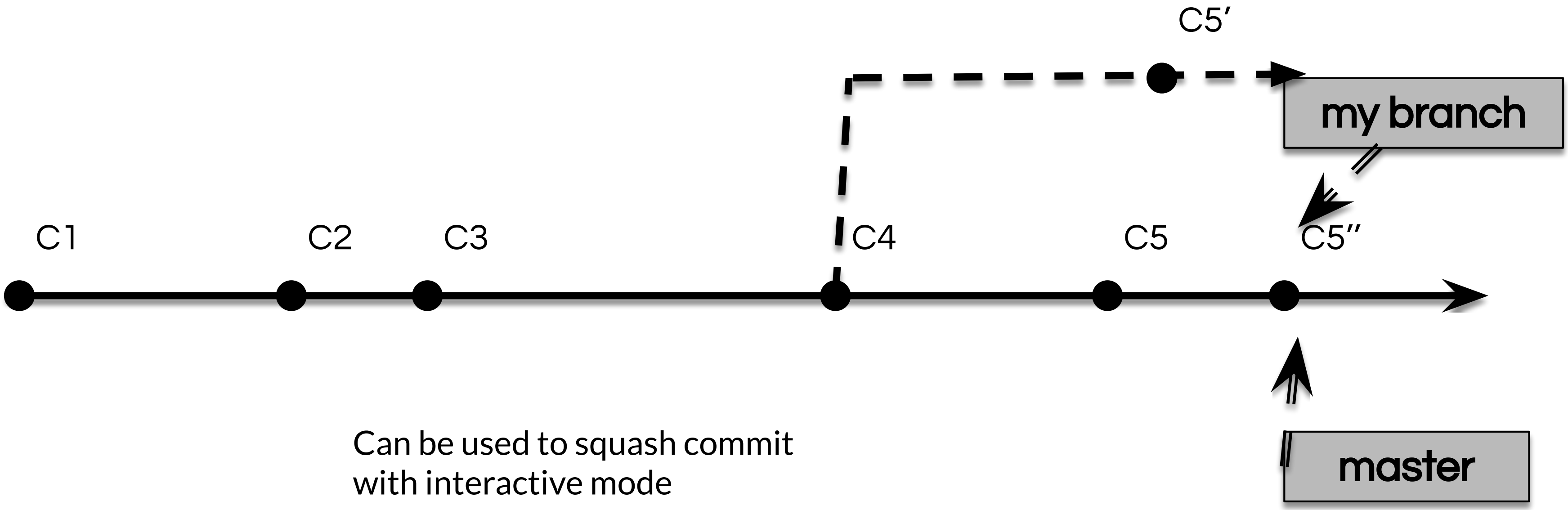


```
$ git checkout -b <branch name>
```

Merge



Rebase



Can be used to squash commit
with interactive mode

Working remotely

Remote

A remote is a named URL pointing to a distant repository

Example:

origin https://github.com/jglouis/tuto_git.git

Where “origin” is the local name of the remote.

Adding a remote

```
git remote add [name of the remote] [URL]
```

Then to inspect configured remotes:

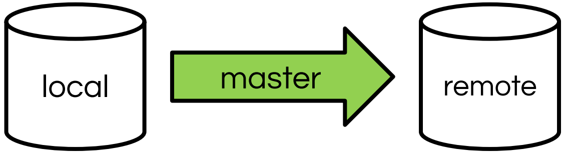
```
git remove -v
```

```
git remote show [name of the remote]
```

Working with a remote

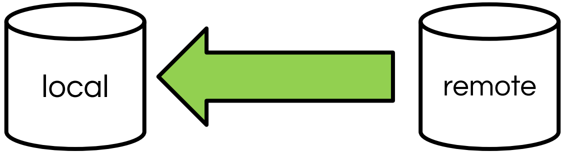
push

```
git push [remote] [branche]
```

