Using GIT

by Stefan KLIKOVITS
based on “Using GIT” by Edmundo LOPEZ BOBEDA

UNIVERSITÉ
DE GENÈVE

Roadmap

- Git's nature
- The solitary workflow
- Working as a team

Git's nature

- DRCS (Distributed RCS) or DVCS (Decentralized VCS)
- Peer-to-peer model, NO CENTRAL SERVER

SVN vs. GIT

- Centralized
- Per folder permissions supported
- User management using Unix FS
- Slower (because of network)

- Decentralized
- Per repo/branch permissions
- User management using other tools
- Very fast (because local)
- Each client is a backup

Git configuration

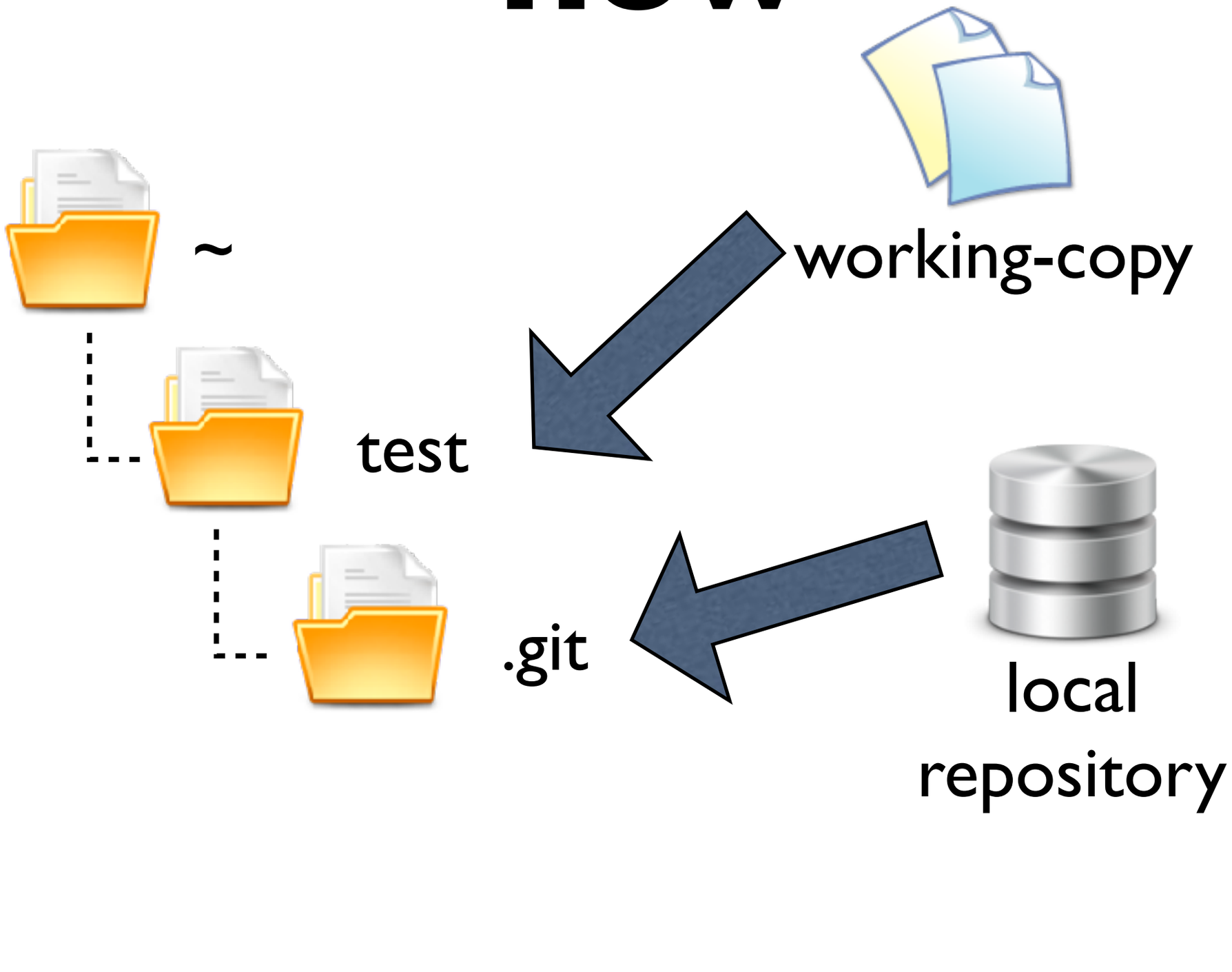
- No central user name data base, so you need to say that locally...

```
~ $ git config --global
user.name "First Last"
~ $ git config --global email
"email@email.com"
```

Create your repo

```
~/test $ git init
Initialized empty Git repository
in ~/test/.git/
```

What you have now



Create your repo

```
~ $ git clone test/ myClone
Cloning into myClone...
done.
```

Recording changes to the repo

Current situation:

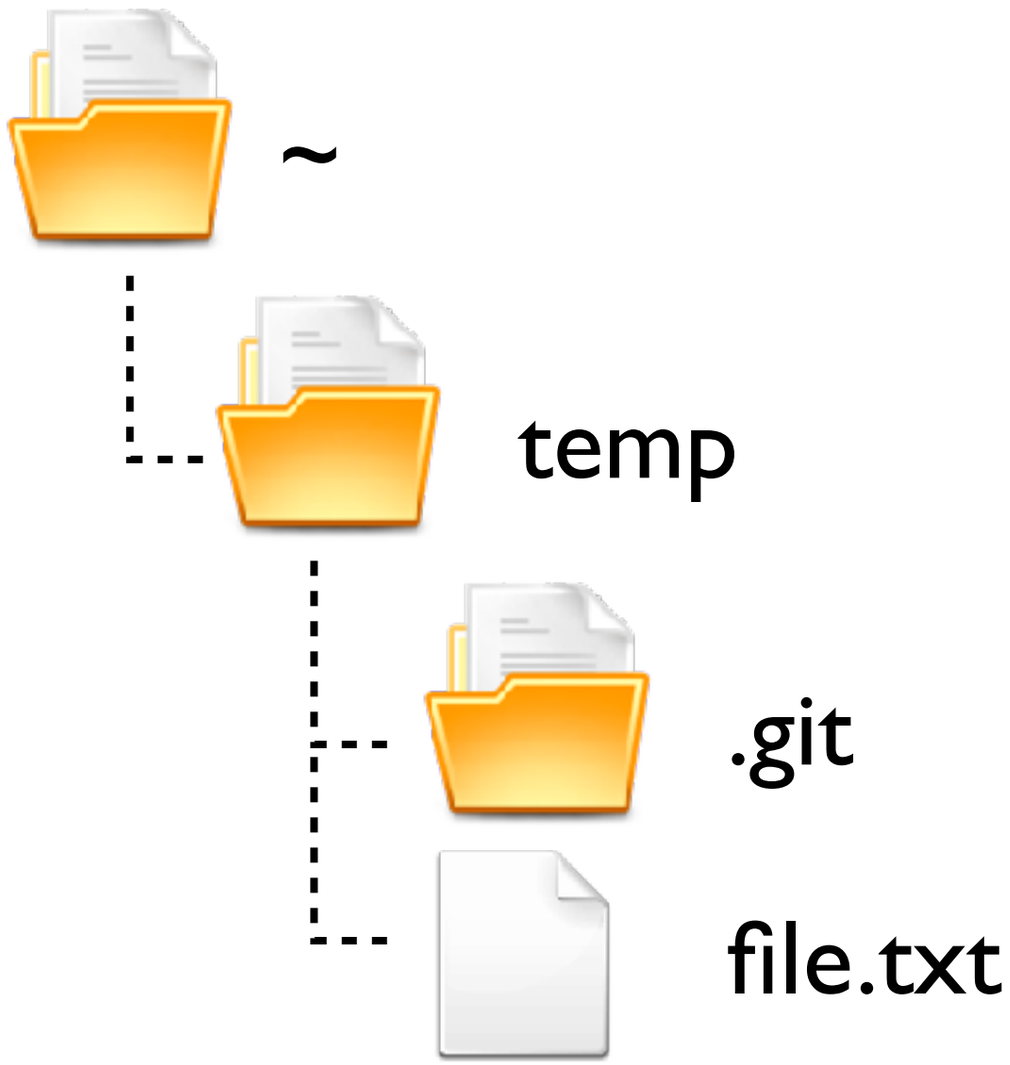
- You modified your working copy
- you want to record your changes

