CSE 391
Lecture 5

Branching with Git

slides created by Zorah Fung

http://www.cs.washington.edu/391/

Problems Working on Master

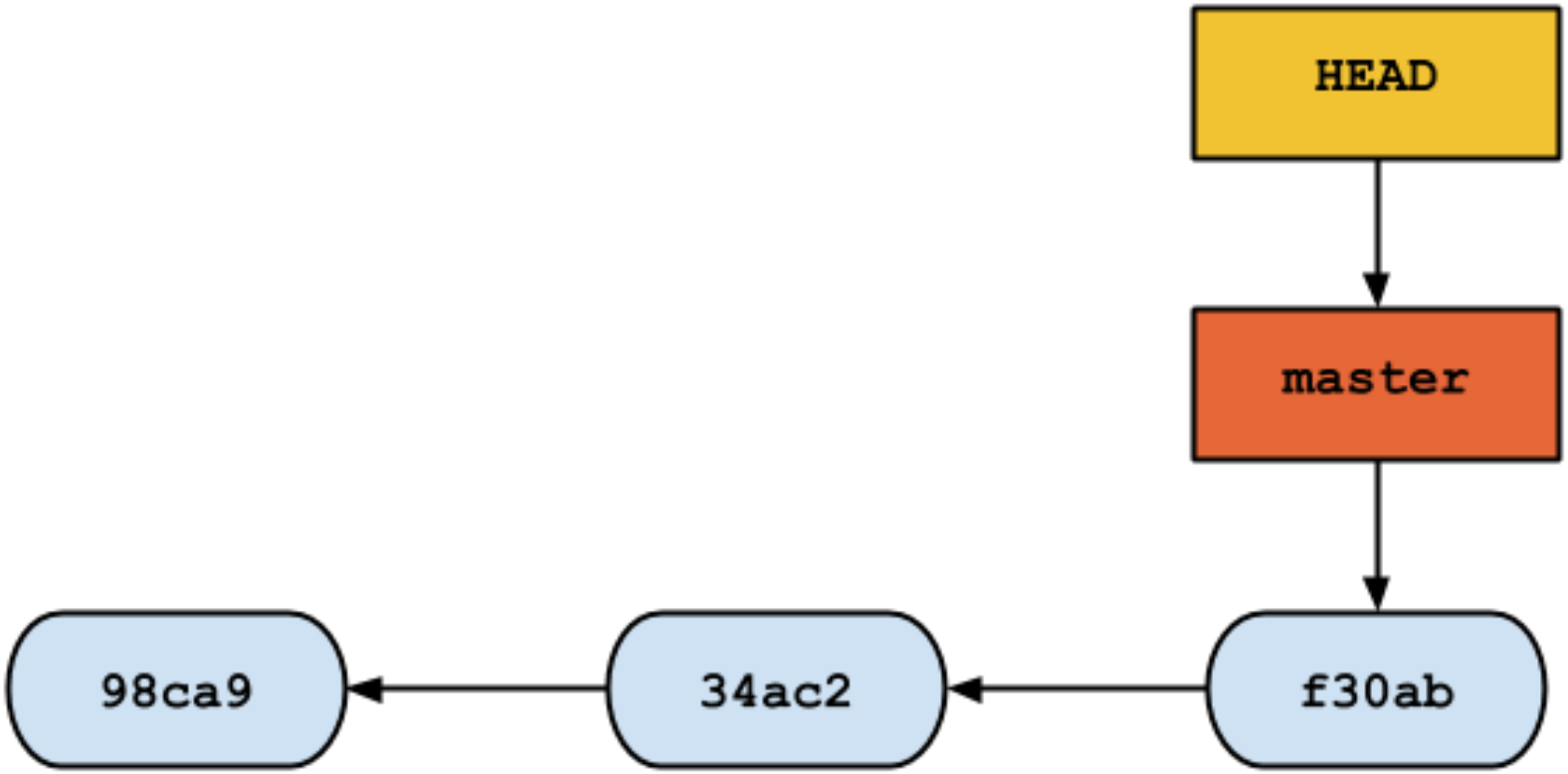
- Difficult to work on two different unrelated features at time
- No way to share work in progress remotely
- Want to isolate changes related to a particular feature or project
- If you push a bad change, the repository is in a bad state for everyone

- Master should represent the “single source of truth” that everyone works from.
- Master should “stay clean”. The code in master should compile and produce the correct behavior.

Solution: Branching

- **branch**: A named sequence of commits representing a state of the git repository
- The tip of a branch is the most recent commit on a branch
- Every time you make a commit, the tip of your current branch moves forward one commit
- **HEAD**: the pointer to the commit currently being referenced by the repository. We say “we’re on master” when HEAD points to the tip of master.
- **master** is the default first branch name in a repository, but there’s nothing “special” about it. It’s just another branch.

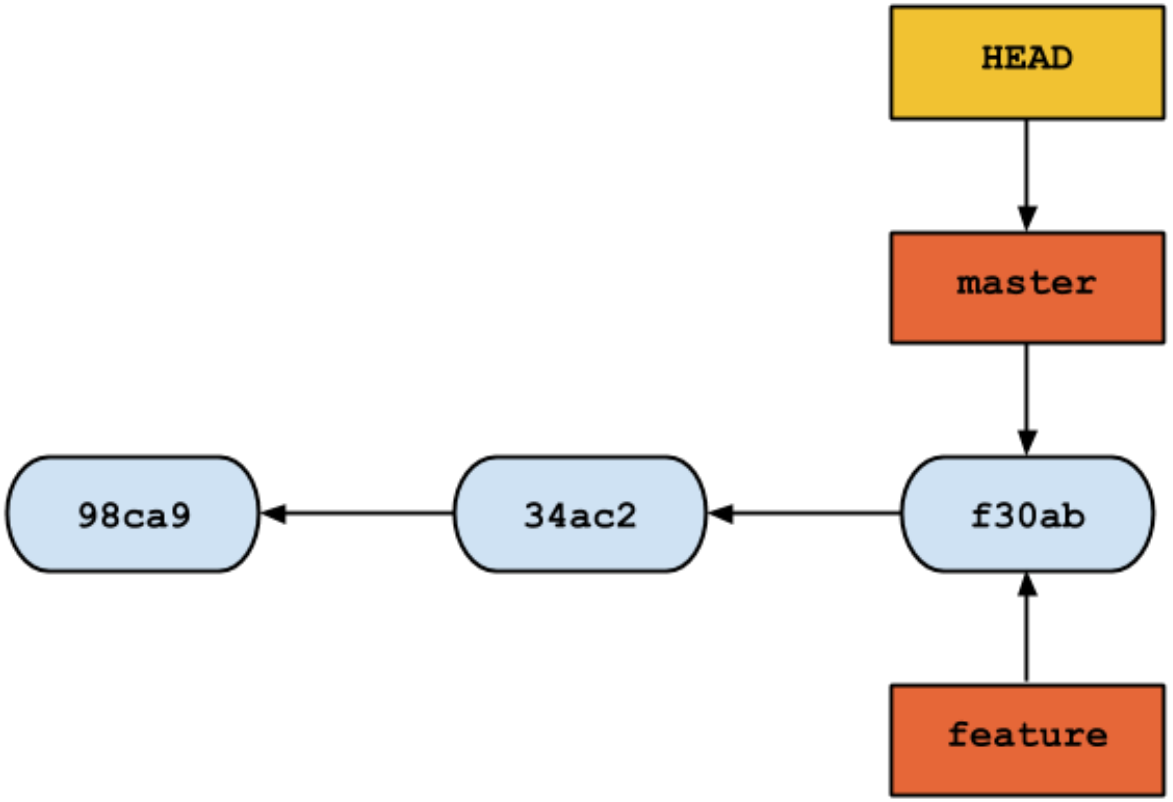
Master Branch



Creating a new branch

When you create a new branch, the new branch will point to the same series of commits as the current branch