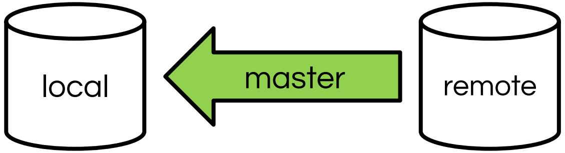


fetch/pull

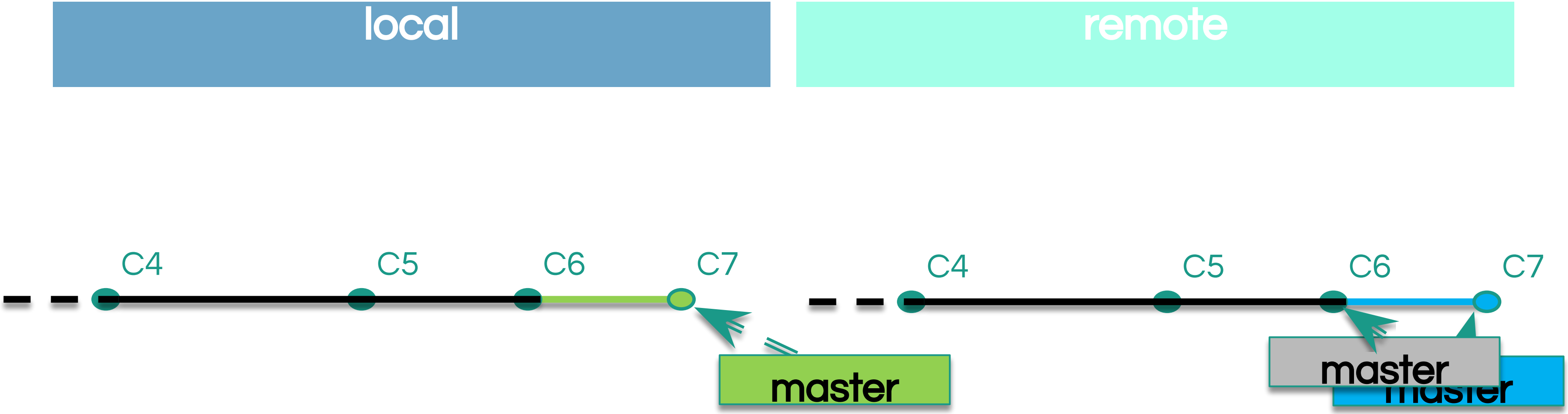
```
git fetch [remote]
```



```
git pull [remote] [branche]
```



push



```
git push [<repository> [<refspec>…]]
```

Setting the upstream branch

This consists in associating a local branch with a remote branch. This way, each time you push or you pull, you don’t have to specify branches

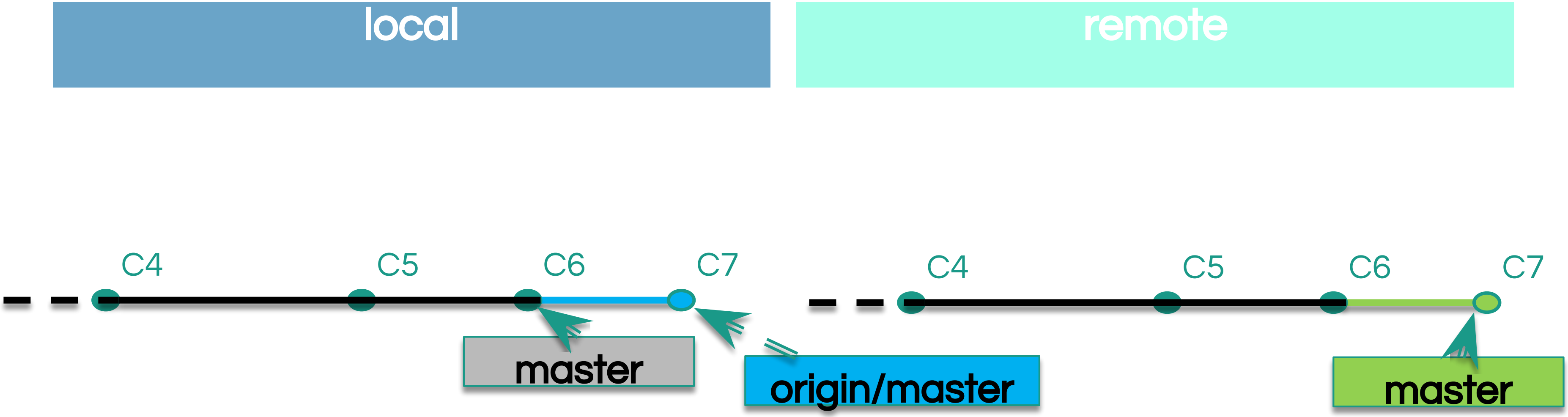
First time you push a branch to a remote, and want to track the association:

```
$ git push -u <remote> <branch>
```