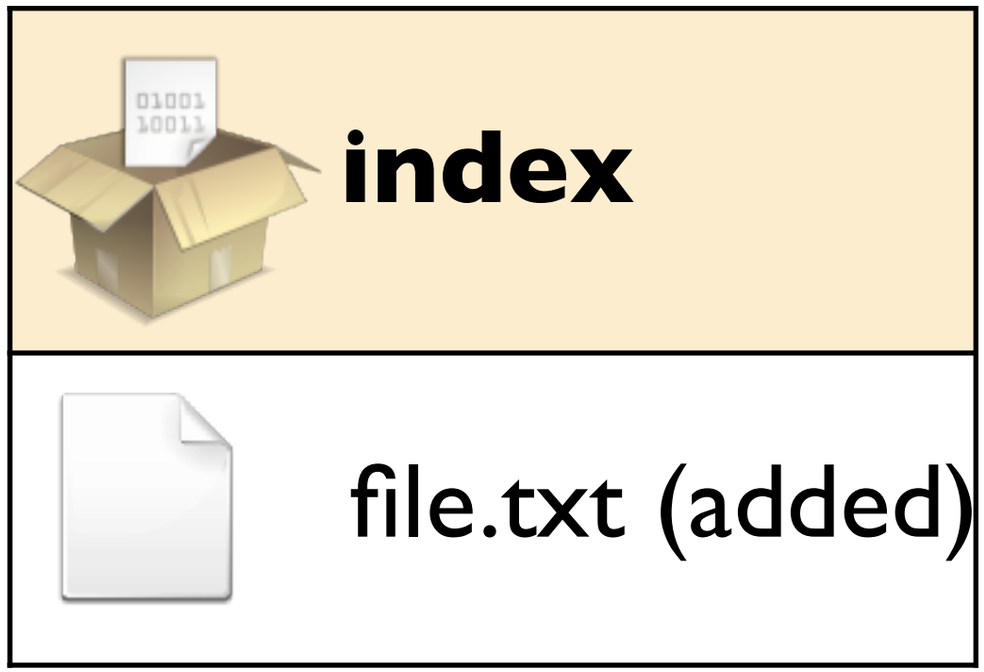
Recording changes to the repo

- Tracking files...

```
~/test $ git init
[[output...]]
~/test $ touch file.txt
~/test $ git add file.txt
```

After the add...





Recording changes to the repo

Commiting files...

```
~/test $ git commit -m
"Commited!"
[output...]
```

Recording changes to the repo

Commiting files without passing through the index...

```
~/test $ git commit -a -m
"Commited!"
[output...]
```

Looking at history

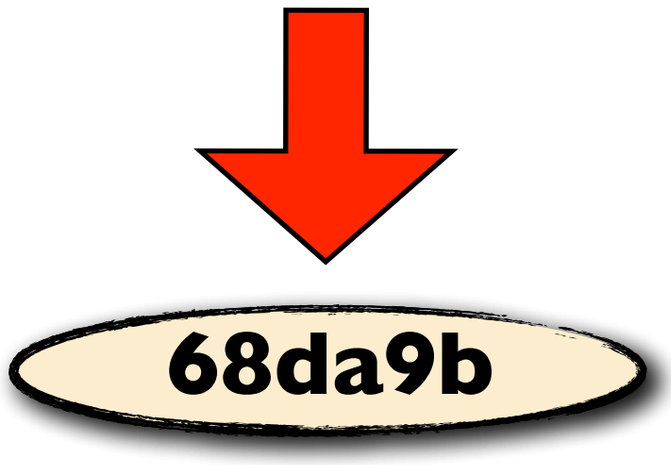
```
~/test $ git log
[output...]
```

Some theory

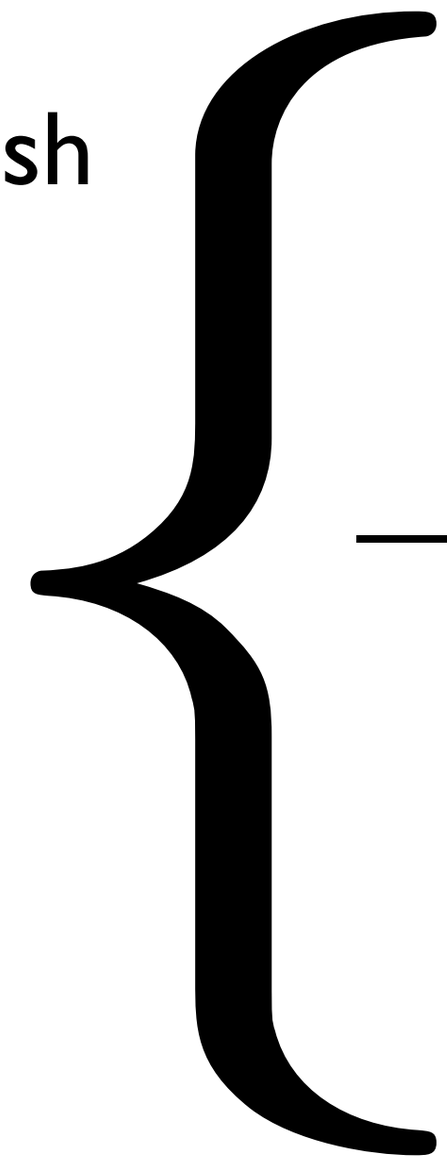
Commit ≠ Δ

Commits

partial SHA = treeish



68da9b



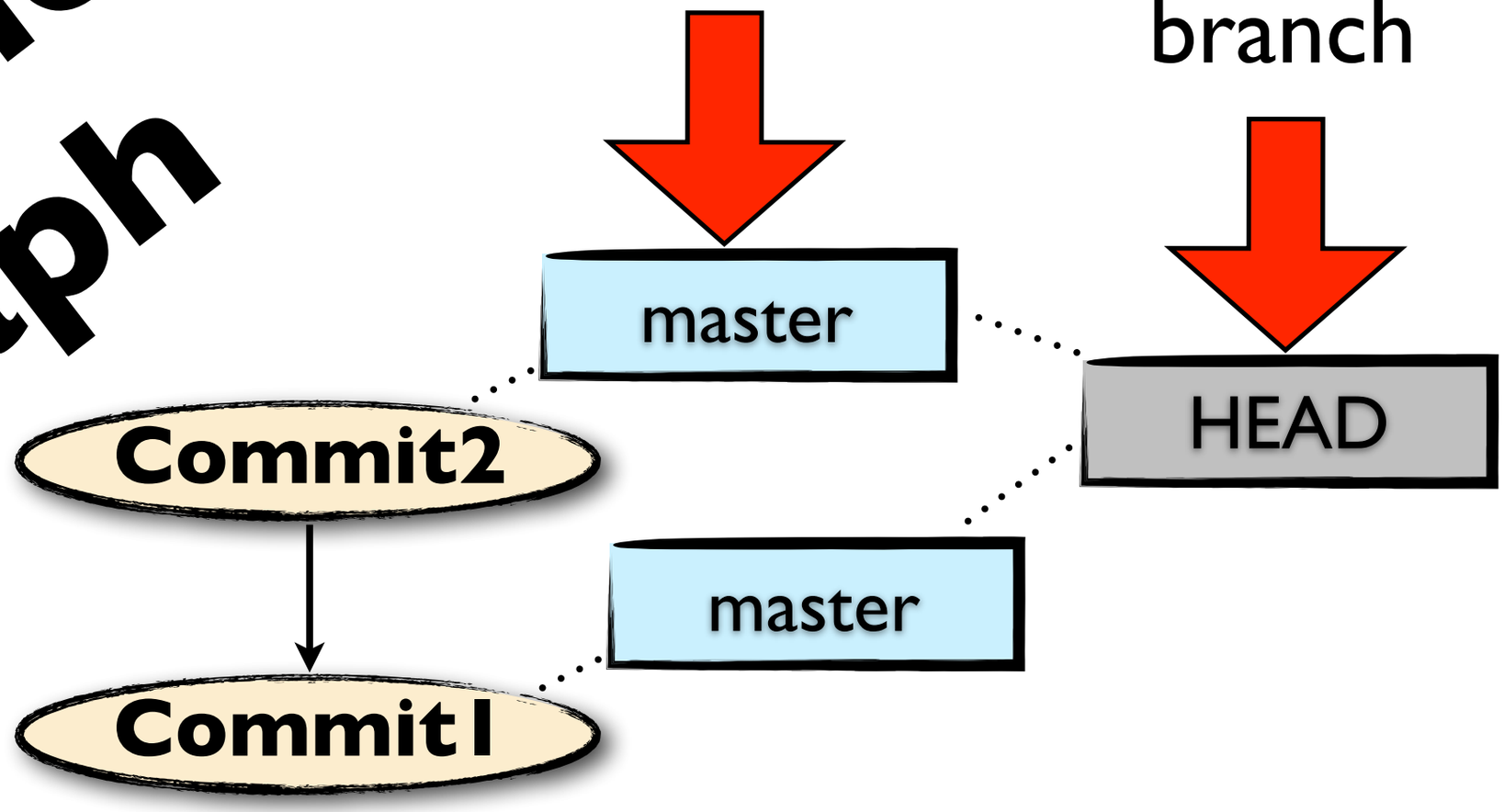
Metadata
68da9bfds456fe
Commiter
etc...

temp

file.txt

History

Directed Acyclic Graph



Summary

- A «commit» contains the whole file structure + metadata (committer, msg, etc)
- «Commits» are referred by SHA-1
- History = Directed Acyclic Graph of Commits
- Commits can have one or more parents

Undoing things (locally)

The git's way:

```
~/test $ git checkout PATH
```

Undoing things (locally)

