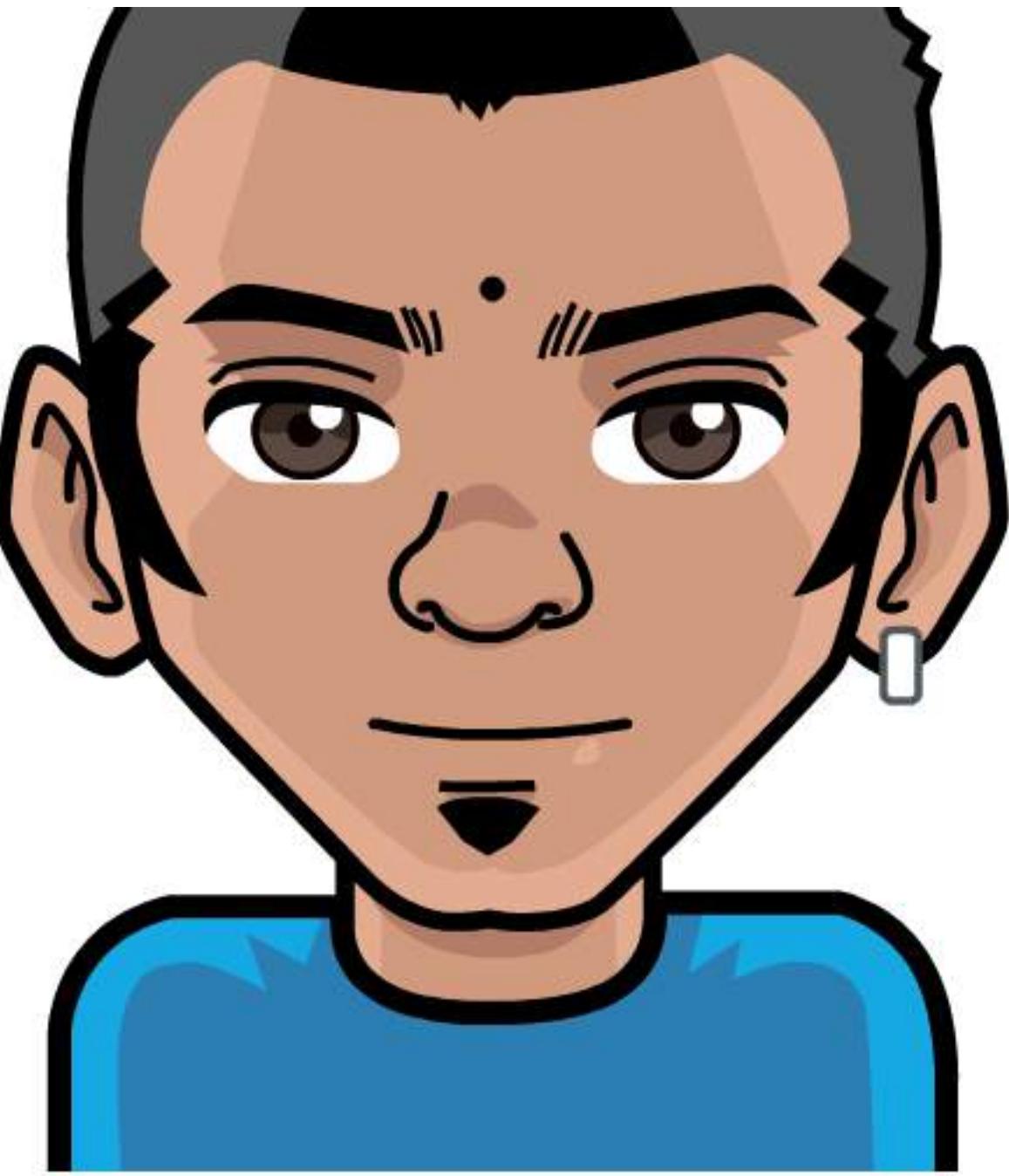


Git Next Steps



RAJU GANDHI
⌚ 🐦 🔳 @LOOSELYTYPED
FOUNDER - DEFMACRO SOFTWARE

Knowing
THE INTERNALS

" bad programmers worry about the code.
**GOOD PROGRAMMERS WORRY ABOUT
DATA STRUCTURES AND THEIR
RELATIONSHIPS**

- Linus Torvalds

config
description
HEAD
hooks
commit-msg.sample

...

info
exclude
objects
info
pack
refs
heads
tags

`config`
~~description~~

`HEAD`
`hooks`

`commit-msg.sample`

`...`

`info`

`exclude`

`objects`

`info`

`pack`

`refs`

`heads`

`tags`

Symbolic Reference

Object Database

References

**Blob
Tree
Commit
Tag**

**Blob
Tree
Commit
Tag**

INTEGRITY

Blobs / Trees / Commits

config
~~description~~
HEAD
hooks
commit-msg.sample

...