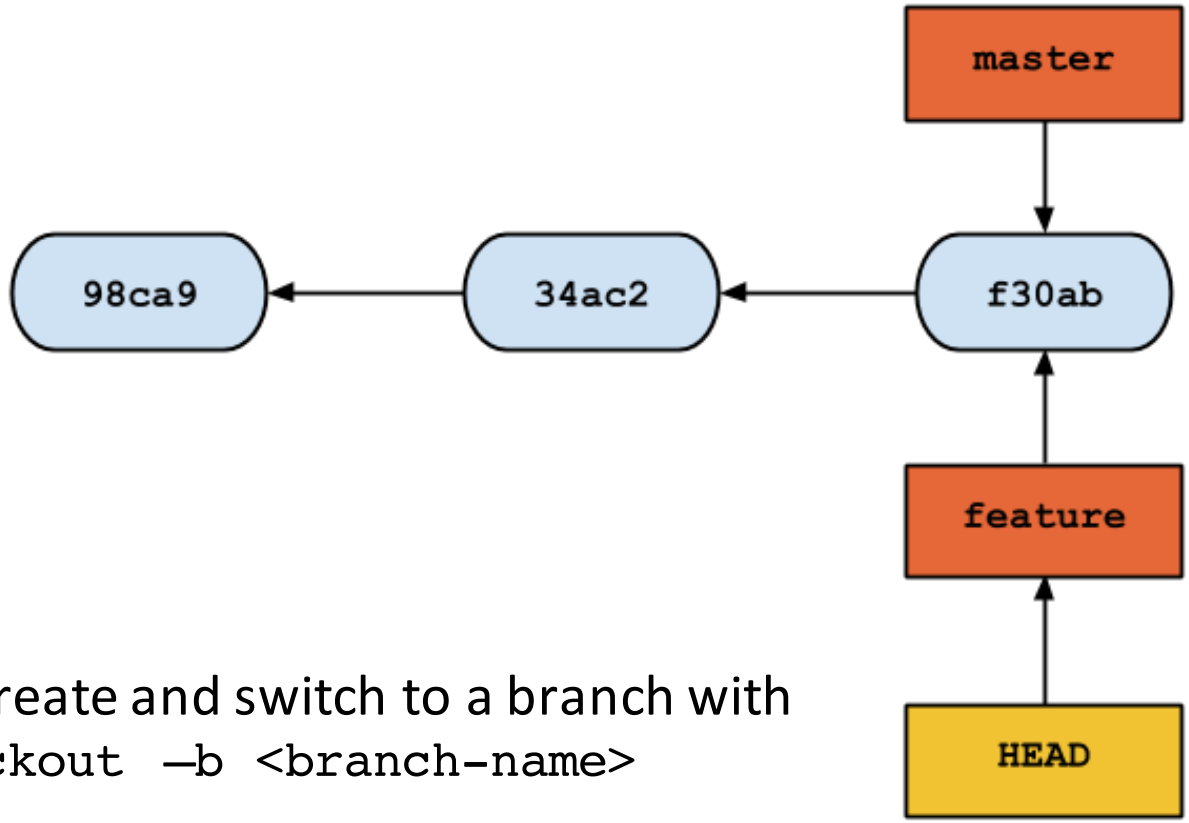
```
git branch <branch-name>
```



Switching branches

To switch your current branch to another branch, you ”checkout” the branch

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

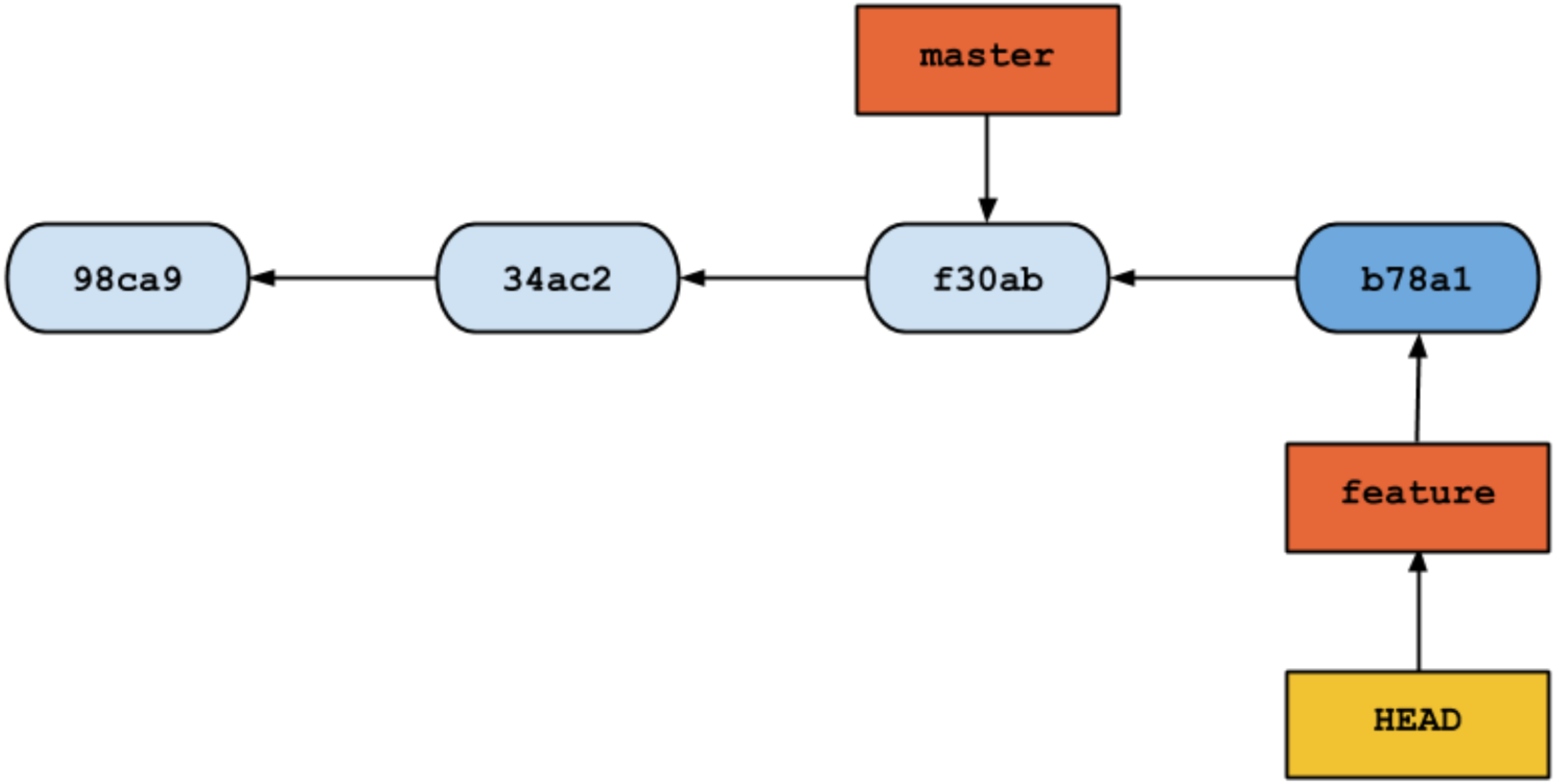


You can create and switch to a branch with

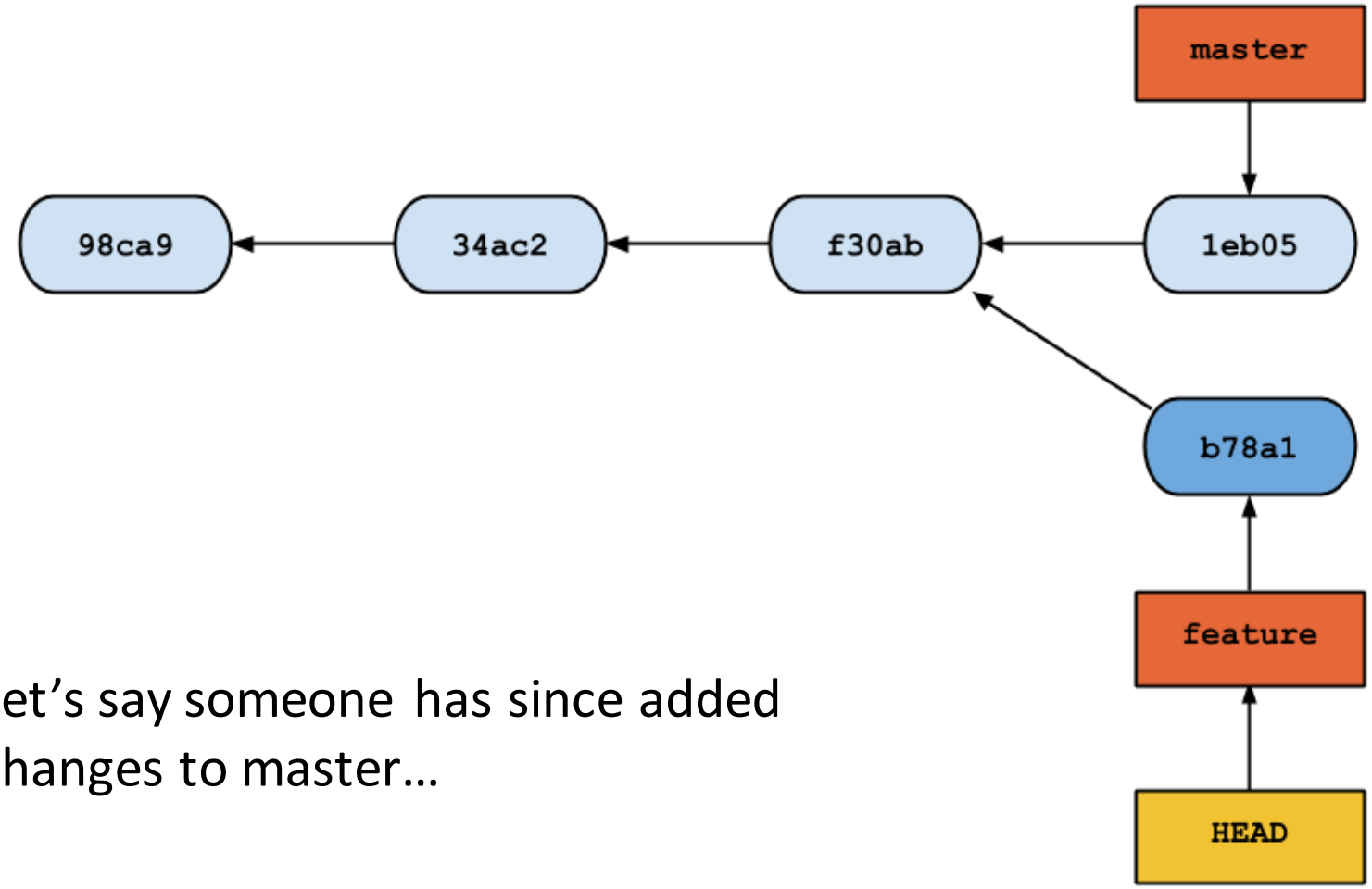
```
git checkout –b <branch-name>
```

Committing to branches

- All commits are made to your current branch
- When you make a commit, the tip of your current branch and HEAD move forward