WARNING

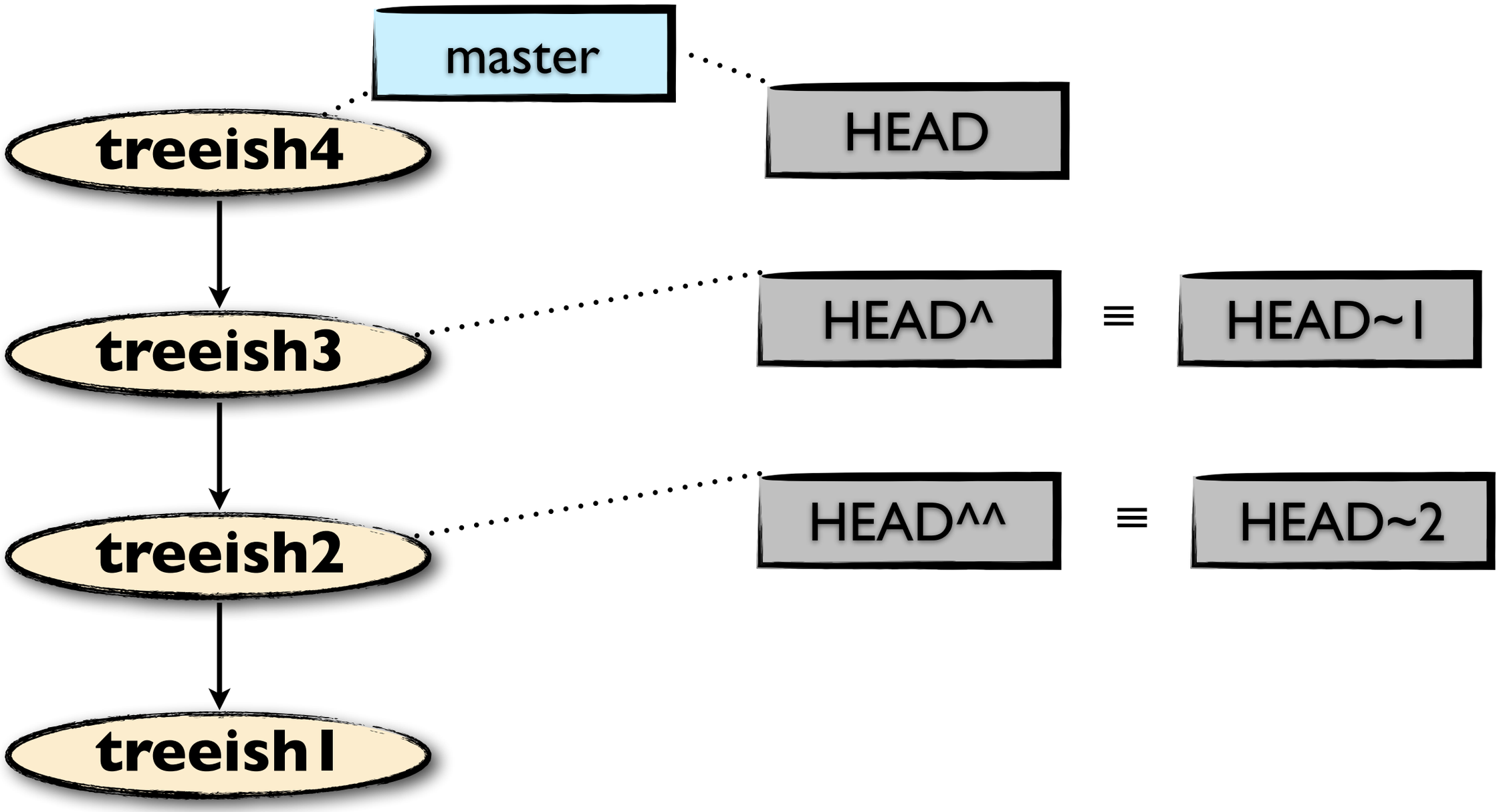
DO NOT amend or erase commits that you already pushed to the server

Undoing things (locally)

I forgot to commit something!!

```
~/test $ git commit --amend
```

Caret notation

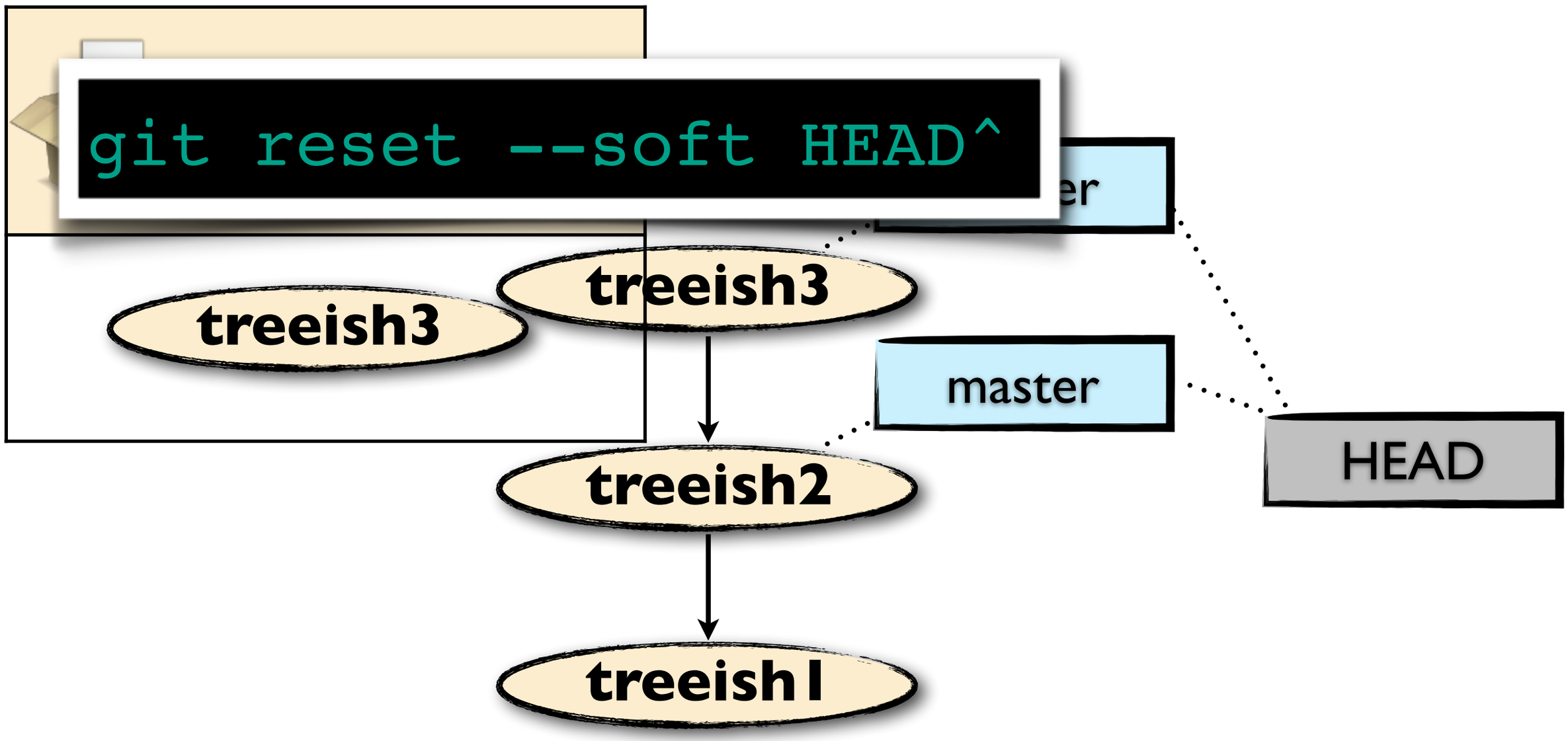


Undoing things (locally)

I want to go completely undo my last commit

```
~/test $ git reset --soft HEAD^
```

Undoing things (locally)