info
exclude
objects
info }
pack
refs
heads
tags

Object Database

SHA-1

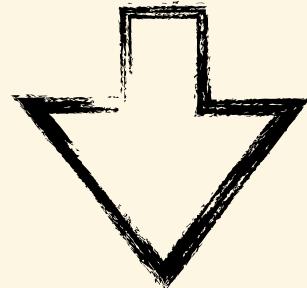
Blob

Store the contents

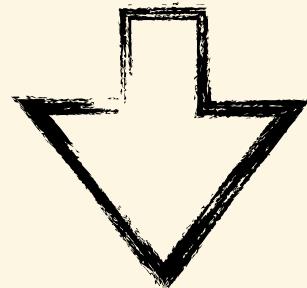
SHA-1

Blob

Hello World!



SHA-1 hash



980a0d5f19a64b4b30a87d4206aade58726b60e3

SHA-1

Blob

980a0d5f19a64b4b30a87d4206aade58726b60e3

```
.git/objects
  └── 98
      └── 0a0d5f19a64b4b30a87d4206aade58726b60e3
  └── info
  └── pack
```

SHA-1

Blob

No metadata

SHA-1

Blob

**Content addressable
filesystem**

Quiz Time!

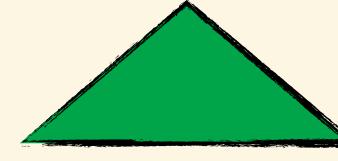
If you create two files with the **same content** in the same repository

- You get only one blob because the contents are the same
- You get two separate blobs because we have two files
- None of the above — blobs don't store file contents

Quiz Time!

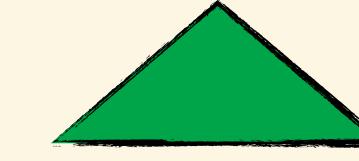
If you **add** a file with some content, then **modify** the content in that file, and **add it again**, how many blobs would you have?

- You get only one blob Git will delete the other blob
- You get two separate blobs because you add-ed two separate contents
- Git will use the previously created blob to store the new context



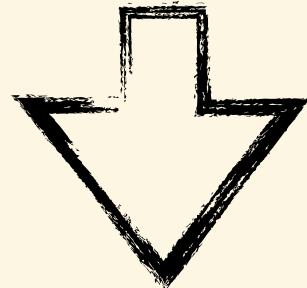
Tree

Stores structure and names

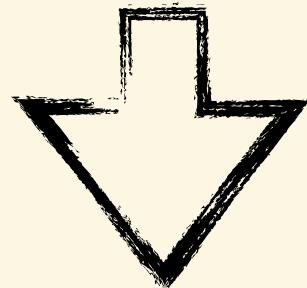


Tree

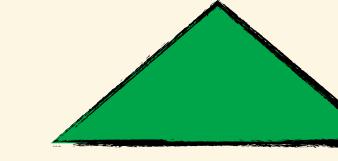
100644 blob 980a0d5f19a64b4b30a87d4206aade58726b60e3 README.md



SHA-1 hash



ad123a385f6c7d11ca711427e575d1530239caad



Tree

ad123a385f6c7d11ca711427e575d1530239caad

.git/objects

├── 98

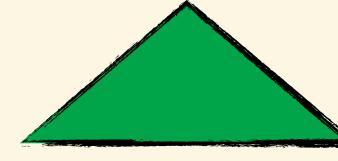
│ └── 0a0d5f19a64b4b30a87d4206aade58726b60e3

├── ad

│ └── 123a385f6c7d11ca711427e575d1530239caad

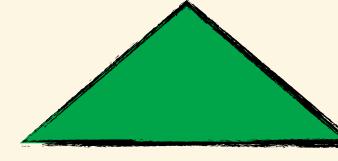
├── info

└── pack



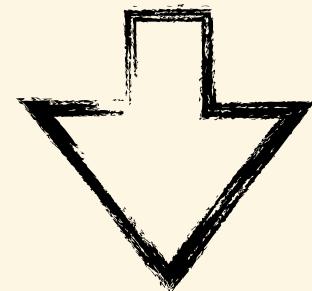
Tree

Nested trees

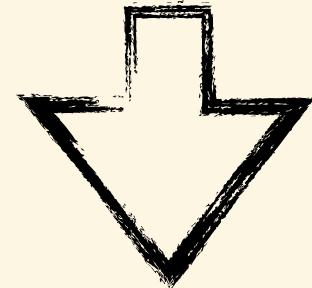


Tree

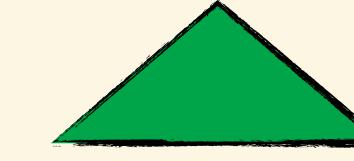
```
100644 blob 980a0d5f19a64b4b30a87d4206aade58726b60e3 README.md  
040000 tree ad123a385f6c7d11ca711427e575d1530239caad lib
```