Alternatively: `$ git checkout -b [branch] [remotename]/[branch]`

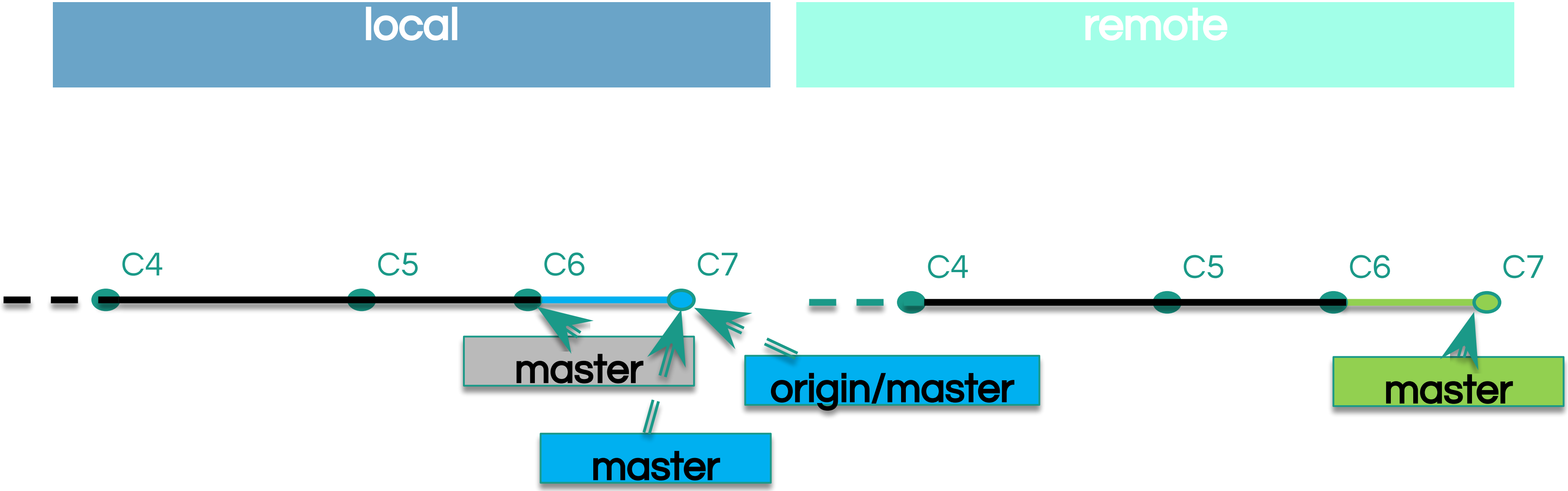
To check how upstream are configured: `$ git branch -vv`

fetch



```
git fetch <remote>
```

pull = fetch + merge



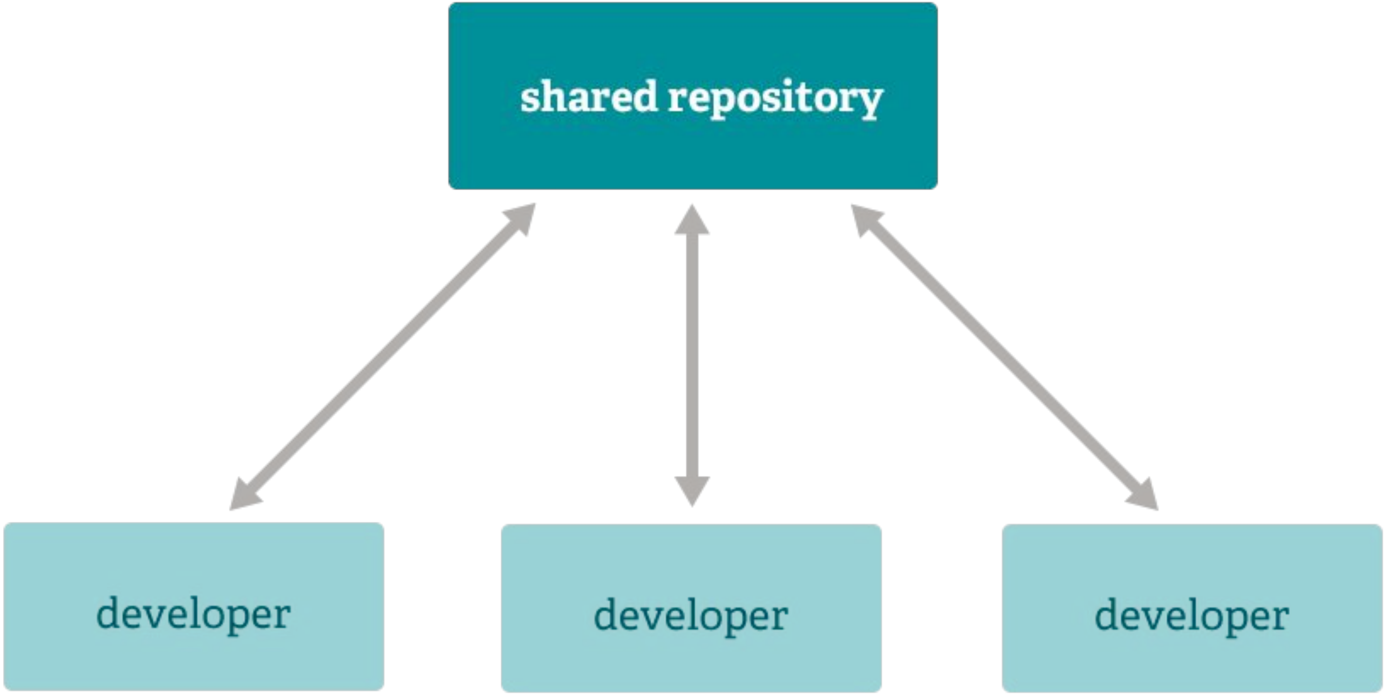
The most important rule when working remotely

Never push a change of history to a remote where other dev are collaborating.