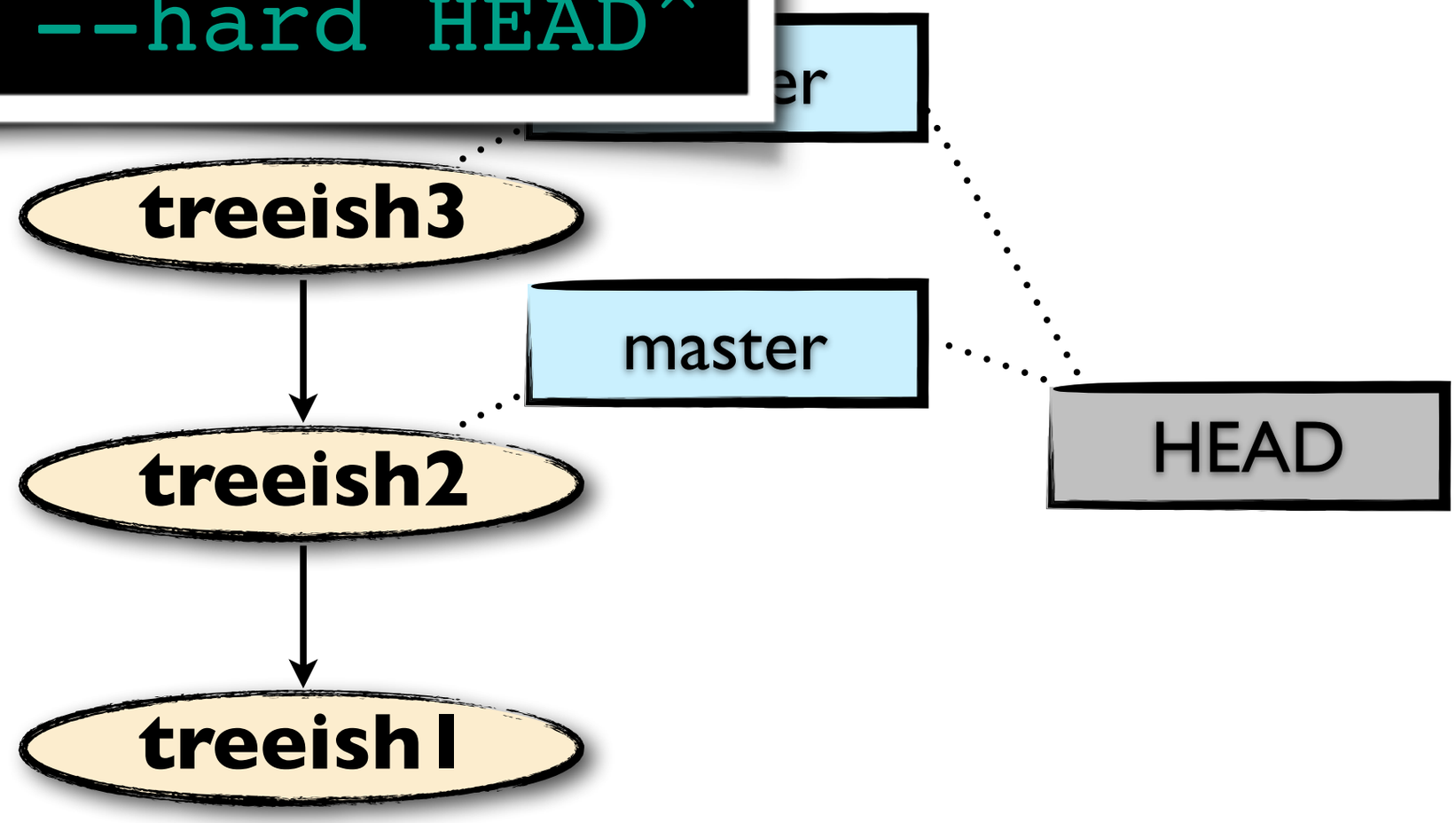
Undoing things (locally)

I want to go COMPLETELY undo my last commit

```
~/test $ git reset --hard HEAD^
```

Undoing things (locally)

```
git reset --hard HEAD^
```



Undoing things (locally)

WARNING

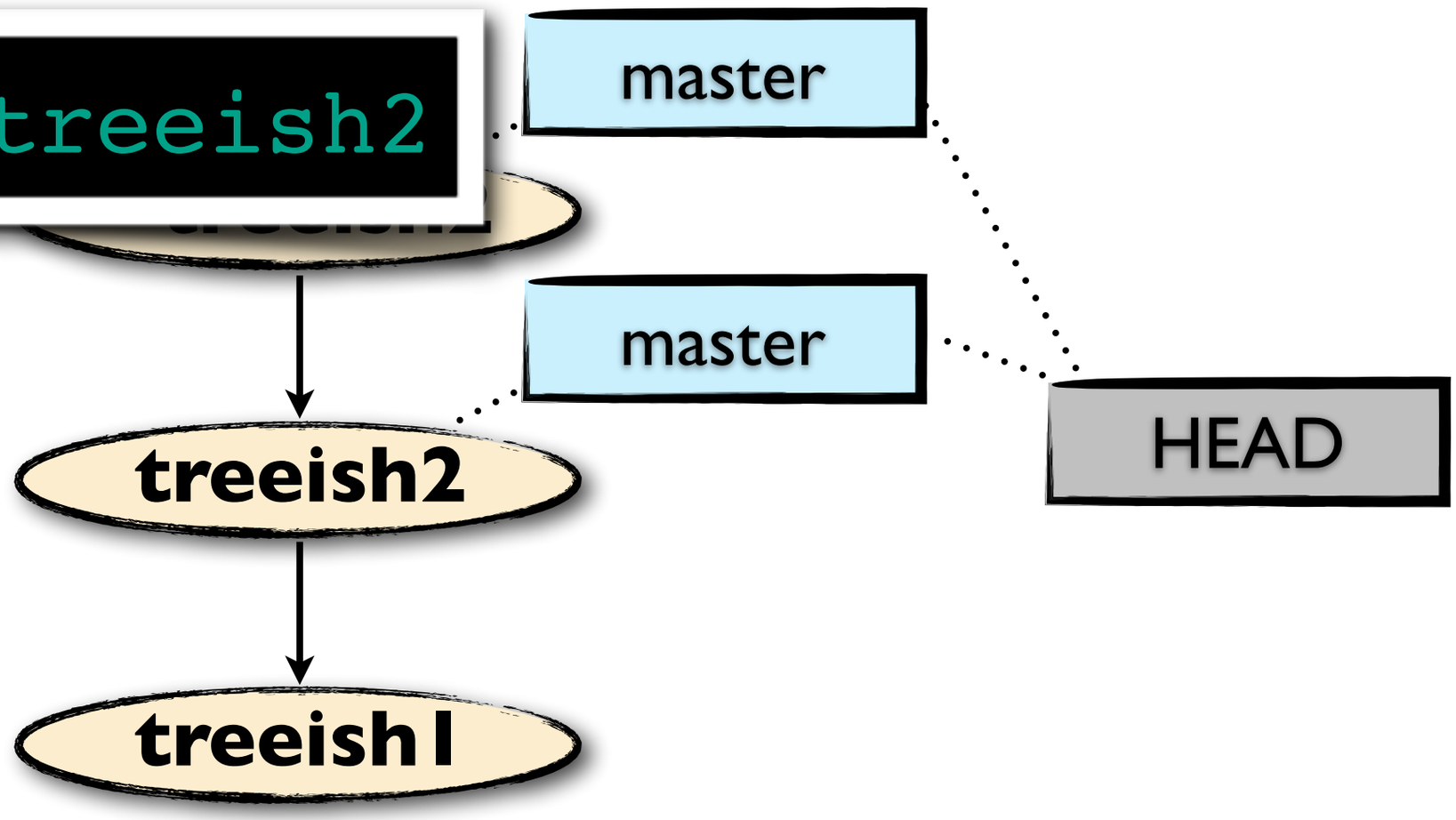
`git reset` will actually erase the history and «reset» the tip of the branch to the given commit