SHA-1 hash



e6dab7bf1353cc6abf8f0f79811f26ff7da9c0e9



Tree

e6dab7bf1353cc6abf8f0f79811f26ff7da9c0e9

.git/objects

└── 98

 └── 0a0d5f19a64b4b30a87d4206aade58726b60e3

└── e6

 └── 9de29bb2d1d6434b8b29ae775ad8c2e48c5391

└── info

└── pack

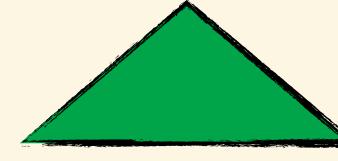
Quiz Time!

Suppose you have the following directory structure.

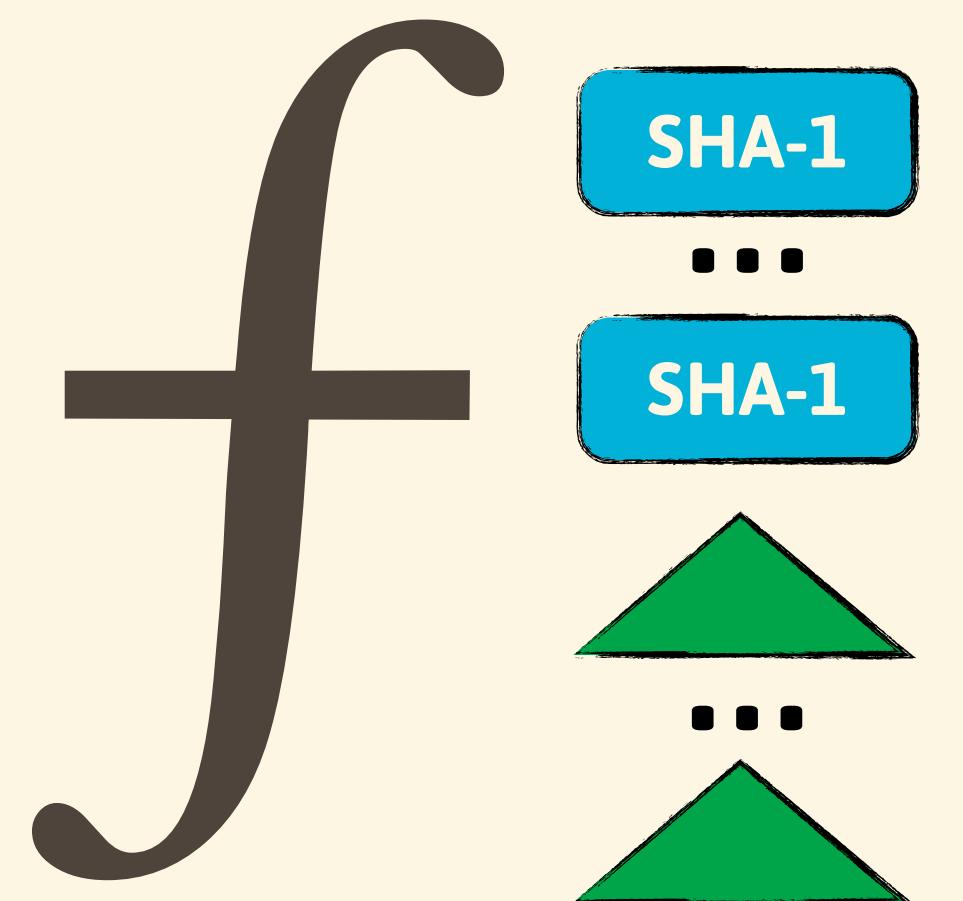
```
.  
└── README.md  
└── src  
    └── HelloWorld.js
```

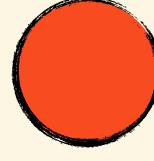
You "git cat-file -p <tree-sha>". How many lines will you see?

- Just one for the blob
- Two. One for for the blob and one for the nested tree
- Three — Two for blobs and one for the nested tree



Tree





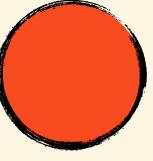
Commit

References a tree

0..n "parent" commits

Author info and timestamp

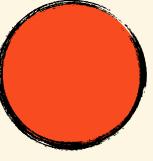
Commit message body



(Initial) Commit

```
tree e6dab7bf1353cc6abf8f0f79811f26ff7da9c0e9
author Raju <raju.gandhi@gmail.com> 1587841642 -0400
committer Raju <raju.gandhi@gmail.com> 1587841642 -0400
```

```
feat: initial implementation of Hello world
```