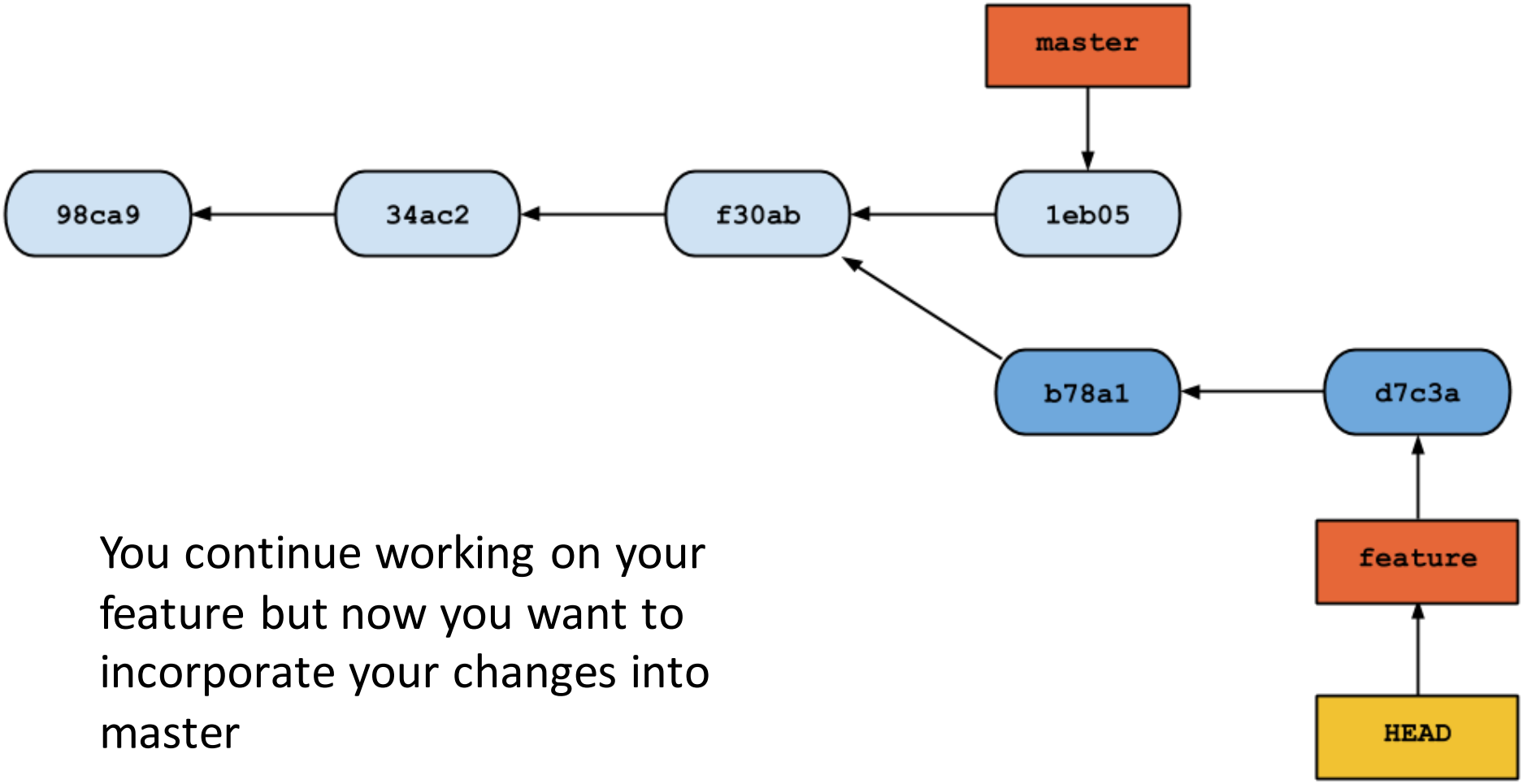


Branch Divergence



Let’s say someone has since added changes to master…

Branch Divergence

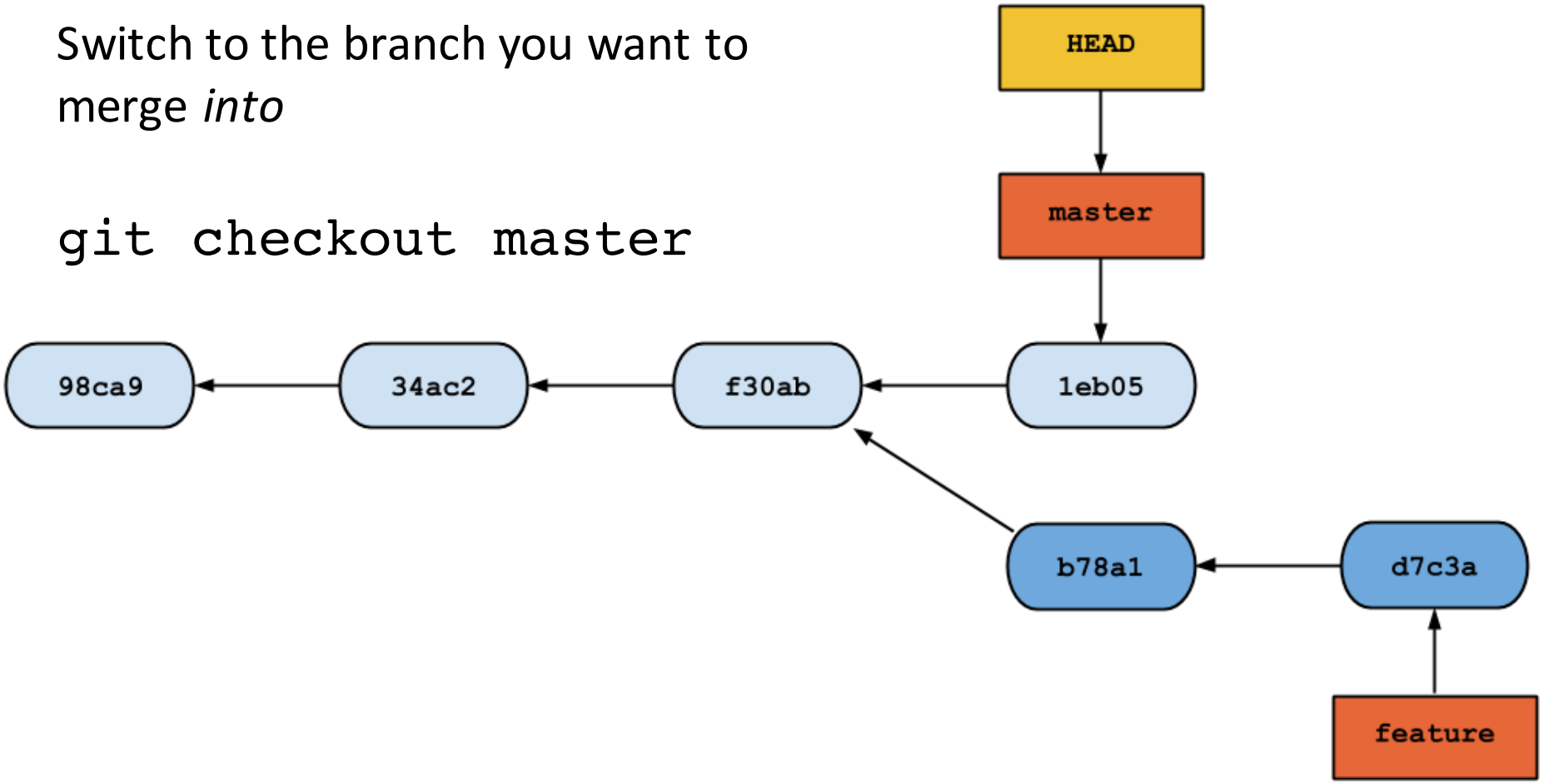


You continue working on your feature but now you want to incorporate your changes into master

Merging branches

Switch to the branch you want to merge *into*

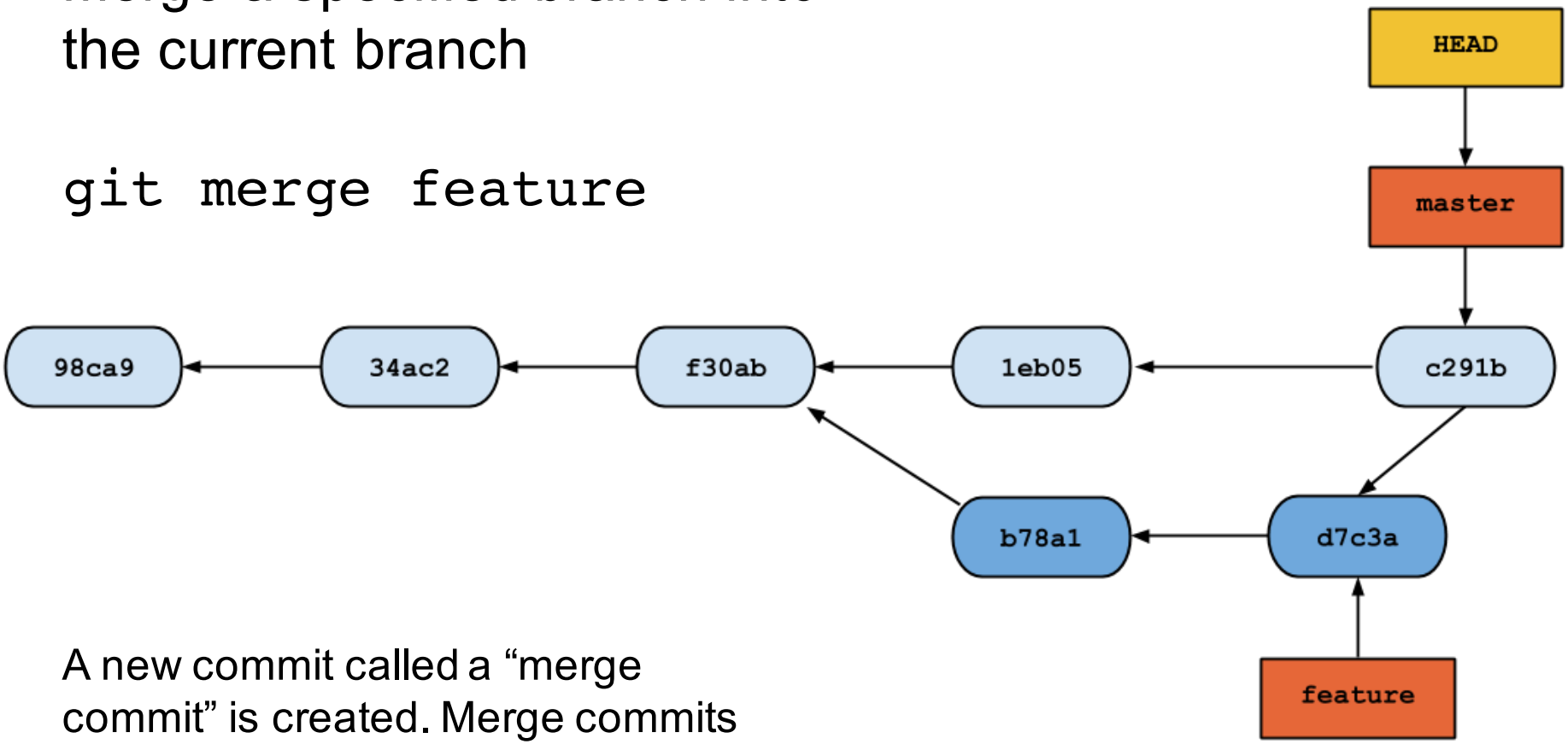
```
git checkout master
```



Merging branches

Merge a specified branch *into* the current branch

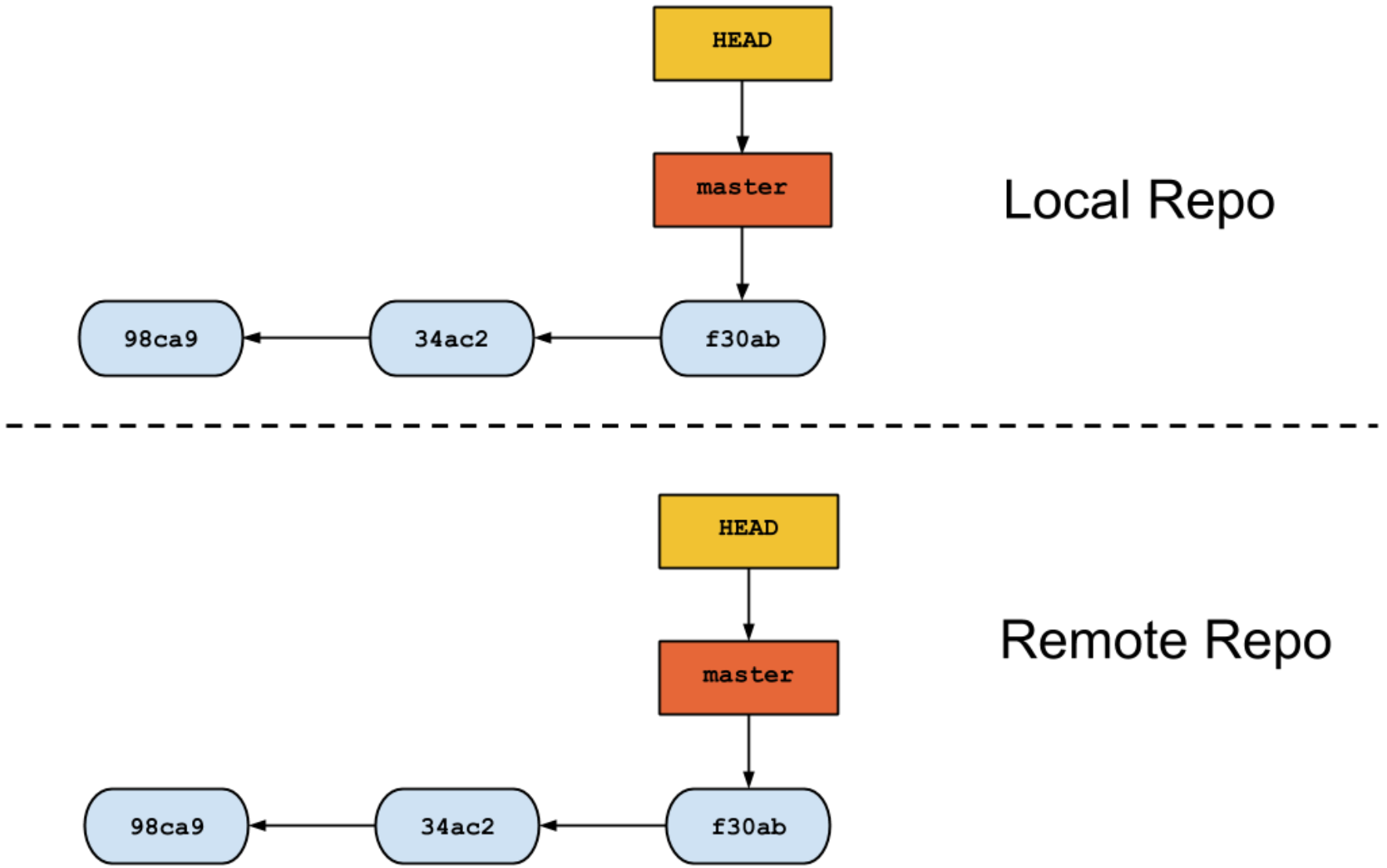
```
git merge feature
```



A new commit called a “merge commit” is created. Merge commits have two parent commits