Reverting your changes

```
~/test $ git revert <treeish>
[output...]
```

Git's Revert

```
git revert treeish2
```



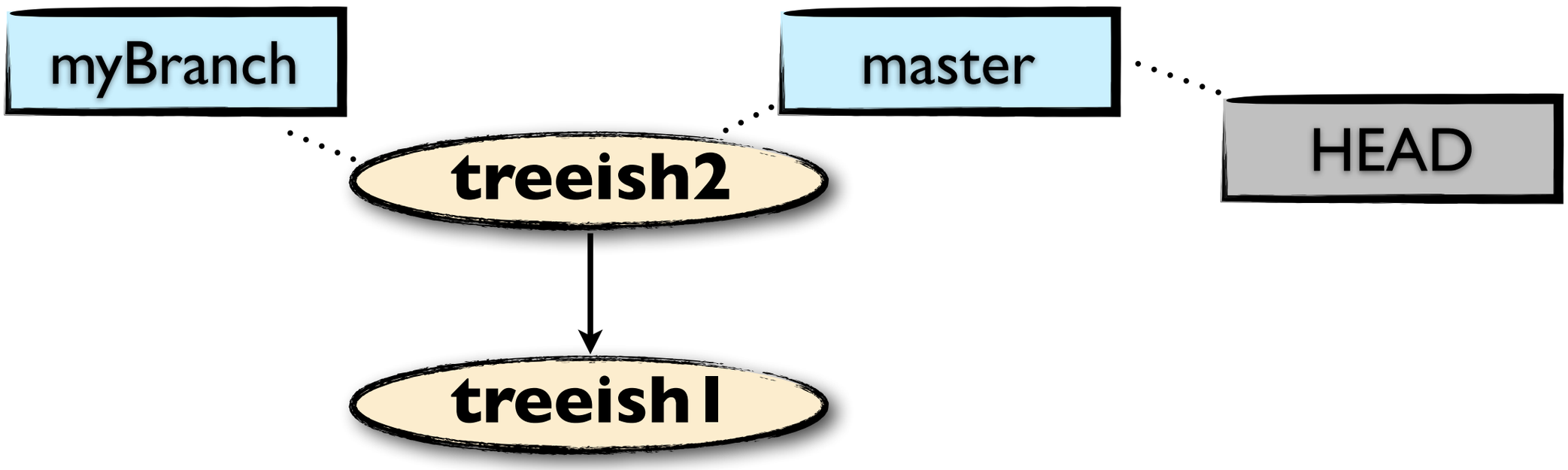
Branching

Create a branch:

```
~/test $ git branch <branchname>
```

Branching

```
git branch myBranch
```

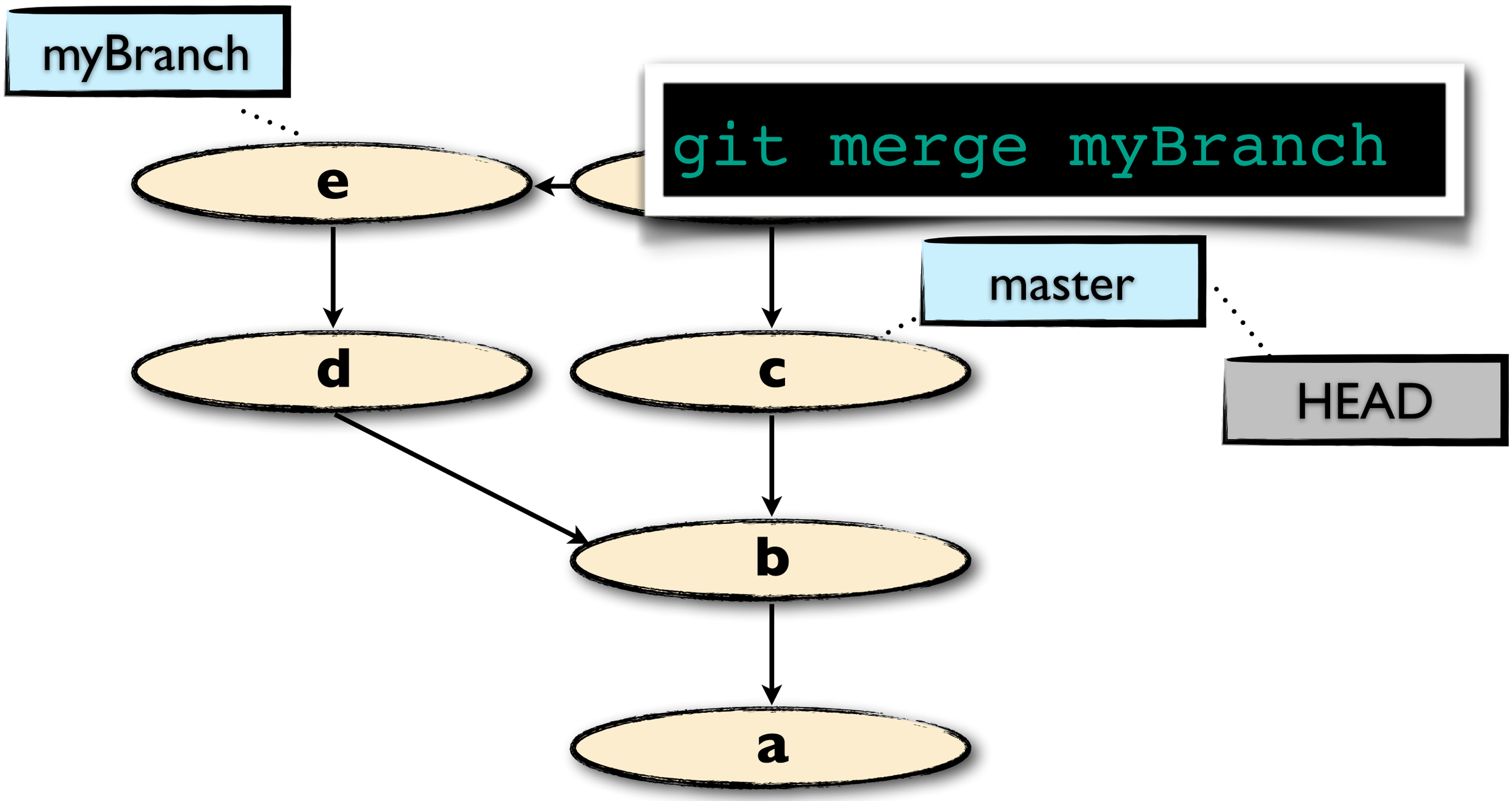


Merging

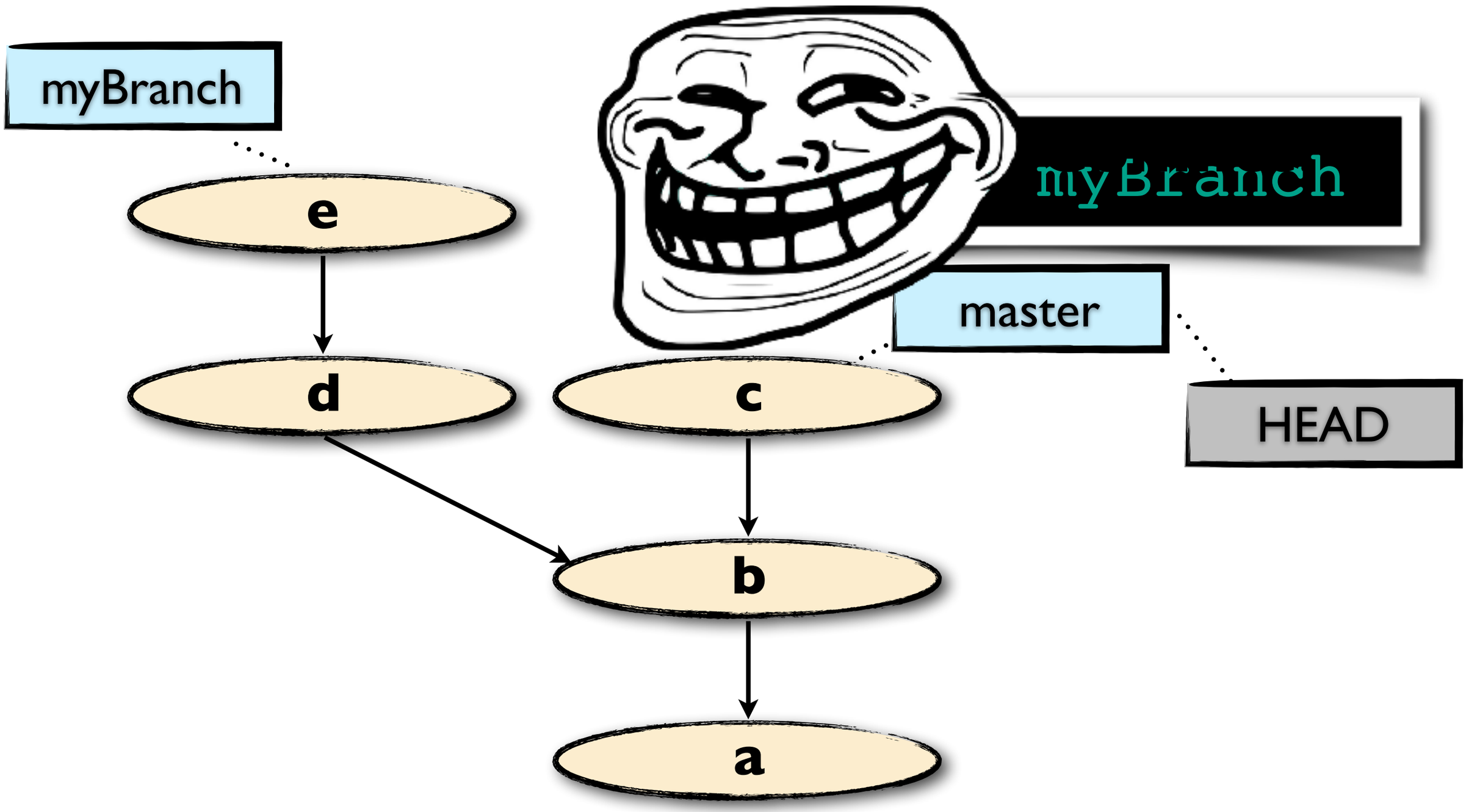
Merging into a branch A into branch B:

```
~/test $ git checkout B
Switched to branch ‘B’
~/test $ git merge A
[output...]
```