(Initial) Commit

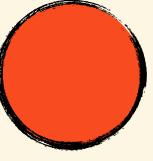
```
tree e6dab7bf1353cc6abf8f0f79811f26ff7da9c0e9
```

State of the working dir

```
author Raju <raju.gandhi@gmail.com> 1587841642 -0400
```

```
committer Raju <raju.gandhi@gmail.com> 1587841642 -0400
```

```
feat: initial implementation of Hello world
```



(Initial) Commit

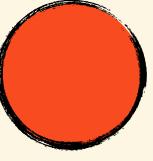
```
tree e6dab7bf1353cc6abf8f0f79811f26ff7da9c0e9
```

```
author Raju <raju.gandhi@gmail.com> 1587841642 -0400
```

```
committer Raju <raju.gandhi@gmail.com> 1587841642 -0400
```

Commit metadata

```
feat: initial implementation of Hello world
```

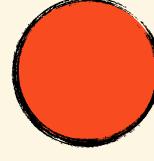


(Initial) Commit

```
tree e6dab7bf1353cc6abf8f0f79811f26ff7da9c0e9
author Raju <raju.gandhi@gmail.com> 1587841642 -0400
committer Raju <raju.gandhi@gmail.com> 1587841642 -0400
```

feat: initial implementation of Hello world

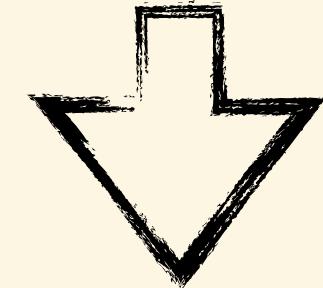
Commit message



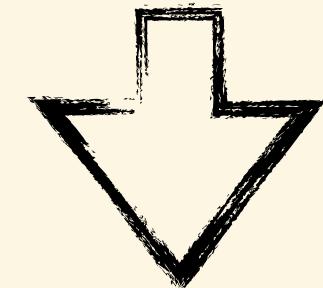
Commit

```
tree e6dab7bf1353cc6abf8f0f79811f26ff7da9c0e9
author Raju <raju.gandhi@gmail.com> 1587841642 -0400
committer Raju <raju.gandhi@gmail.com> 1587841642 -0400
```

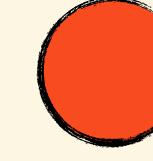
```
feat: initial implementation of Hello world
```