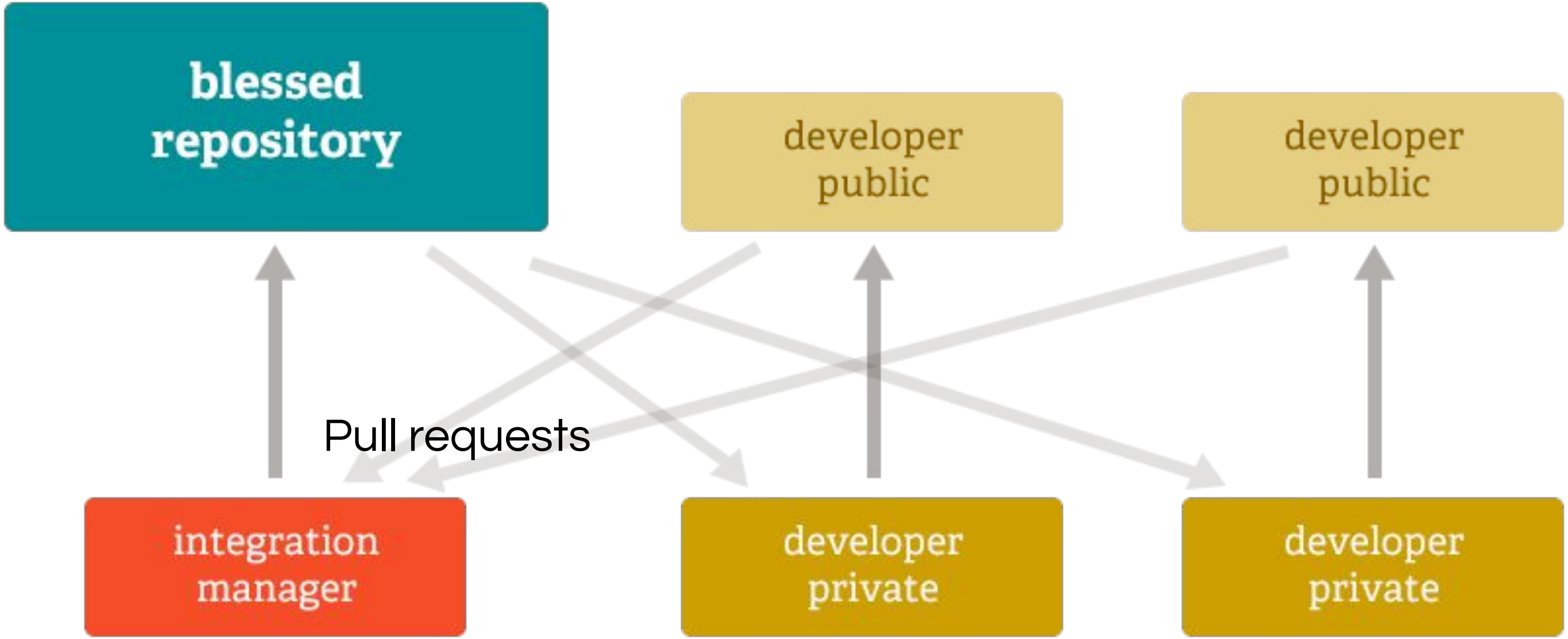
Don’t delete branches, don’t rewrite commit history.