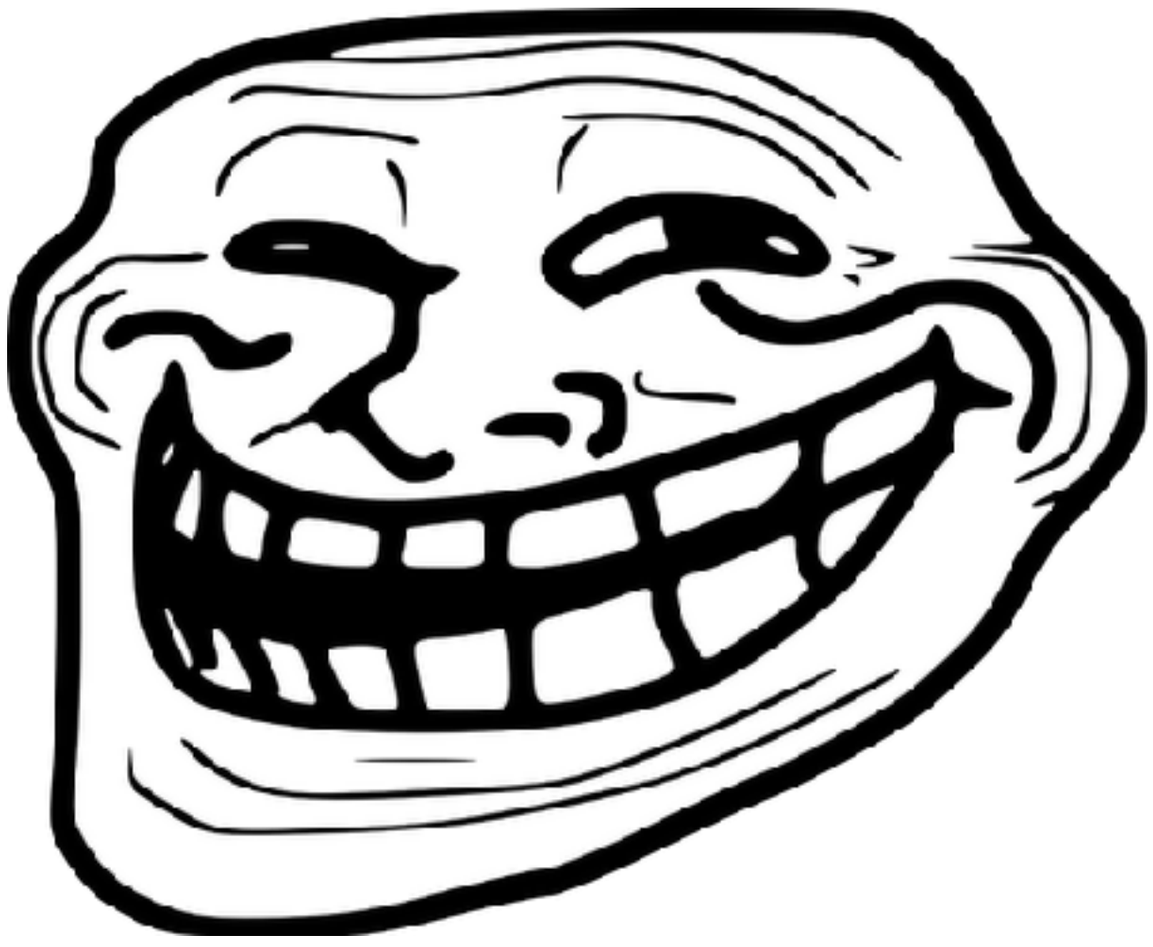
Merging



Merging



Dealing with conflicts



Dealing with conflicts

- Conflicted files stay in the index, you can get them using
  `git checkout [--ours | --theirs] <filename>`
- Once the commit is solved use `git add <filename>` to «mark as merged»
- Finally commit as normal

Diffs

Compare to arbitrary commits

```
~/test $ git diff <treeish1>
<treeish2> PATH
```

Diffs

Will use the default system tool for «diffing»

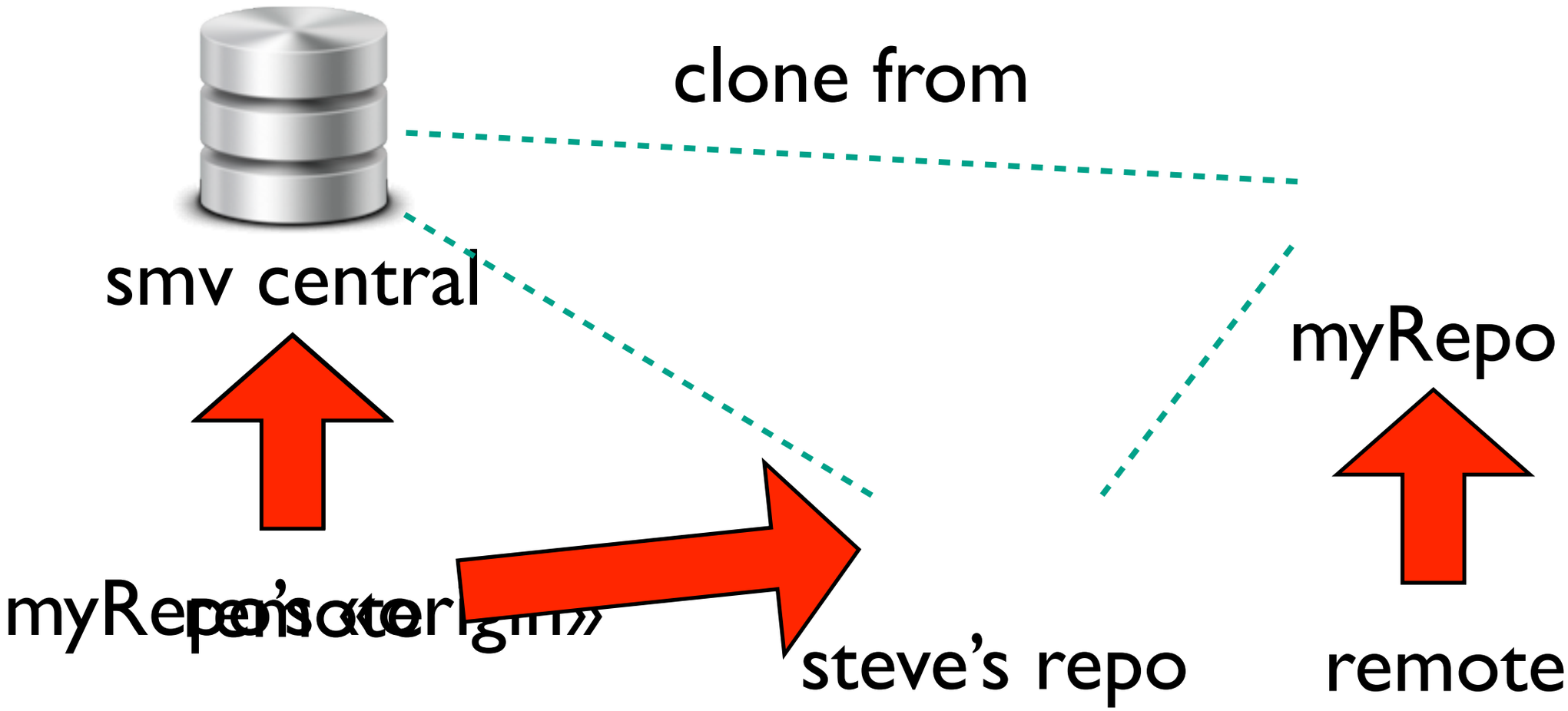
```
~/test $ git difftool <treeish1>
<treeish2> PATH
```

Workaround...

Get an arbitrary file from an arbitrary revision

```
~/test $ git show treeish:PATH >
arbitraryName
```