SHA-1 hash



ff4d71c72a91378206f6ce04e9153aeb02316737



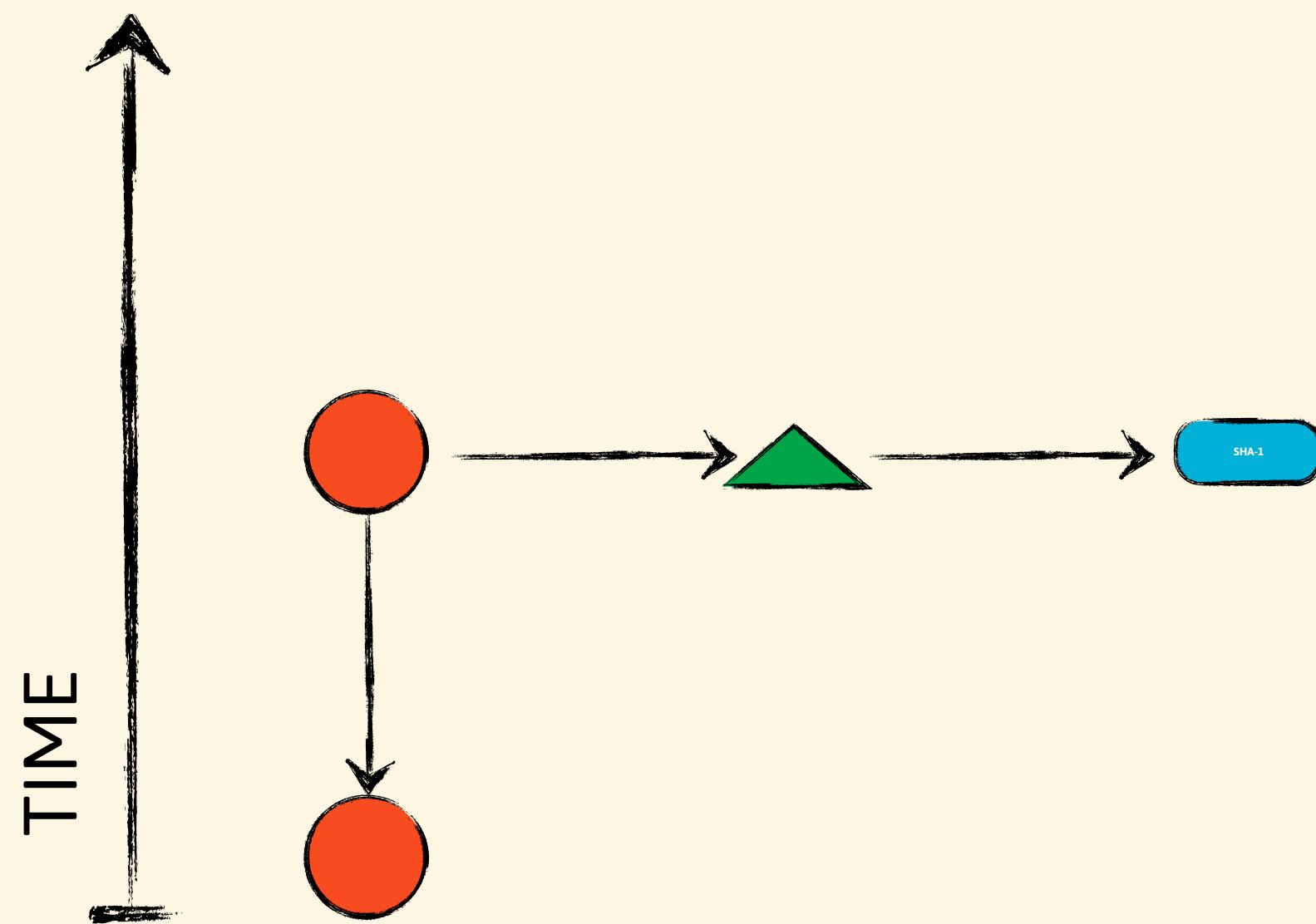
Commit

ff4d71c72a91378206f6ce04e9153aeb02316737

.git/objects

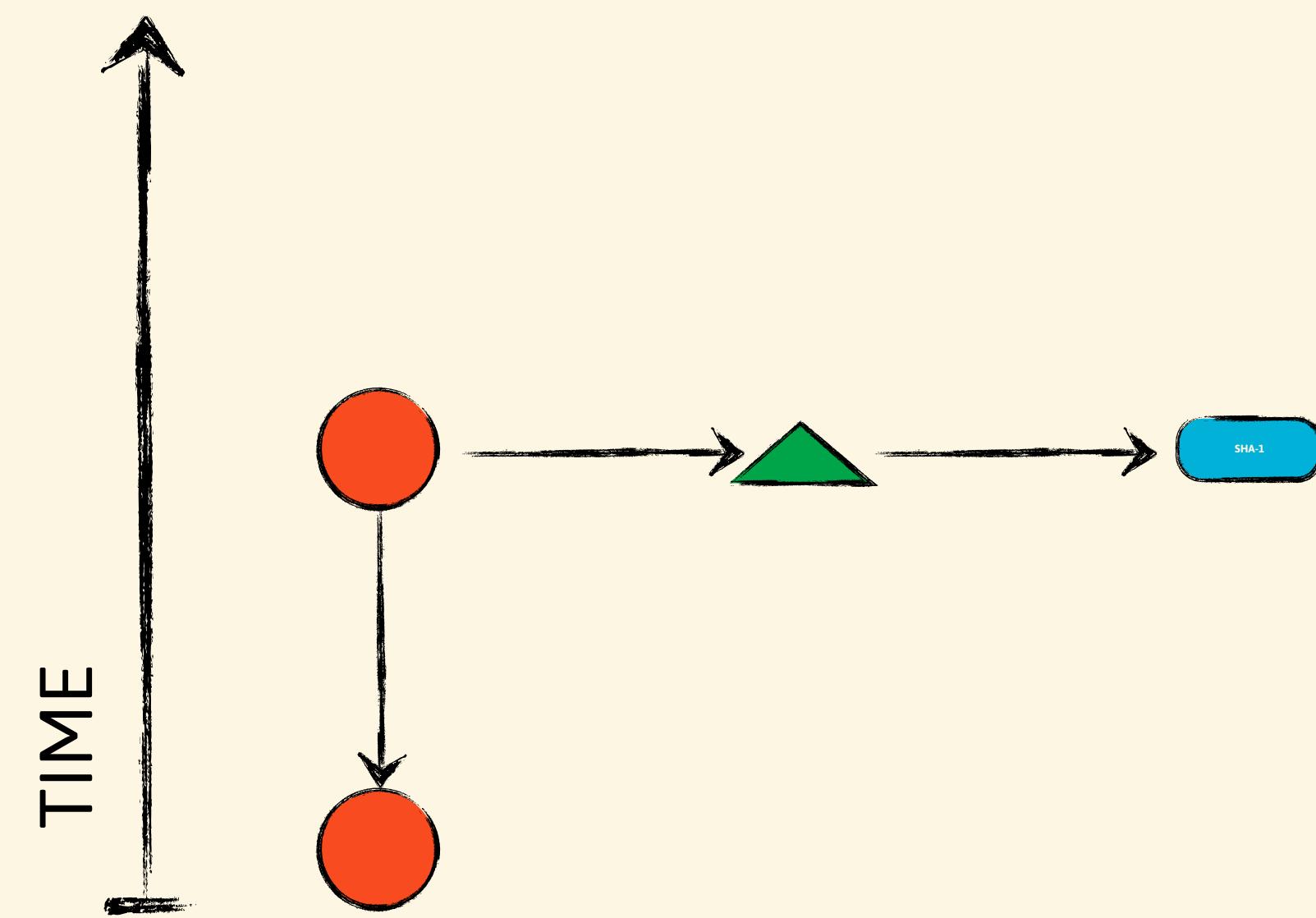
```
  └── 98
      └── 0a0d5f19a64b4b30a87d4206aade58726b60e3
  └── e6
      └── 9de29bb2d1d6434b8b29ae775ad8c2e48c5391
  └── ff
      └── 4d71c72a91378206f6ce04e9153aeb02316737
  └── info
  └── pack
```