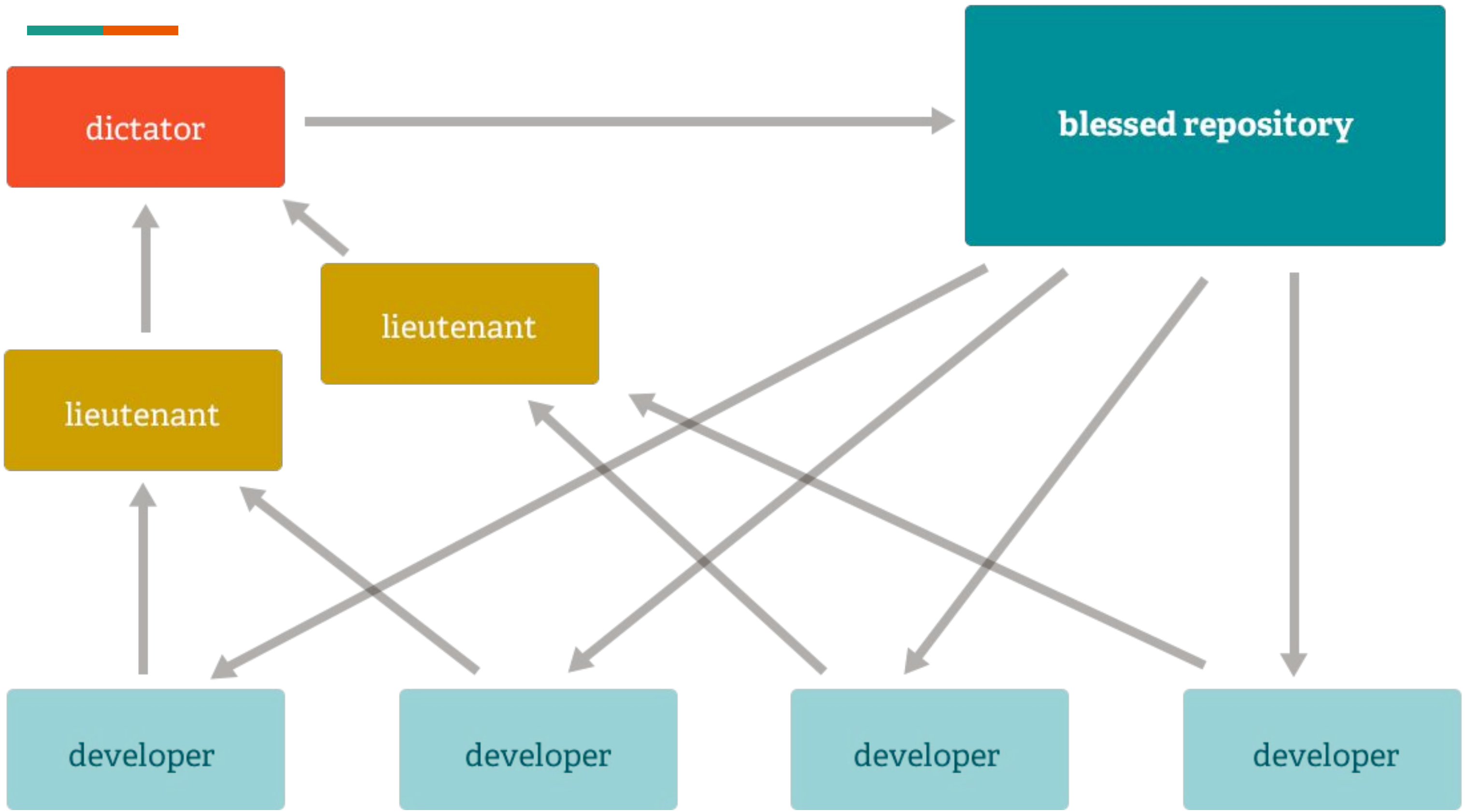
Some common workflows

Centralised



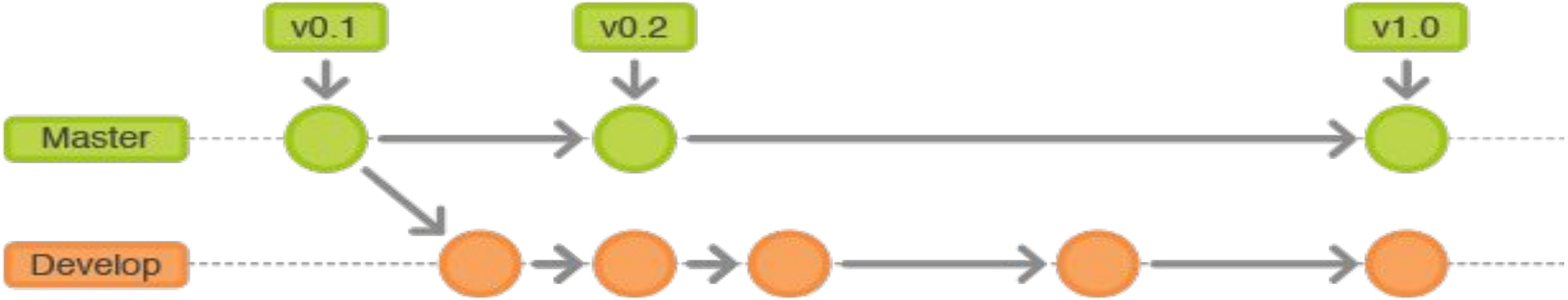
Integration manager