Peer to peer mode

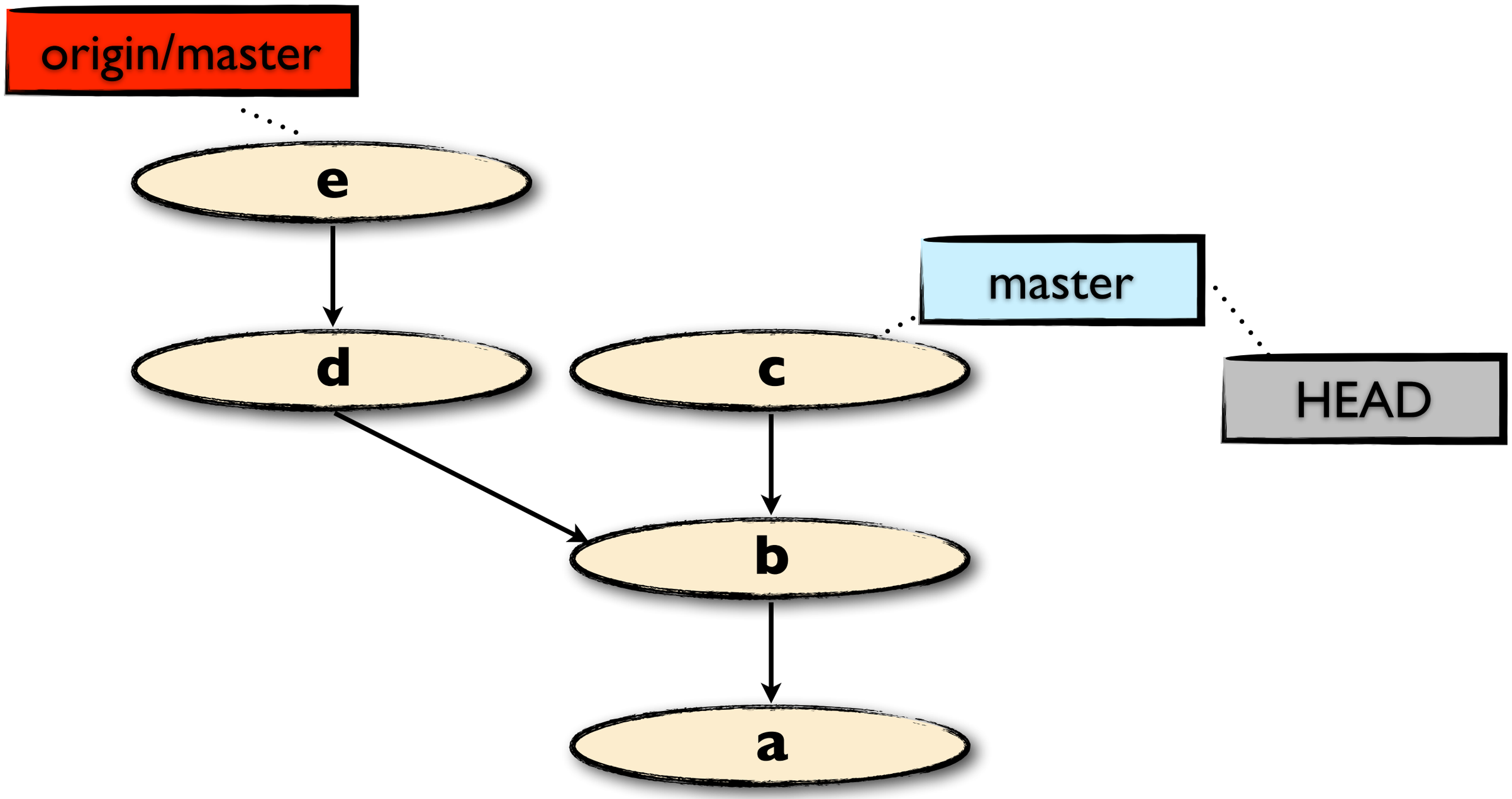


Remotes

List and manage remotes... (see man page)

```
~/test $ git remote
```

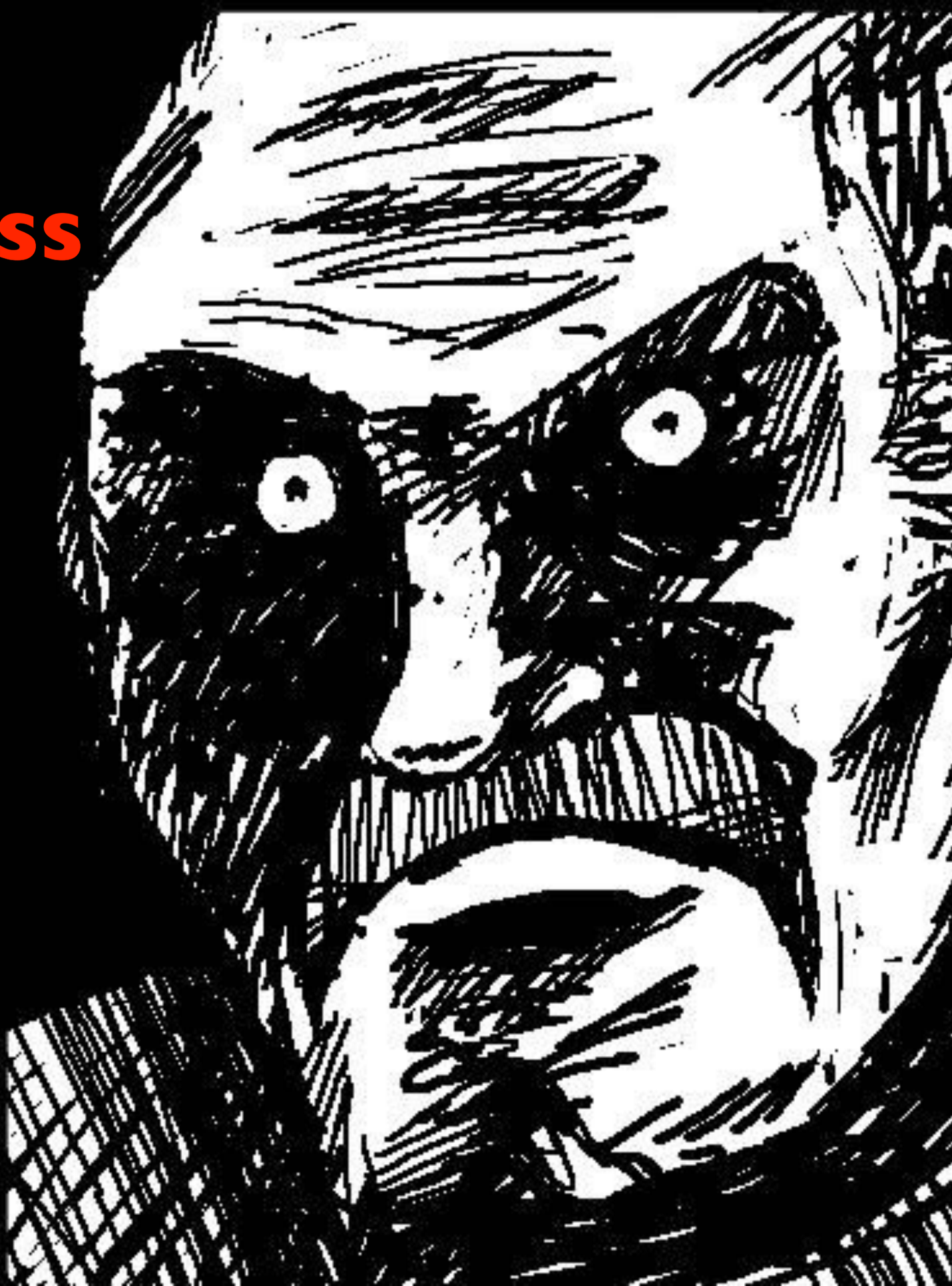
Remote branches



You NEVER mess

with remote

branches



Remote branches

Remember

Remote branches are to be manipulated uniquely by the remote. You cannot directly commit to them or even switch to them

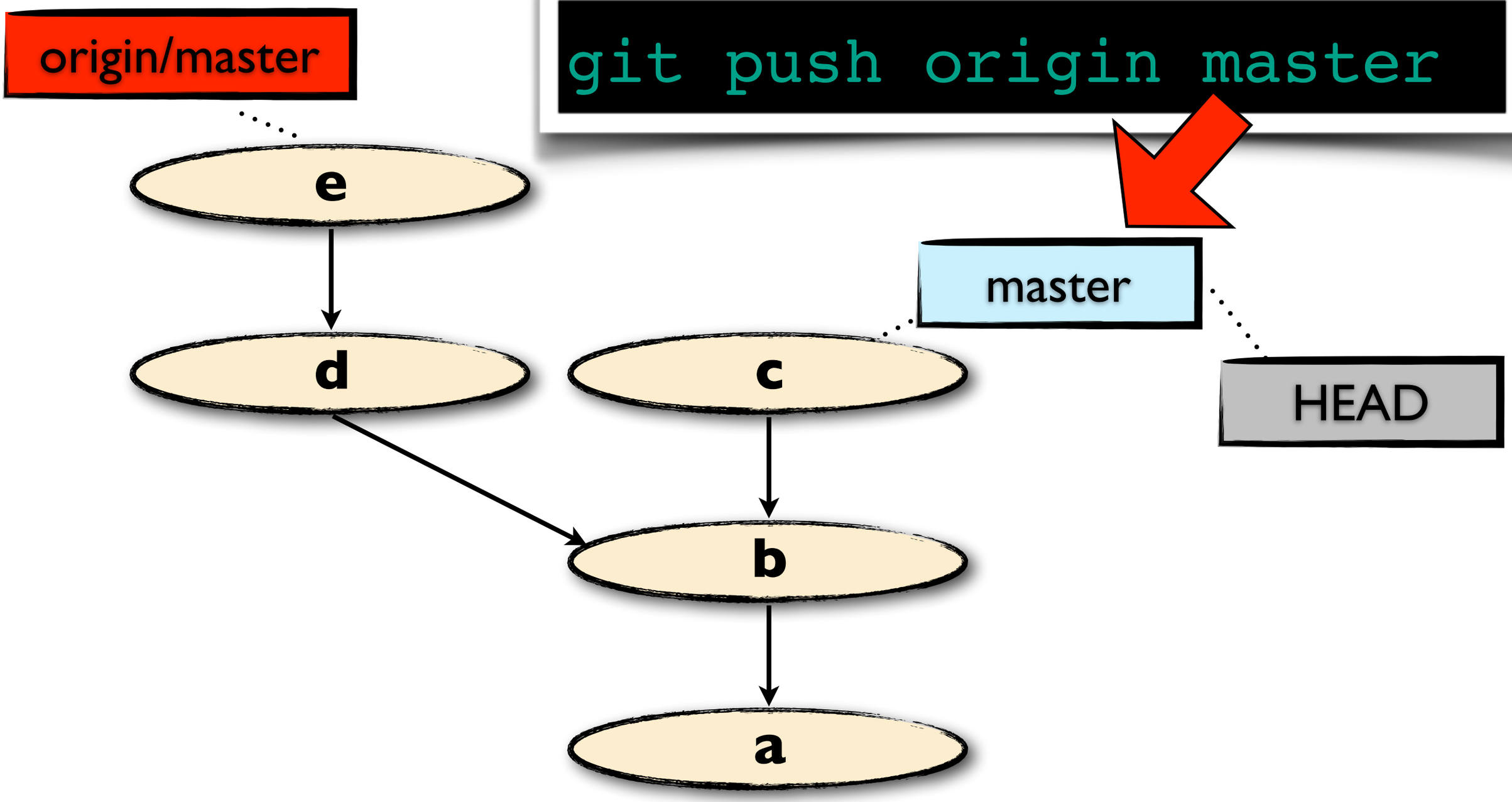
git push

From git's manual page:

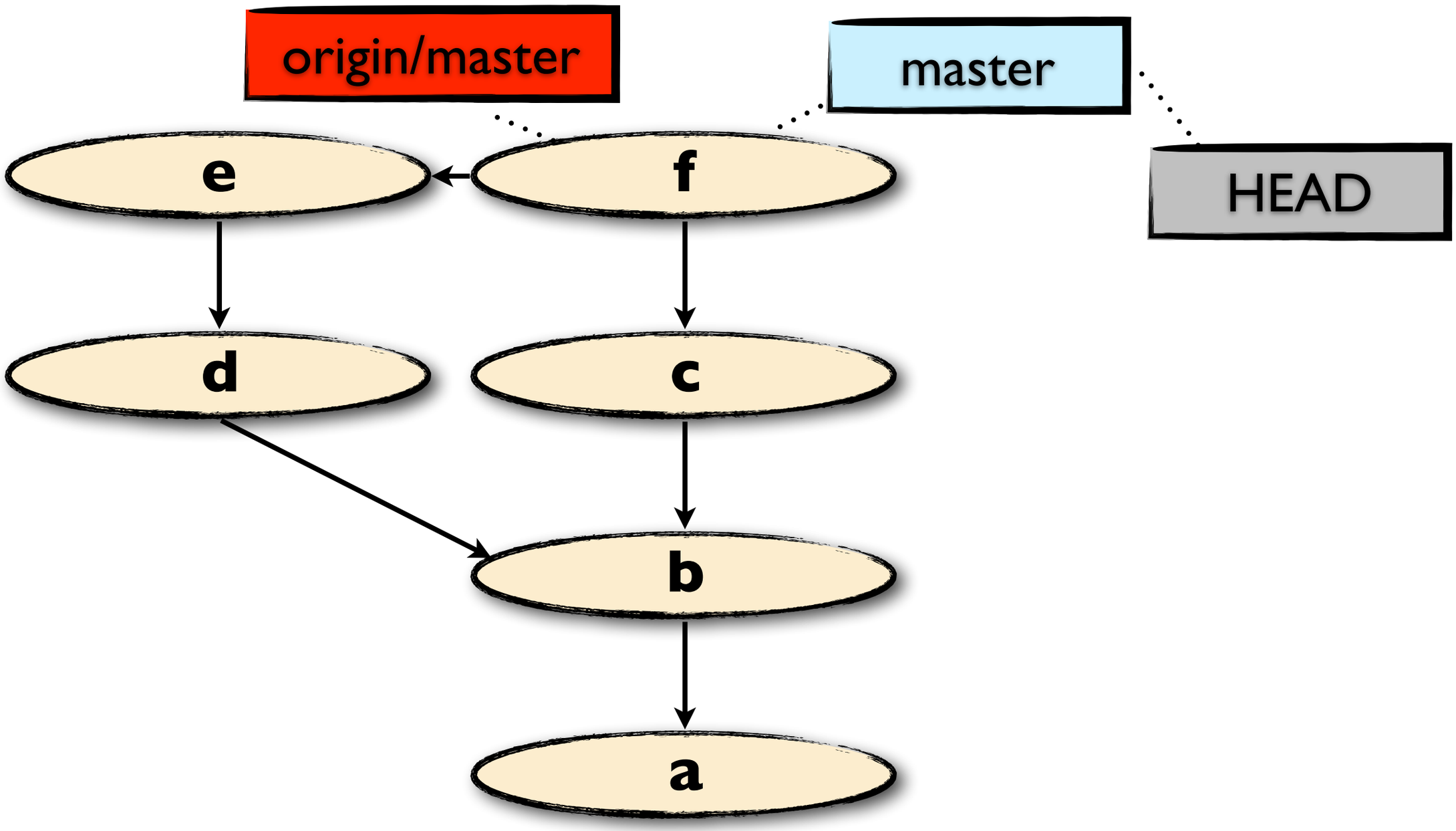
`git push <remote> <refspec>` .-
Updates remote refs using local refs, while sending objects necessary to complete the given refs.

git push

```
git push origin master
```



git push



git fetch

From git's manual page:

> `git fetch` .- Fetches named heads or tags from one or more other repositories, along with the objects necessary to complete them.

git pull

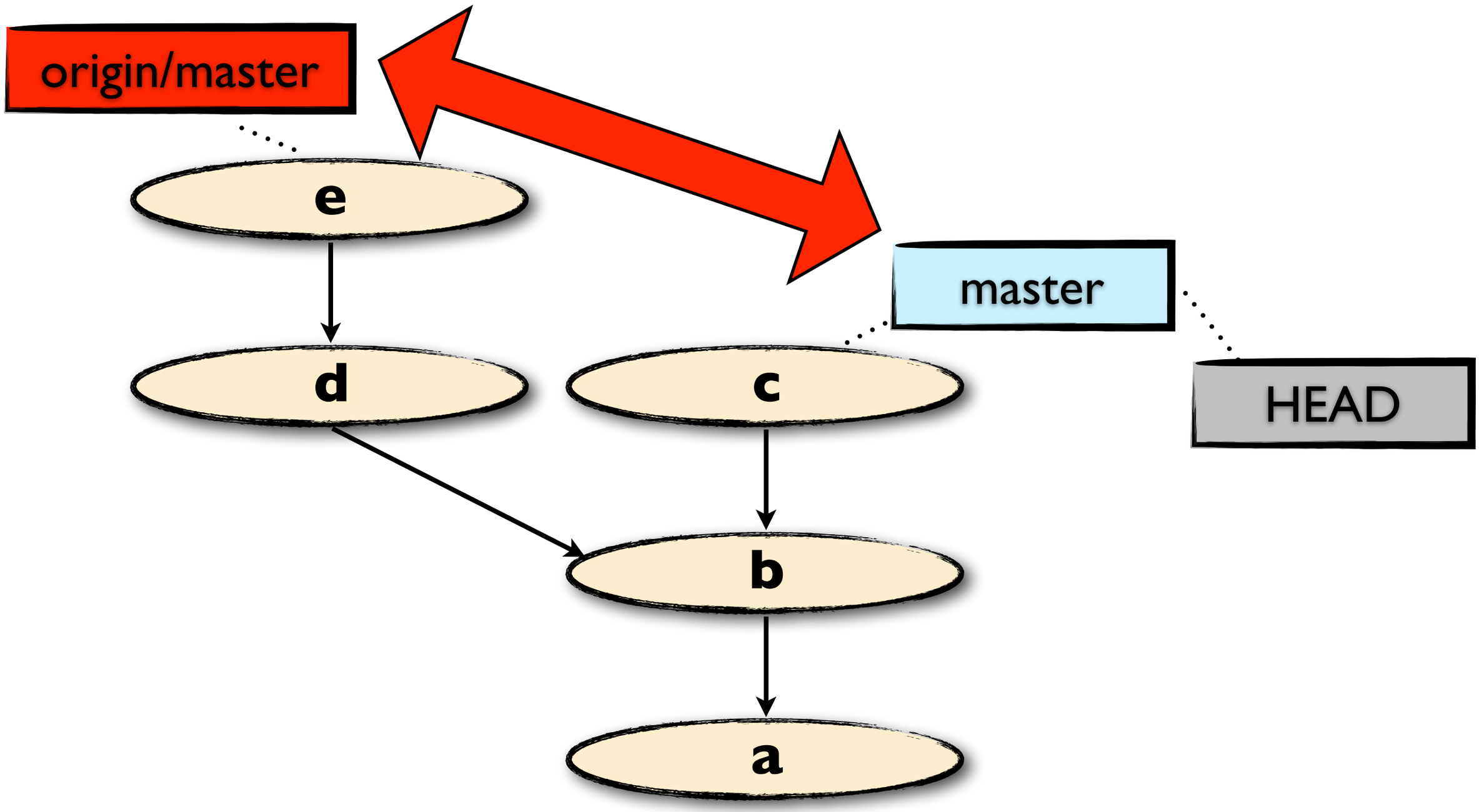
From git's manual page:

> `git pull` .- Incorporates changes from a remote repository into the current branch

git pull

fetch + merge

tracking branch



Desktop Clients

- https://git-scm.com/downloads/guis
- https://desktop.github.com/
- https://tortoisegit.org/
- http://www.syntevo.com/smartgit/

…

--questions?

Exercises, Reading, etc.

! Create Github account

! install git on your dev-machine

+ Interactive Git tutorial
http://learngitbranching.js.org/

+ Github Guide (Hello World, Github Flow)
https://guides.github.com

~ create projects with friends
init, clone, commit, push, pull, …

/ Vogella's Git Intro:
http://www.vogella.com/tutorials/Git/article.html