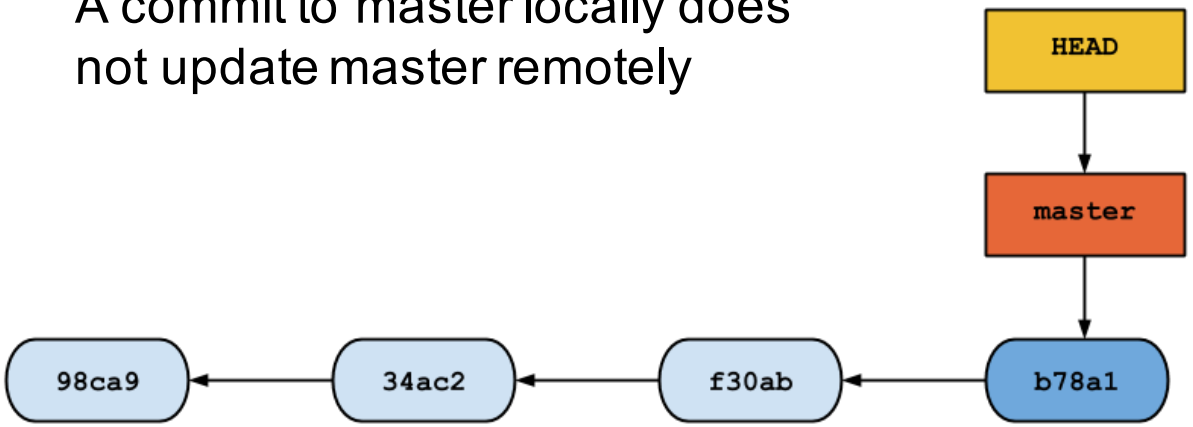
Local and Remote branches

- Local and remote repositories have their own copies of branches
- Branches on your local repository and remote repository are tracked independently
- Branches created locally need to be pushed to be created on the remote repository
- Branches on the remote repository need to be fetched (more on fetching later) to be created locally

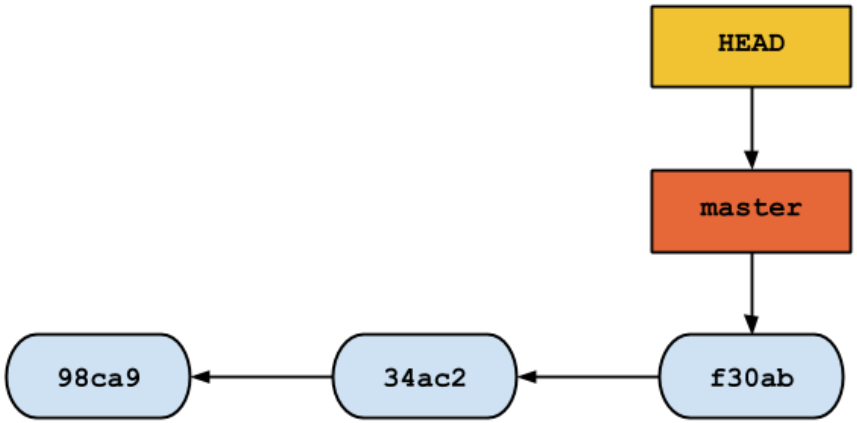

HEAD
master
98ca9
34ac2
f30ab
Local Repo
HEAD
master
98ca9
34ac2
f30ab
Remote Repo

Local and Remote branches

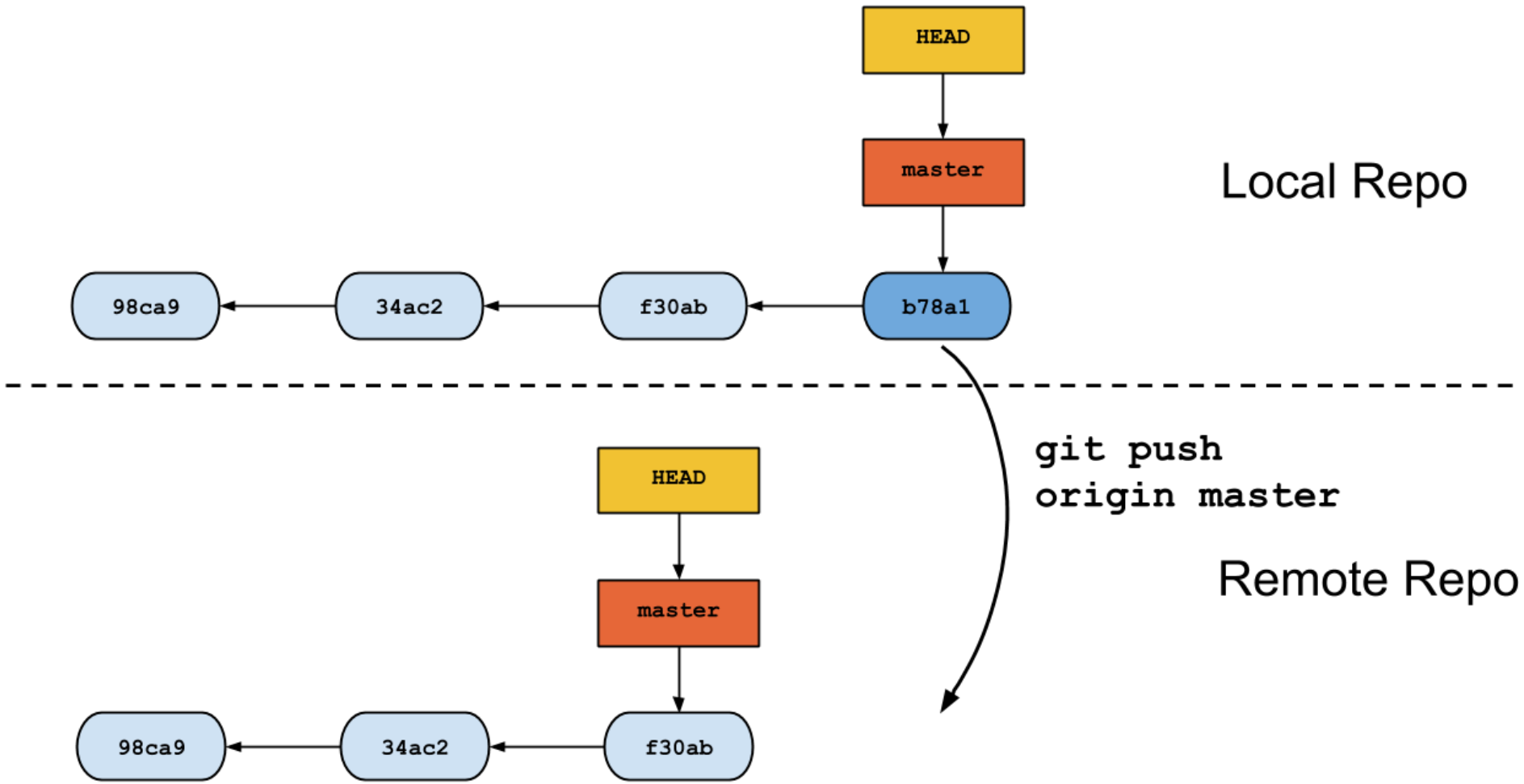
A commit to master locally does not update master remotely