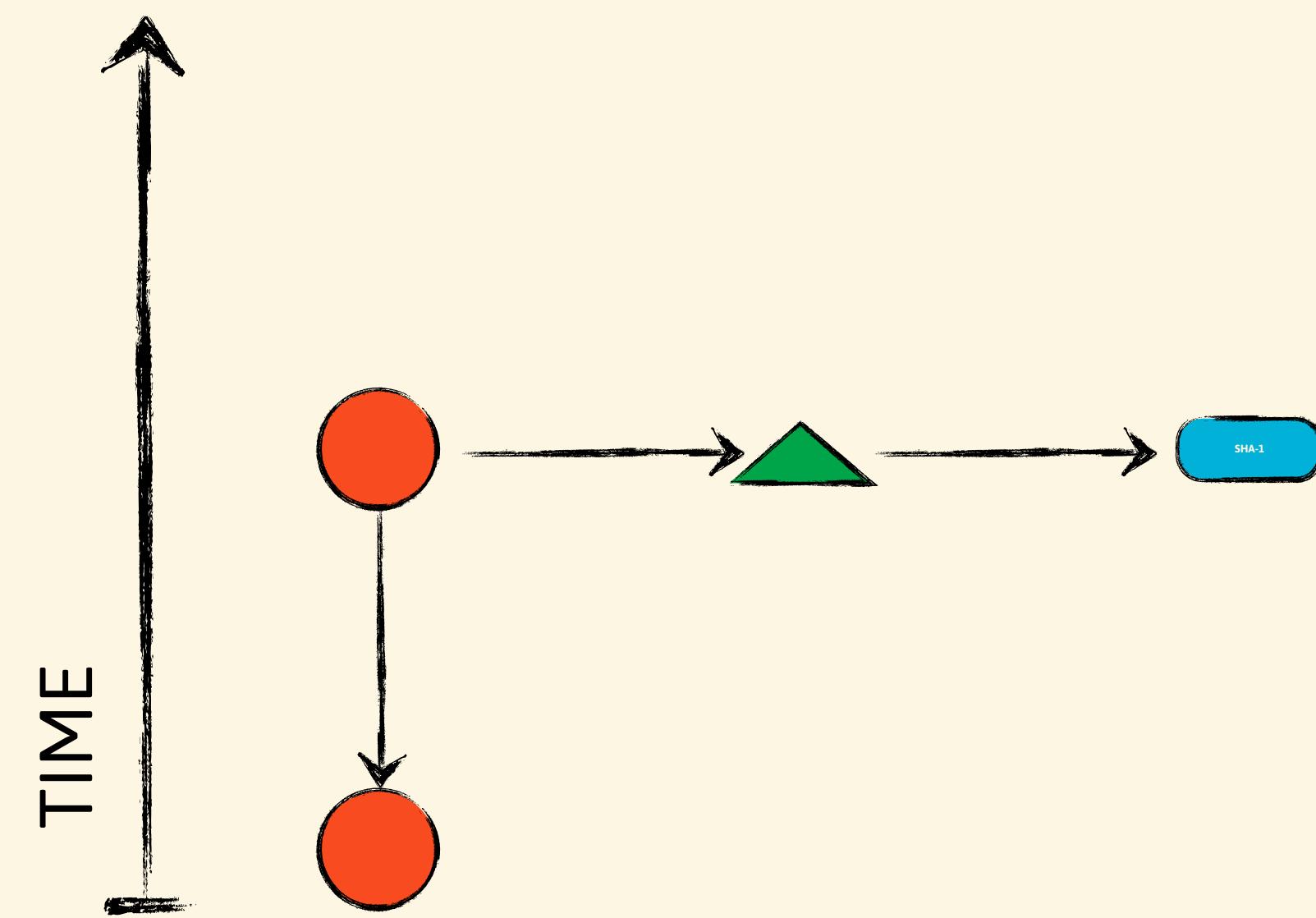
Commit



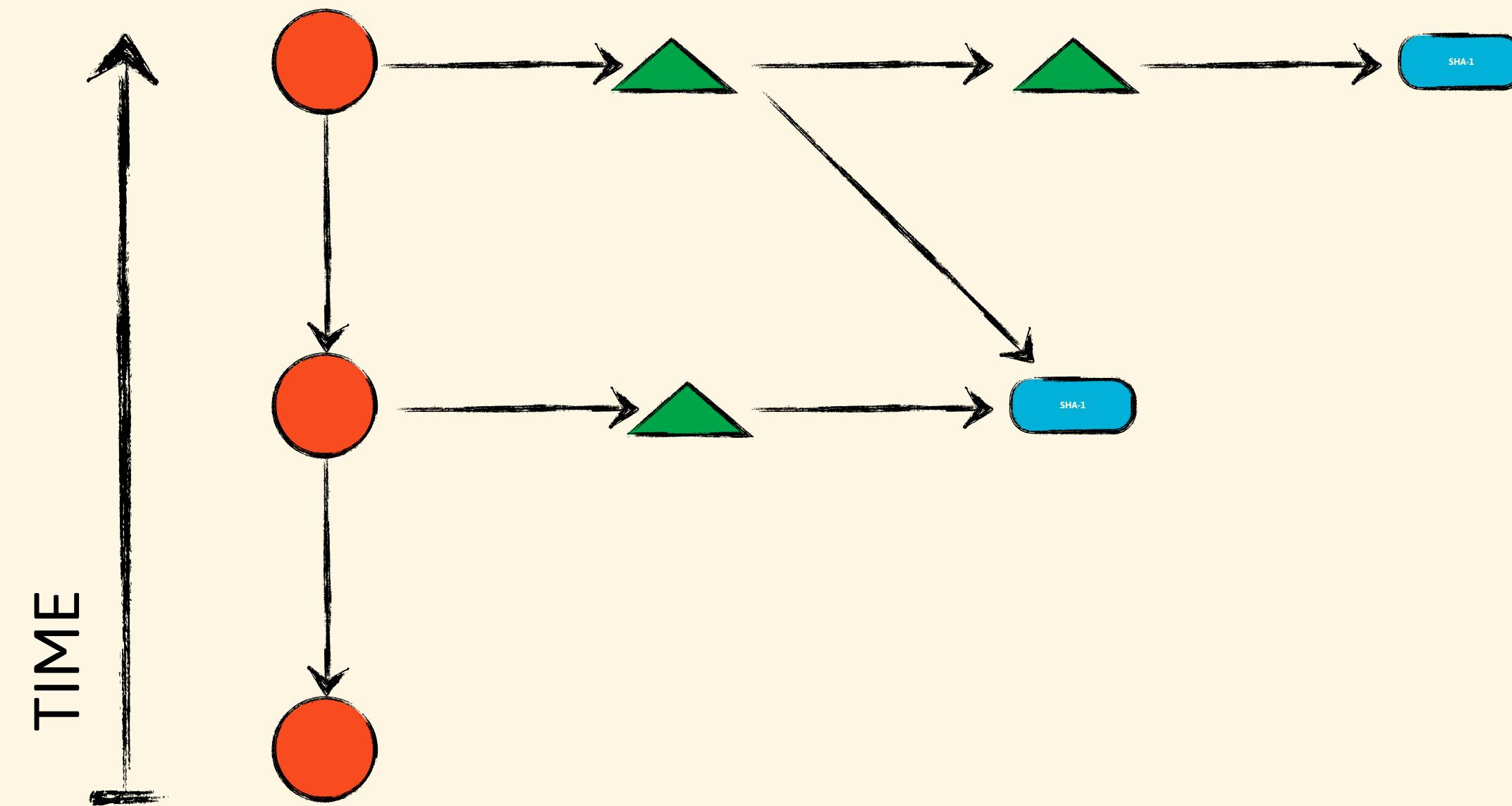
Git commits the **entire** tree per commit

**Say
Whatt???**





```
(master) > (mkdir lib && touch lib/newfile.txt)
(master) > git add .
(master) > git commit -m "New dir/file"
```



```
(master) > (mkdir lib && touch lib/newfile.txt)
(master) > git add .
(master) > git commit -m "New dir/file"
```

*" While a tree represents a particular directory state of a working directory, a commit represents that **state in "time", and explains how to get there***

- git-commit-tree man page

Quiz Time!

Suppose you have committed all your work and have a clean working directory. Also assume there is a way (and there is) to edit the message of a commit. Will this cause the ID of the commit to change. Explain your answer.