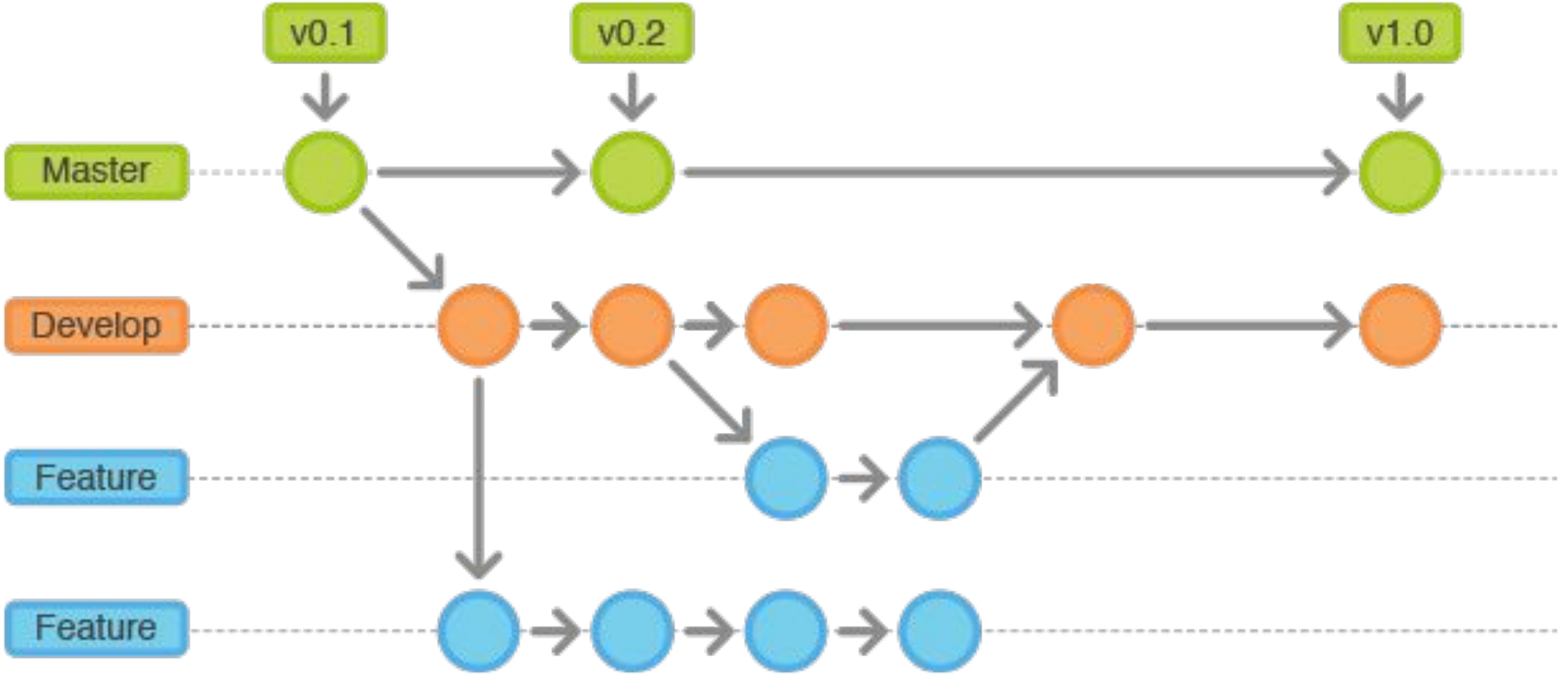

dictator
blessed repository
lieutenant
lieutenant
developer
developer
developer
developer