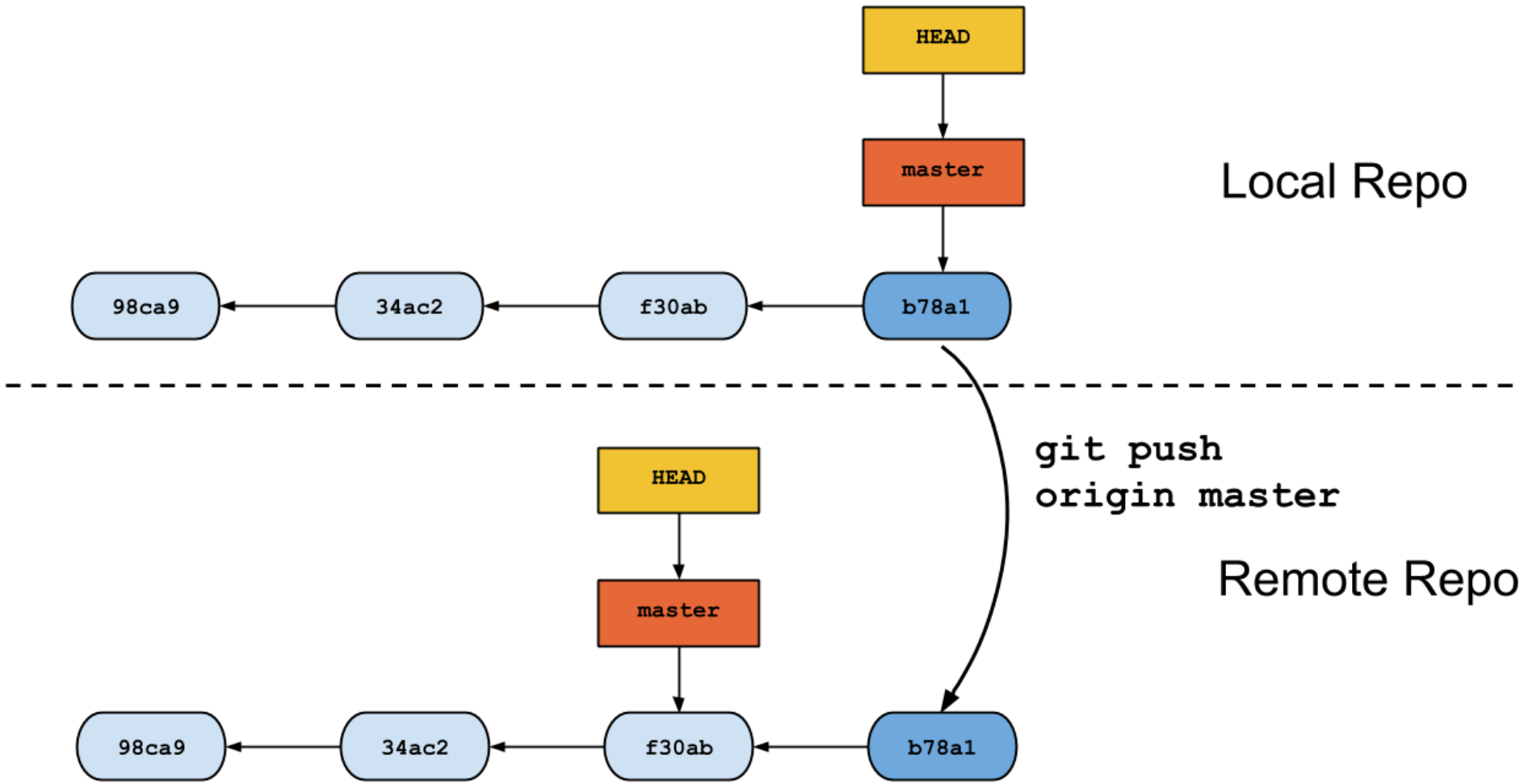


Local Repo

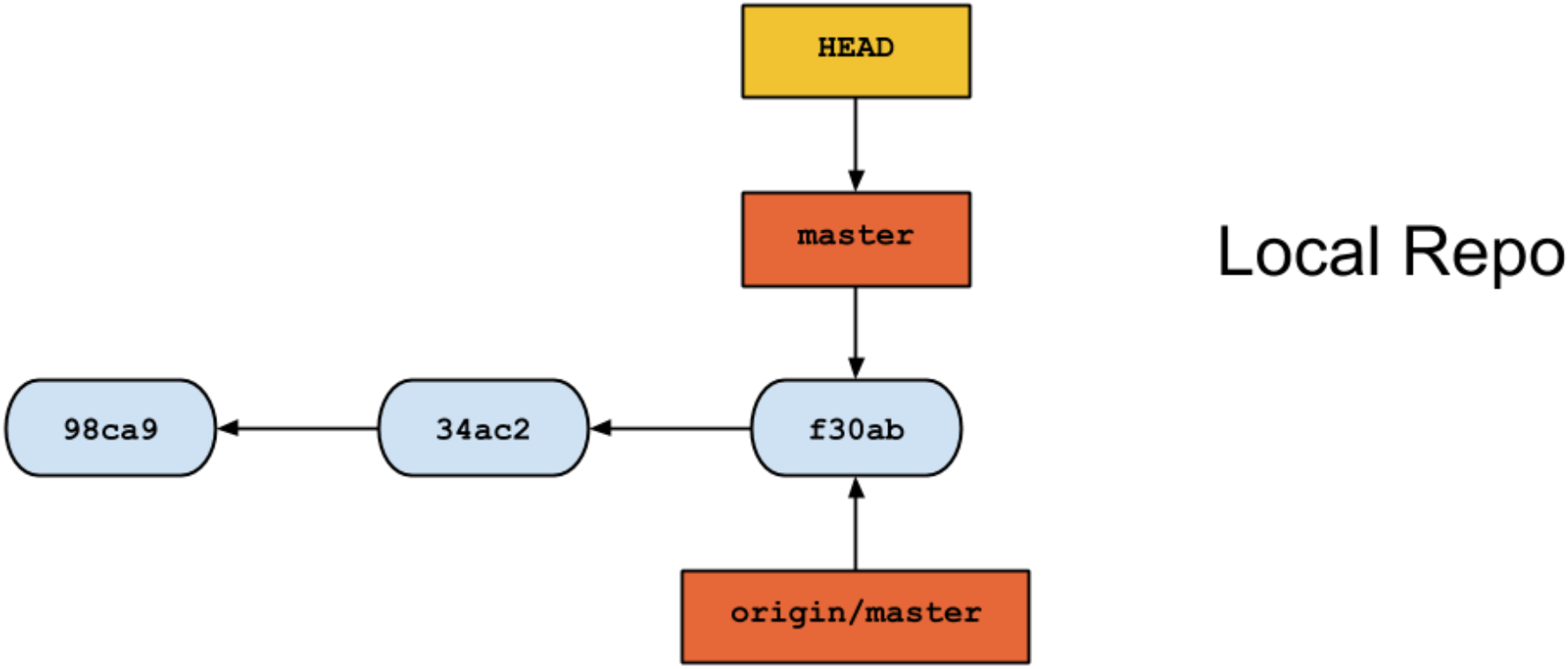


Remote Repo

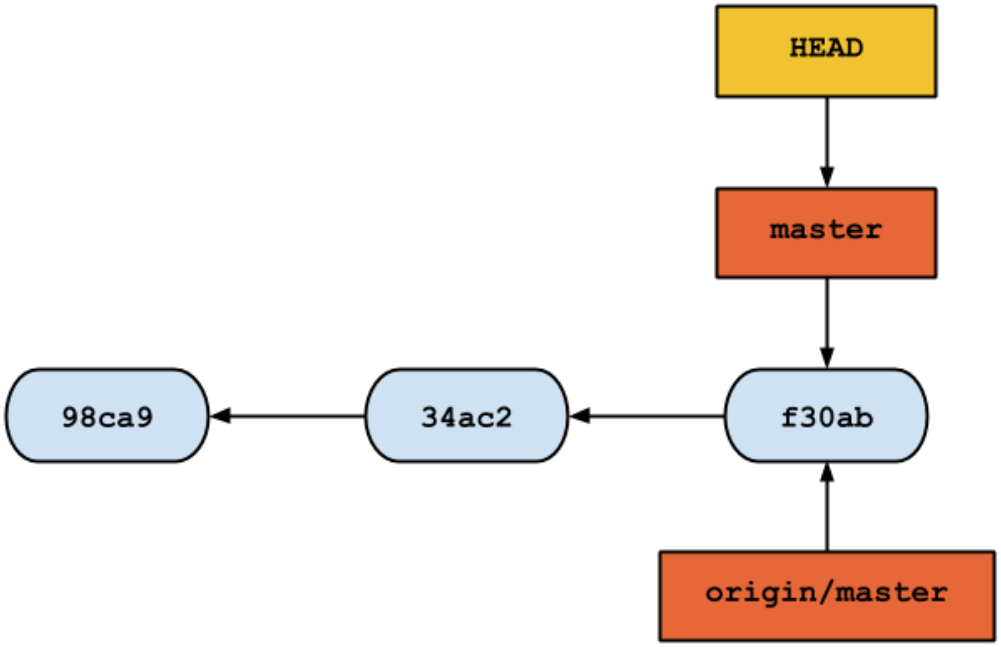

HEAD
master
Local Repo
98ca9
34ac2
f30ab
b78a1
git push
origin master
HEAD
master
Remote Repo
98ca9
34ac2
f30ab


HEAD
master
Local Repo
98ca9
34ac2
f30ab
b78a1
git push
origin master
HEAD
master
Remote Repo
98ca9
34ac2
f30ab
b78a1

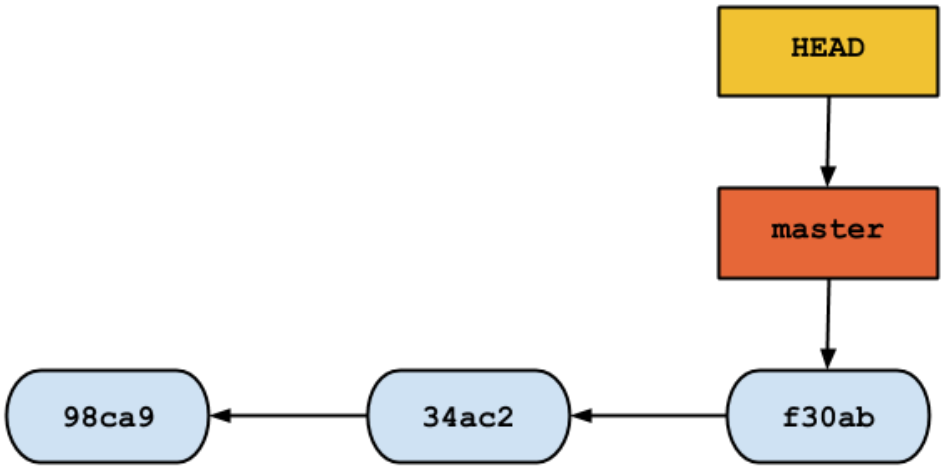
Remote branch references



In your local repository, you will have branches that reference that last known state of a remote branch. They will be prefixed with `origin/`

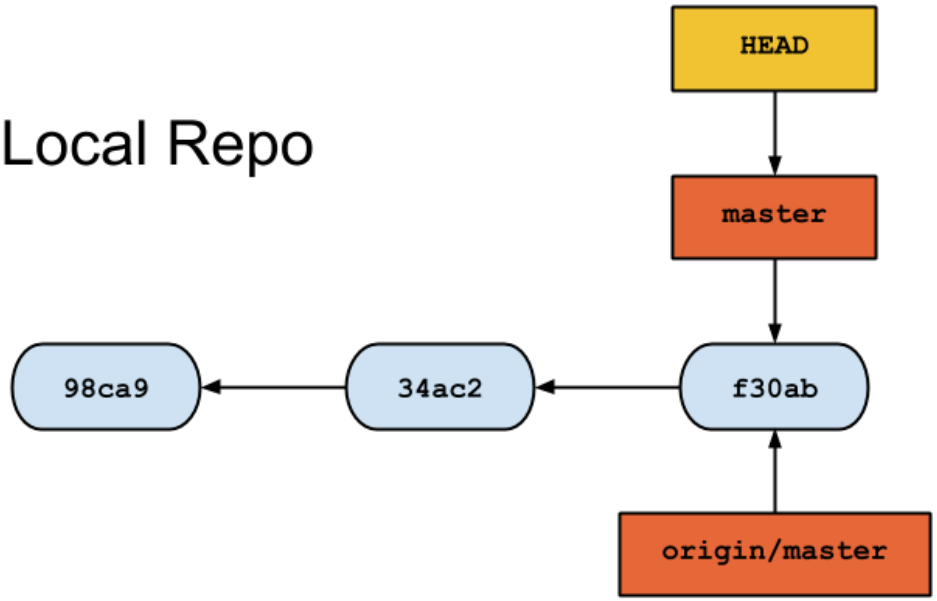


Local Repo

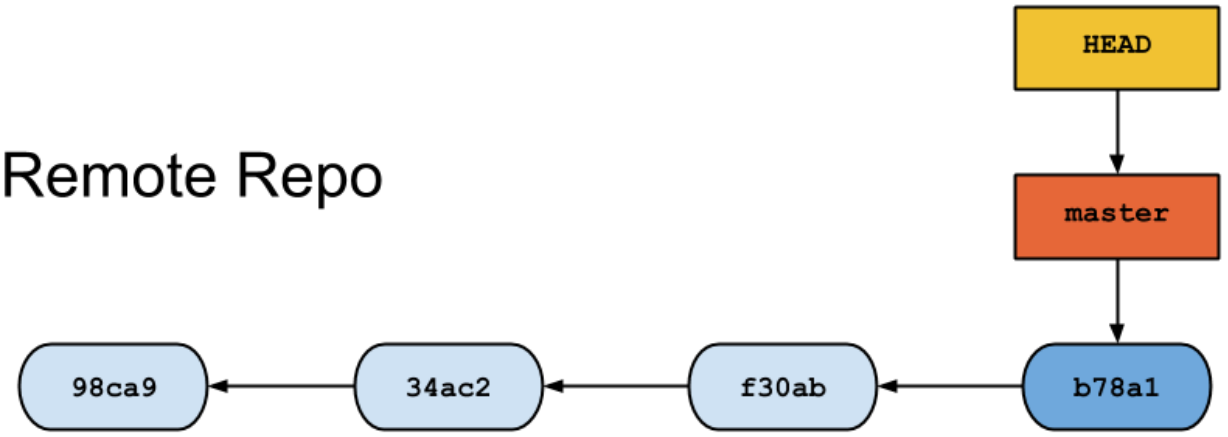


Remote Repo

Local Repo



Remote Repo



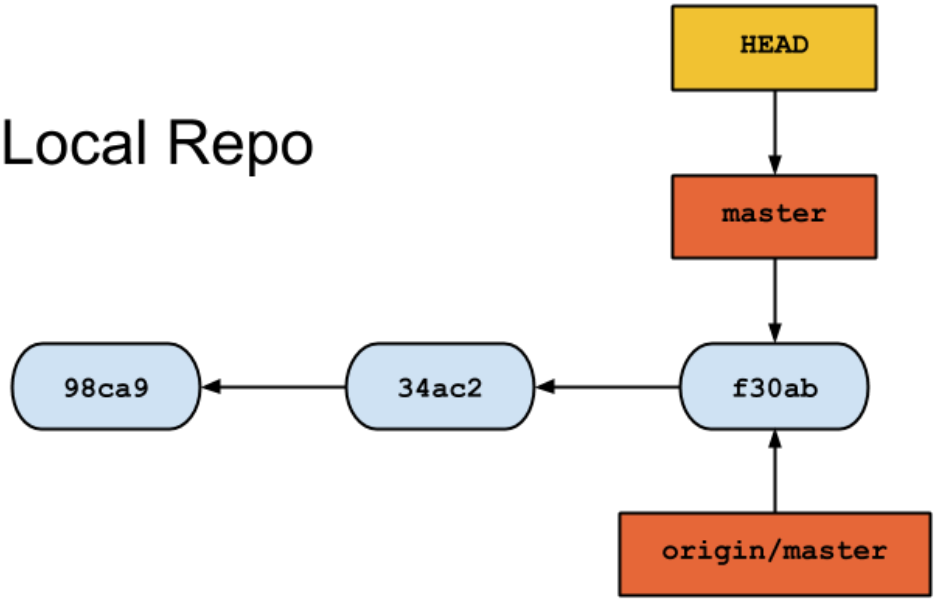
An update to master on the remote repo does NOT automatically change your local `origin/master` branch

git pull

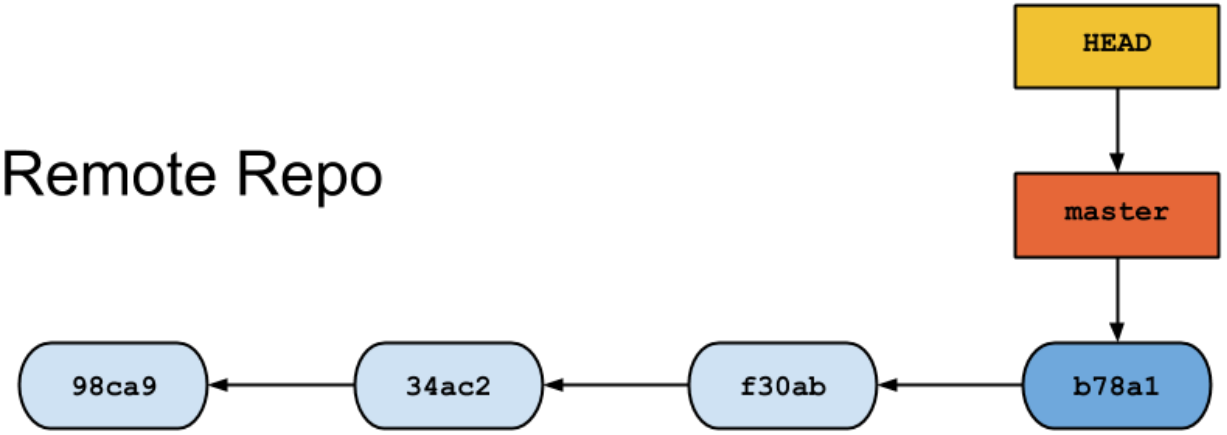
git pull

- `git pull origin branch` is actually a short hand for a `git fetch origin <branch>` followed by a `git merge origin/<branch>`
- `git fetch` will update your `origin/<branch>` to match the `<branch>` on the remote repository
- `git merge` will merge your `origin/<branch>` into your current local `<branch>`

Local Repo

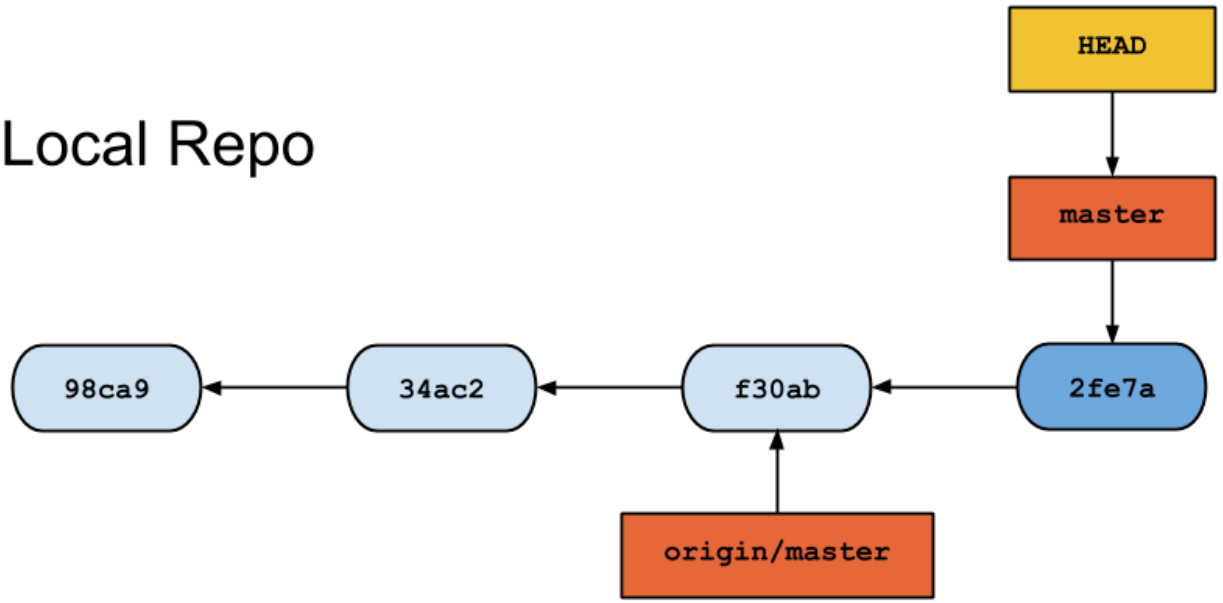


Remote Repo



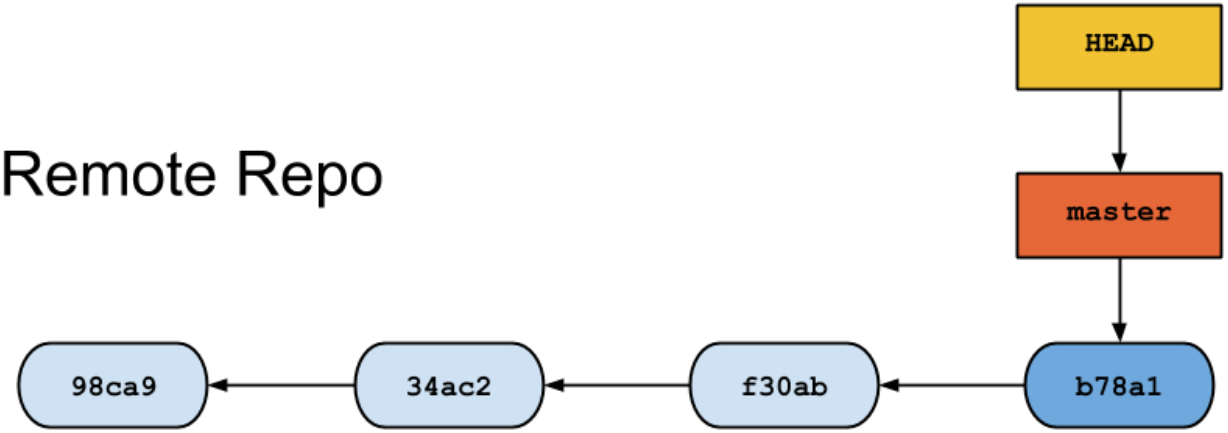
Someone makes some changes to the remote master branch