Gitflow Workflow

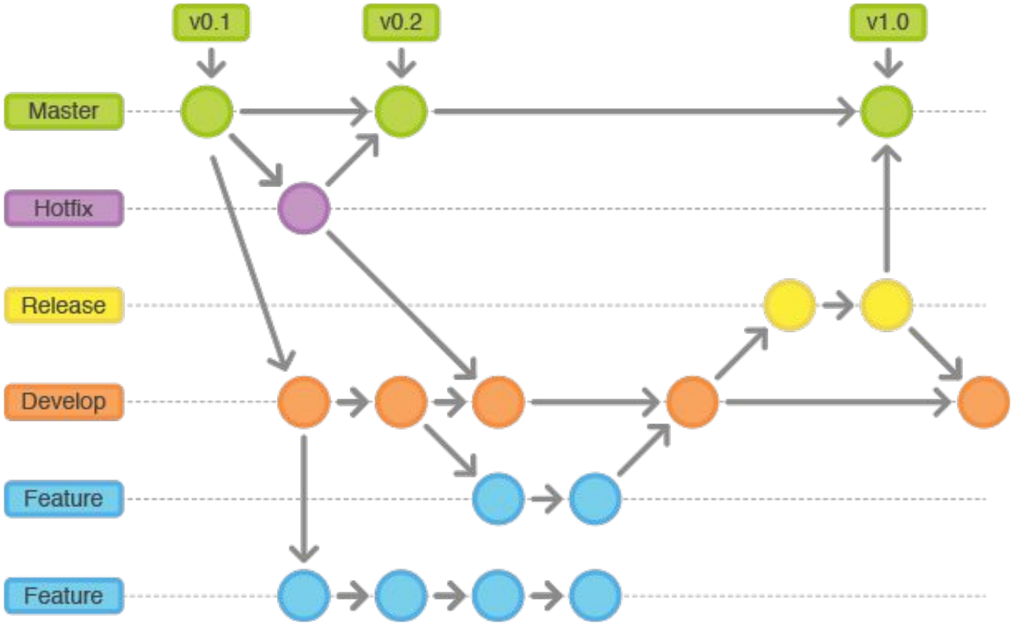
Clean Master



Feature branches

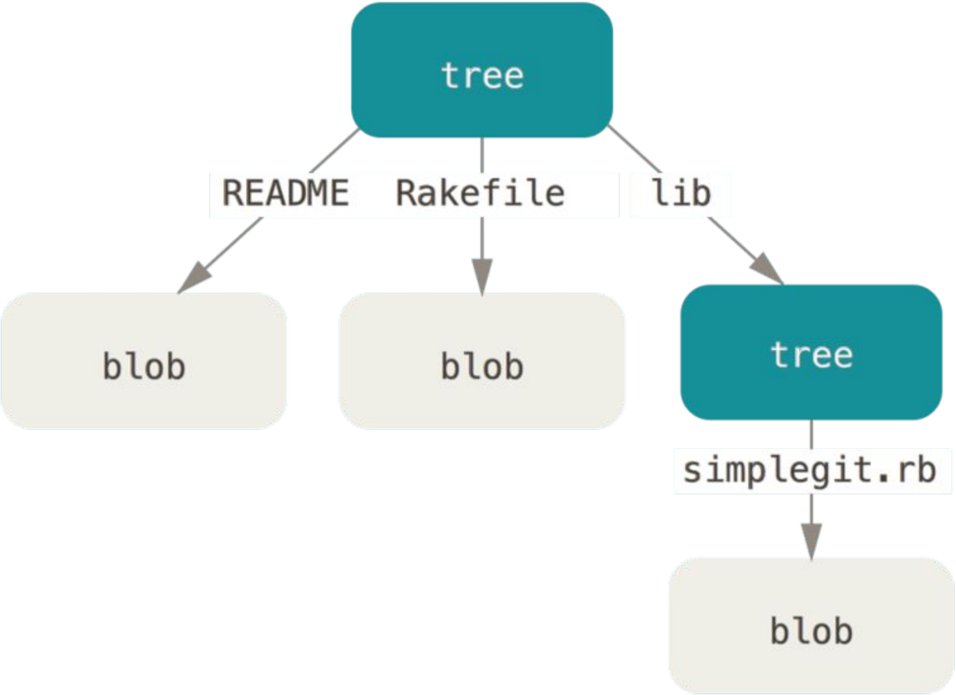


Maintenance branches

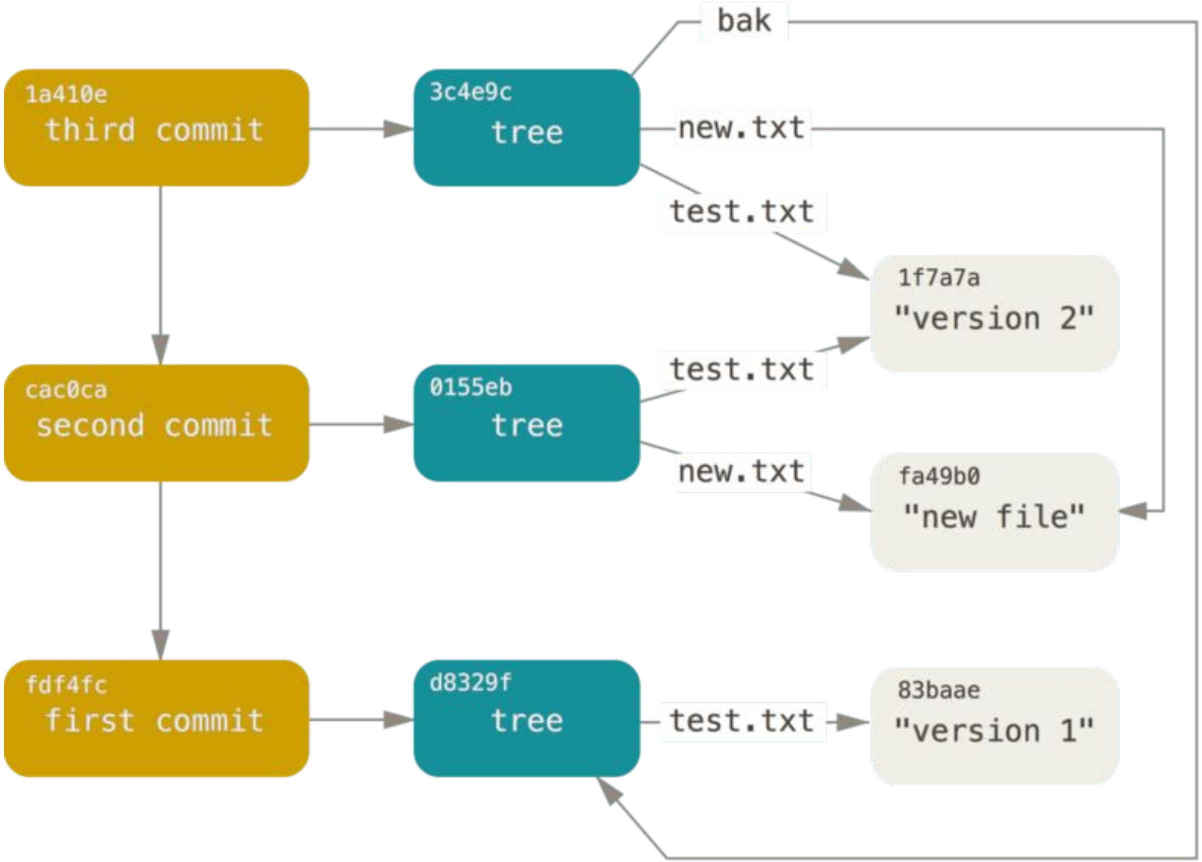


The object model of Git

Tree

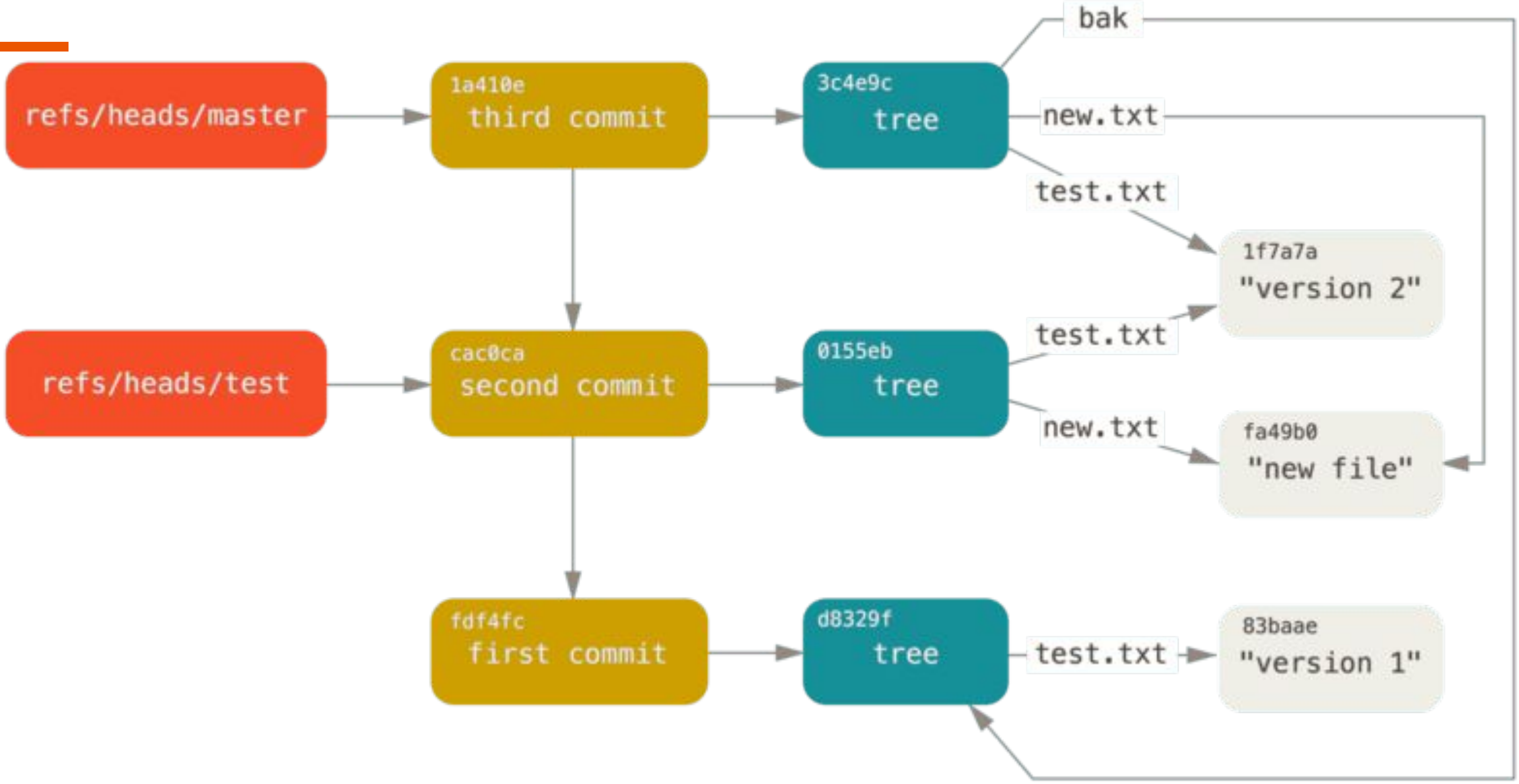


Commits



bak
1a410e
third commit
3c4e9c
tree
new.txt
test.txt
1f7a7a
"version 2"
test.txt
cac0ca
second commit
0155eb
tree
new.txt
fa49b0
"new file"
fdf4fc
first commit
d8329f
tree
test.txt
83baae
"version 1"

Git references (branches)



Tags

Like a branch reference, except it never moves.

Useful for tagging a version or an important commit.

Some general tips

Tips

- Prefer small commits
- Commit often - one commit for each logical change
- Pay attention to the formatting of your commit message
- `git stash`is a very handy tool when you want to keep some changes locally, but don’t want to publish them

A word on Git LFS

“Large File Storage”

https://git-lfs.github.com/

Large files

- Git is suited for versioning text files (compression / diff)
- Big binary files tend to clutter the repository and take a lot of space.
- A big repository can become slow and some hosting services may apply limits to the size of a repository or a stored file.

Git LFS

It is both a git plugin and a paid service from github.com

It basically allows to store files outside of the repository on an external service. The Git repository is then only storing pointers to theses resources.