Local Repo

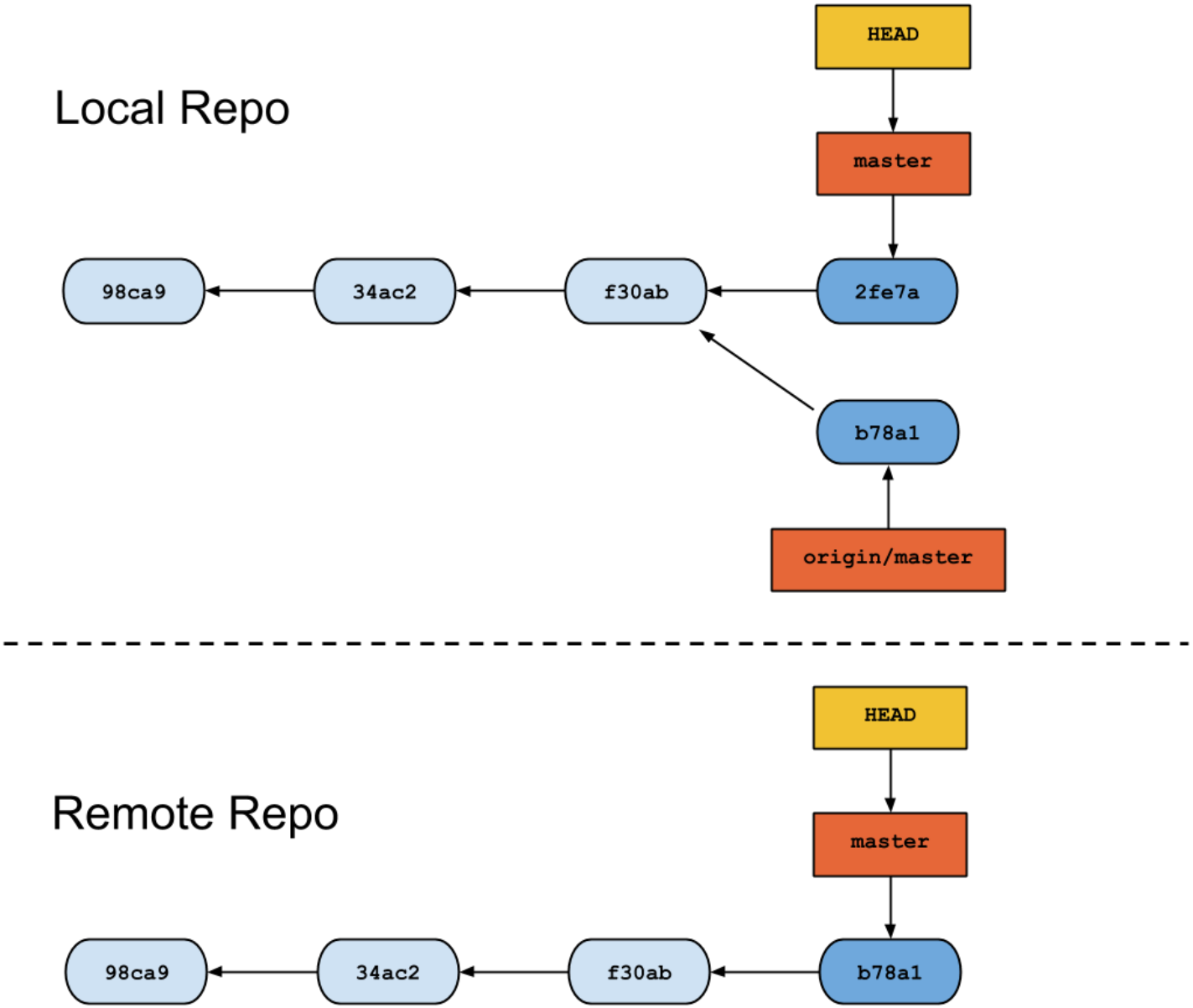


You make changes to local master branch

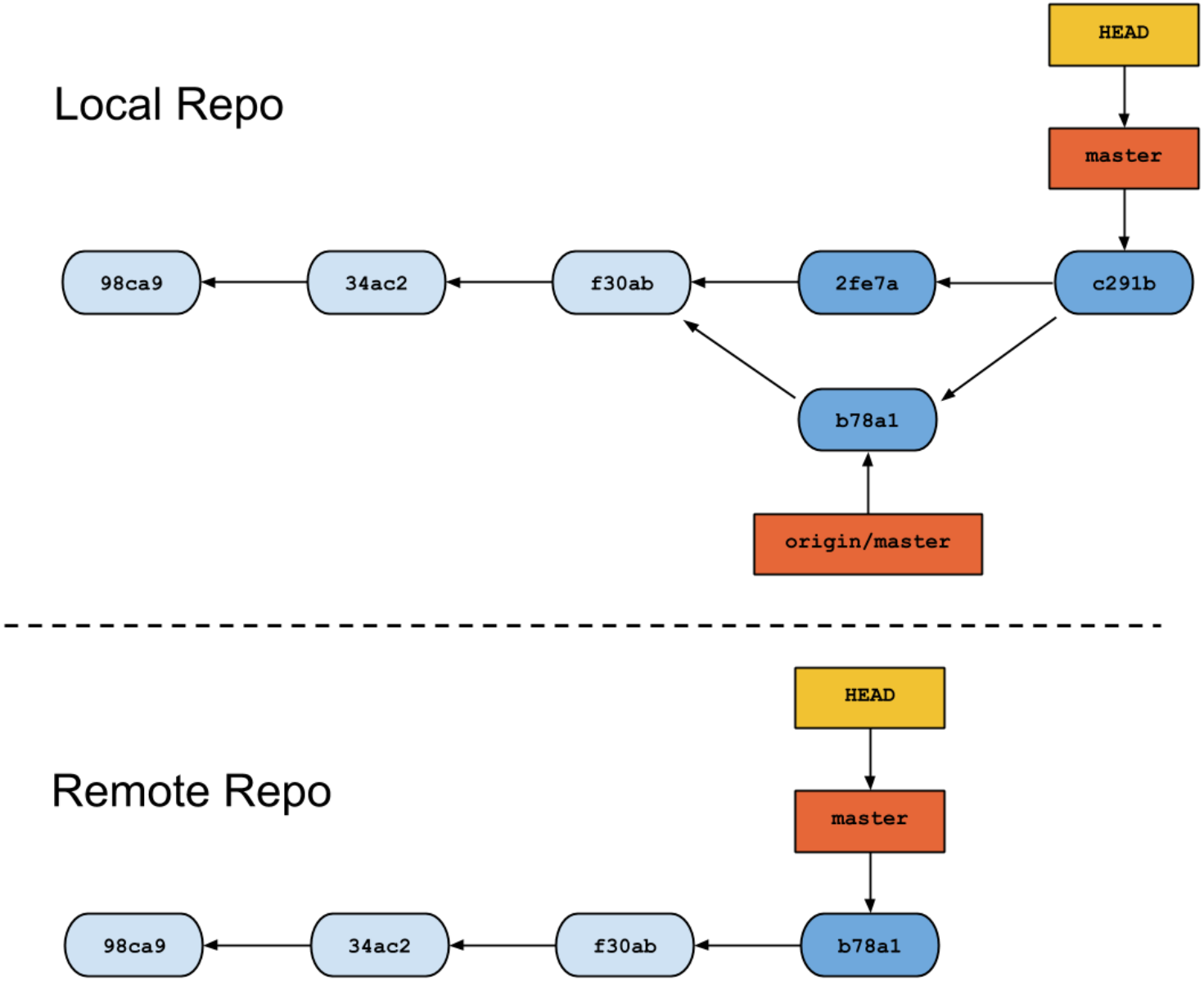
Remote Repo



git pull – first performs git fetch



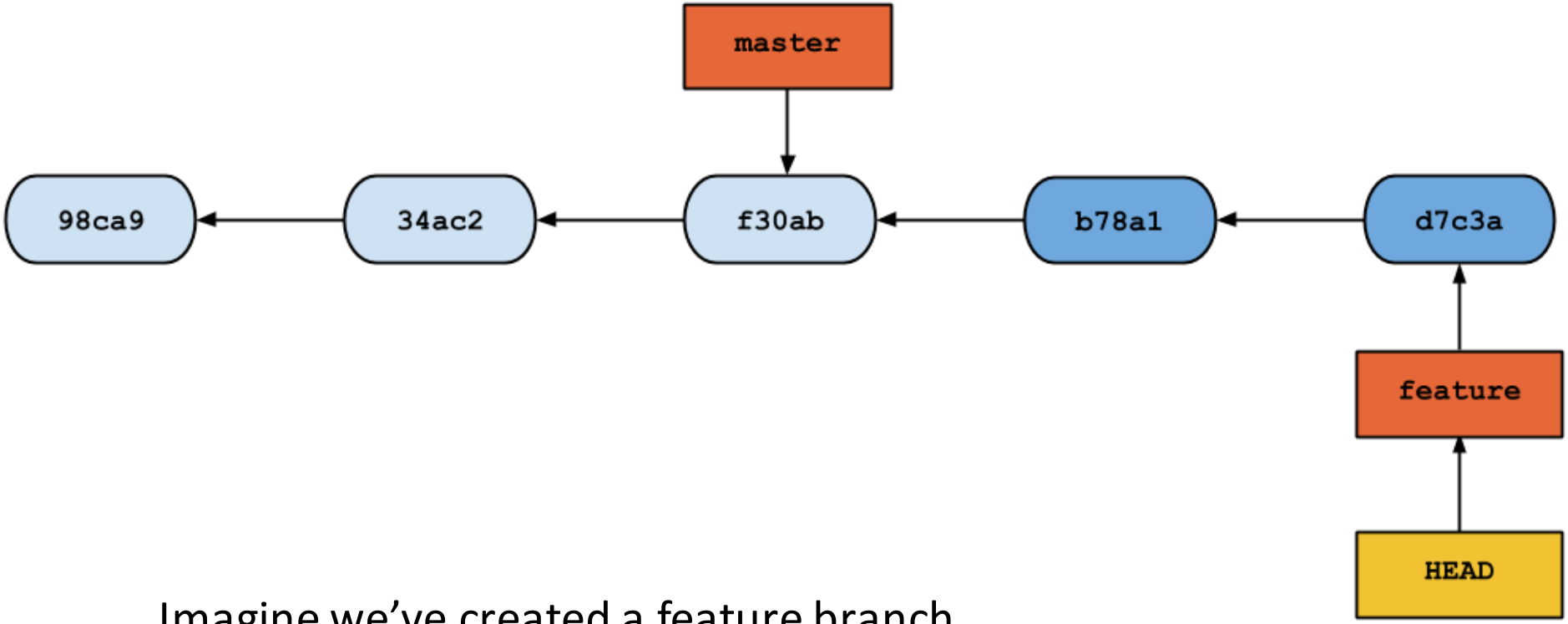
git pull – then performs git merge



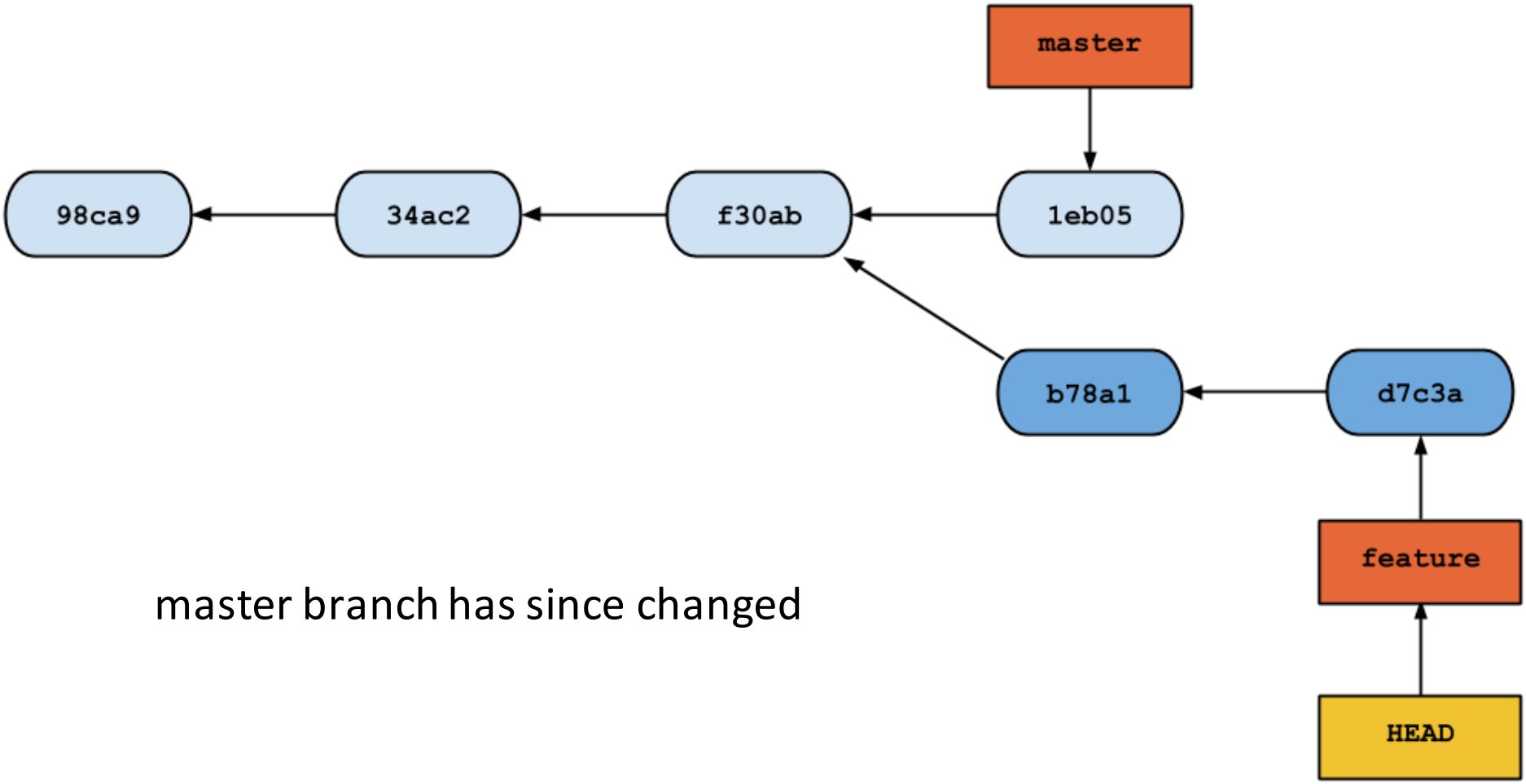
git rebase

Rebasing

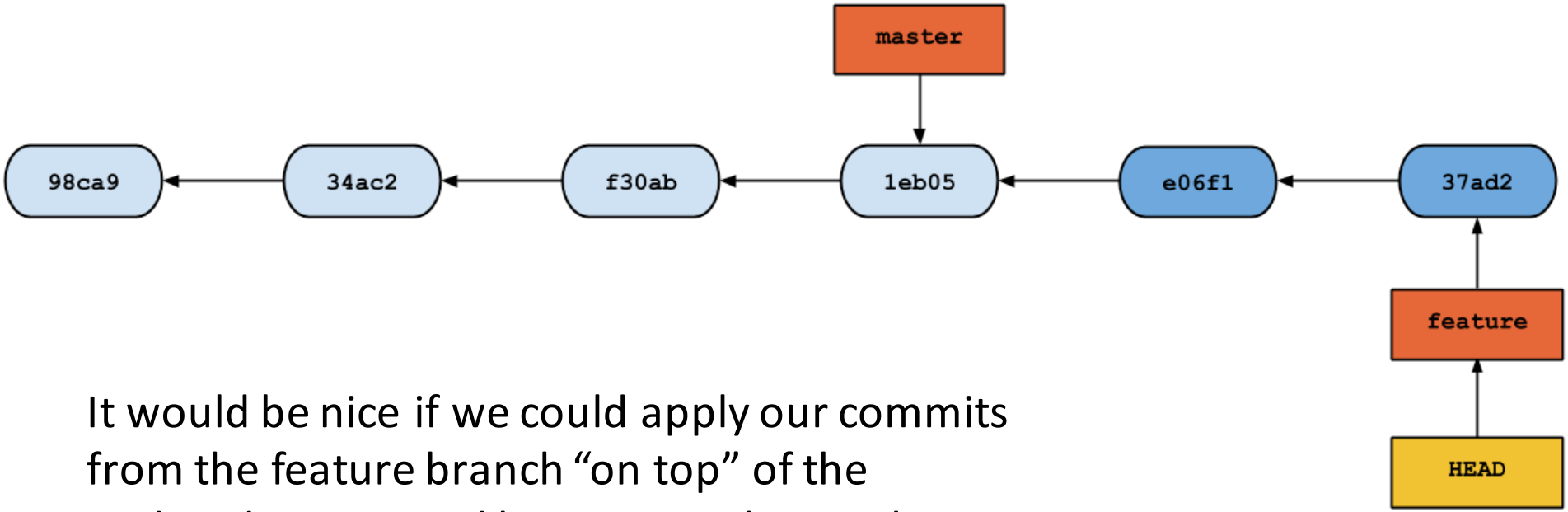
- **Rebasing:** Change the base of a branch by replaying a series of commits on top of different commit
- Rebasing keeps your commit history linear and easy to follow
- Rebasing also allows you to edit your commit history, such as squashing together multiple commits into one commit.
- Rebasing will change the SHAs of the commits you replay



Imagine we've created a feature branch off master and made some changes

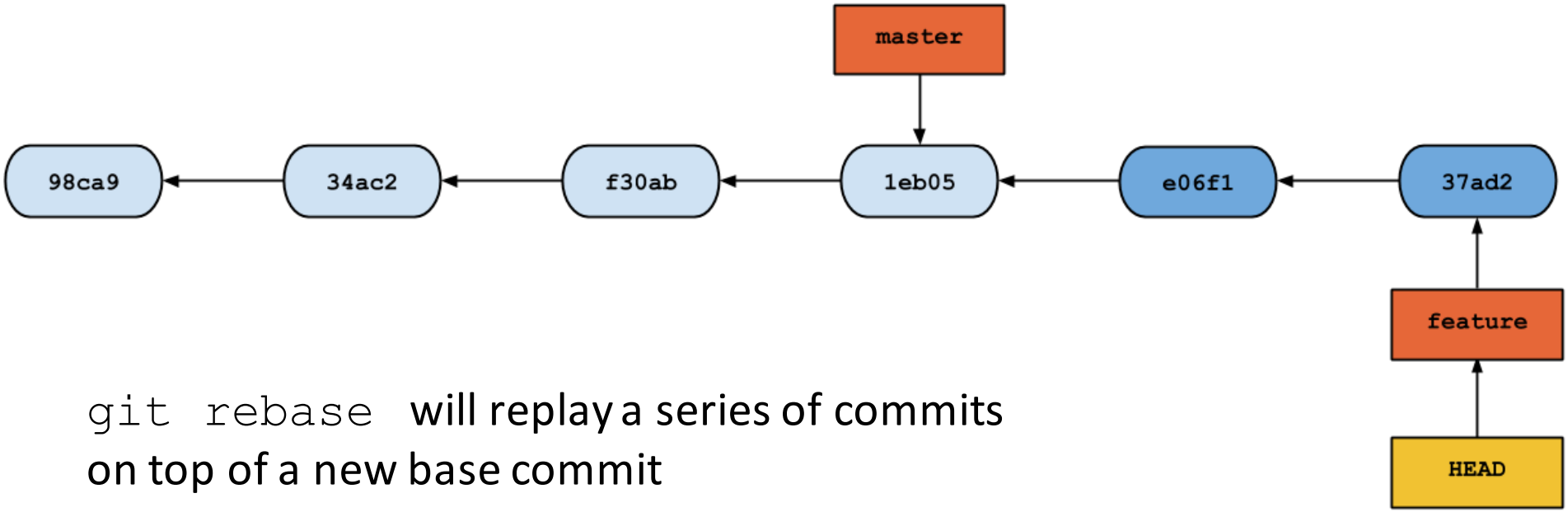


master branch has since changed



It would be nice if we could apply our commits from the feature branch “on top” of the updated master and keep our git history linear

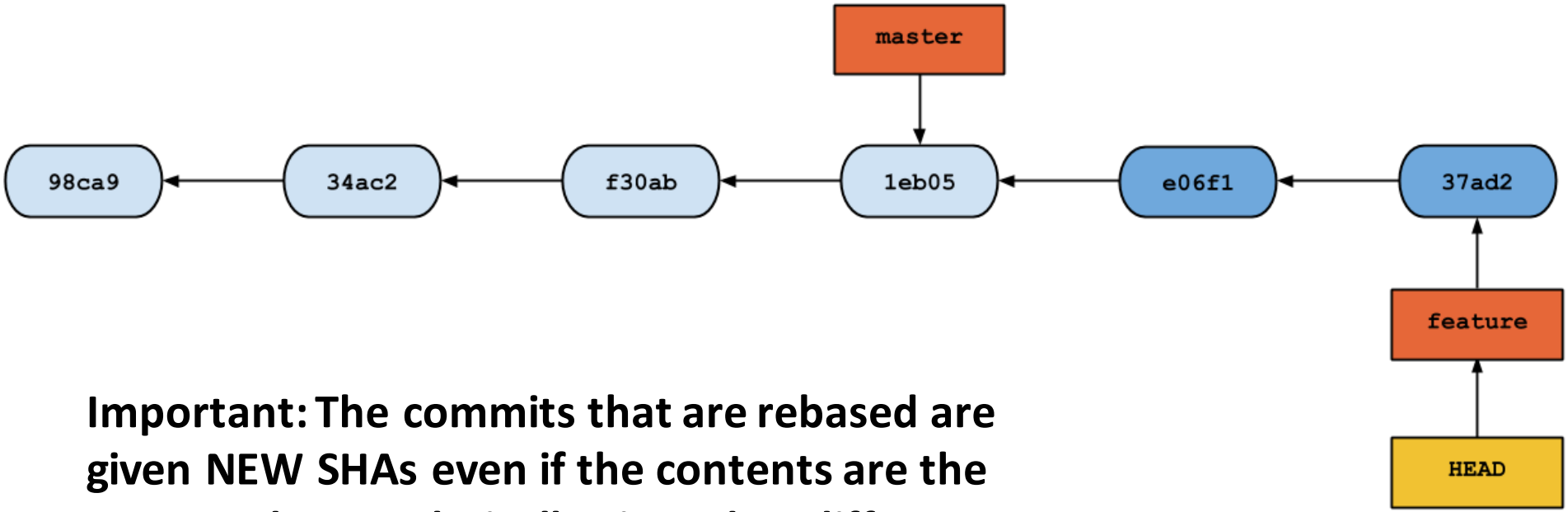
git rebase



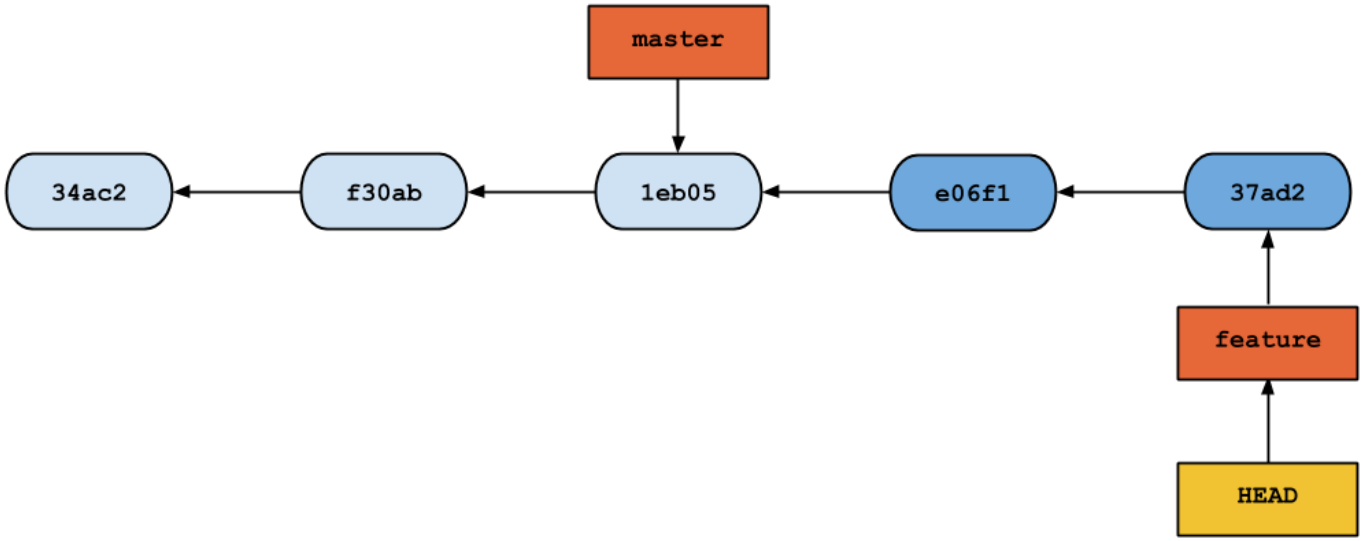
`git rebase` will replay a series of commits on top of a new base commit

`git rebase master` will replay commits from the current branch on top of the `master` branch

Rebasing

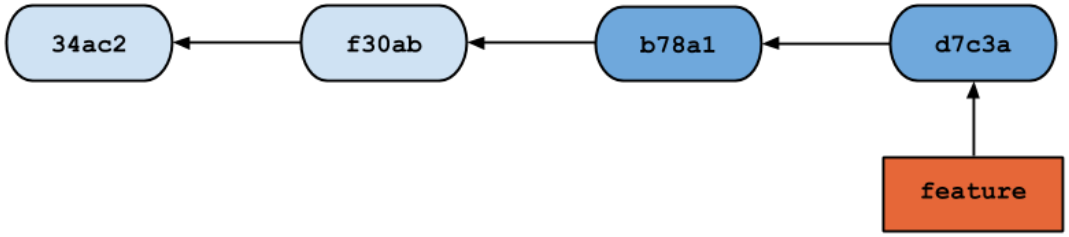


Important: The commits that are rebased are given NEW SHAs even if the contents are the same and are technically viewed as different commits by git



Local Repo

After rebasing



Remote Repo

Pushed before rebasing

Because the branches contain different SHAs as each other, `git push` will fail. git will suggest `git pull` but instead, you should ***force*** push with `git push -f`

Rebasing

!!! WARNINGS ABOUT REBASING !!!

- **REBASING RE-WRITES HISTORY SO BE INCREDIBLY CAREFUL HERE. IF YOU PERFORM A BAD REBASE, YOU MIGHT LOSE YOUR PREVIOUS COMMITS.**

- Rebasing locally requires force pushing to your remote branch, so **ONLY REBASE BRANCHES YOU OWN. NEVER REBASE MASTER AND NEVER FORCE PUSH TO MASTER.** If you are working on a branch with others, you may want to avoid rebasing (or be very careful)