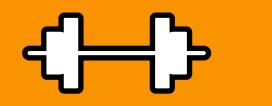
Yes

No

Quiz Time!

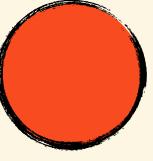
If two different people in two different repositories create the same directory structure with the exact same content, and commit it, will the commit IDs be the same? Explain your answer

- Yes
- No
- Maybe



Exercise

- Create a bunch of commits in your repository (Be sure to add several directories and subdirectories containing files)
- Inspect one or more commits using "git cat-file -p <commit-sha>" (You can use git log to find the ID of a commit)
- Use "git show <commit-sha>" and see what information it displays



(Most) Commits

```
tree e6dab7bf1353cc6abf8f0f79811f26ff7da9c0e9
```

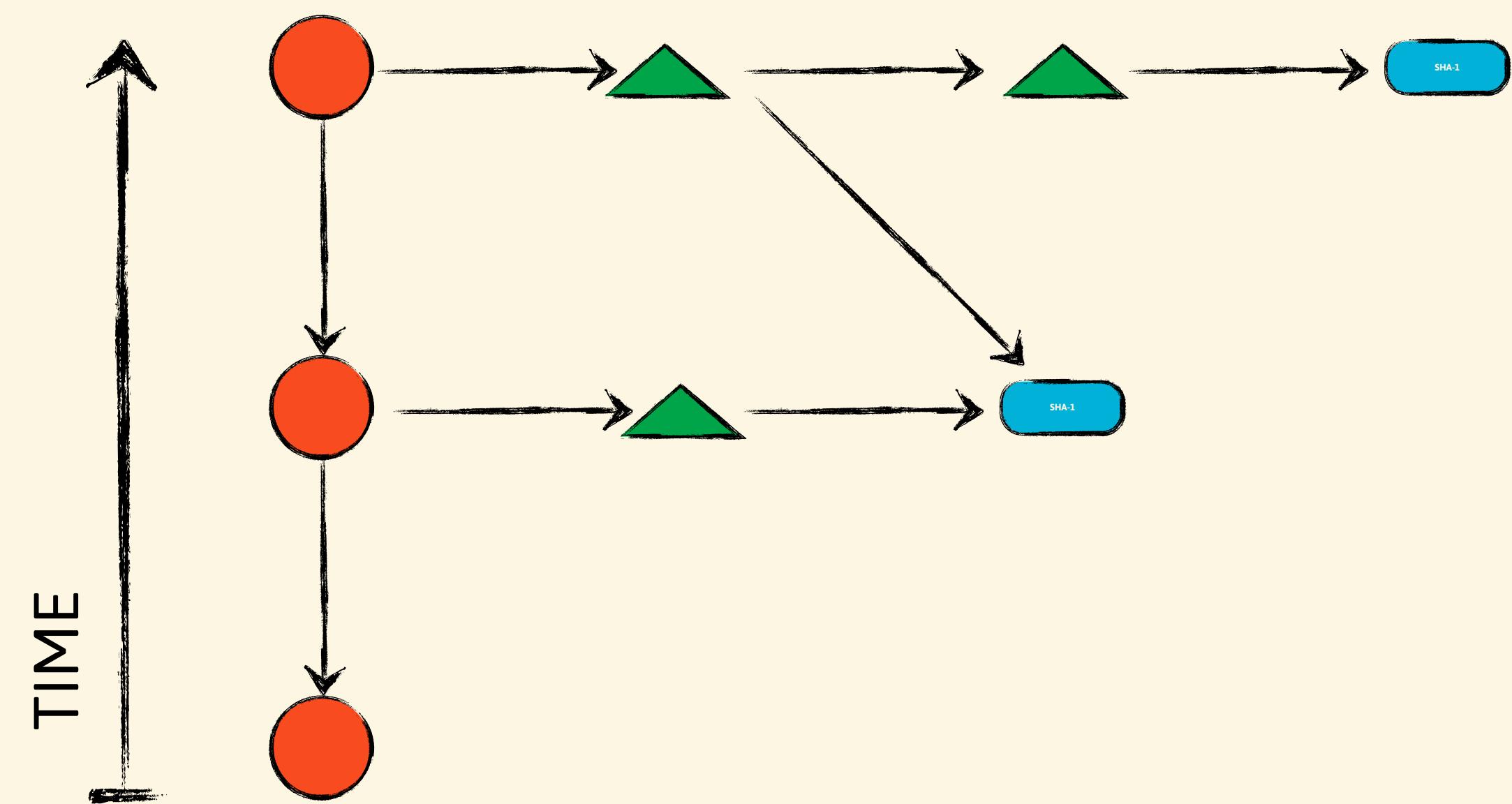
```
parent 08de1a6e2bc04f1758ea50799206cb26c55deb98
```

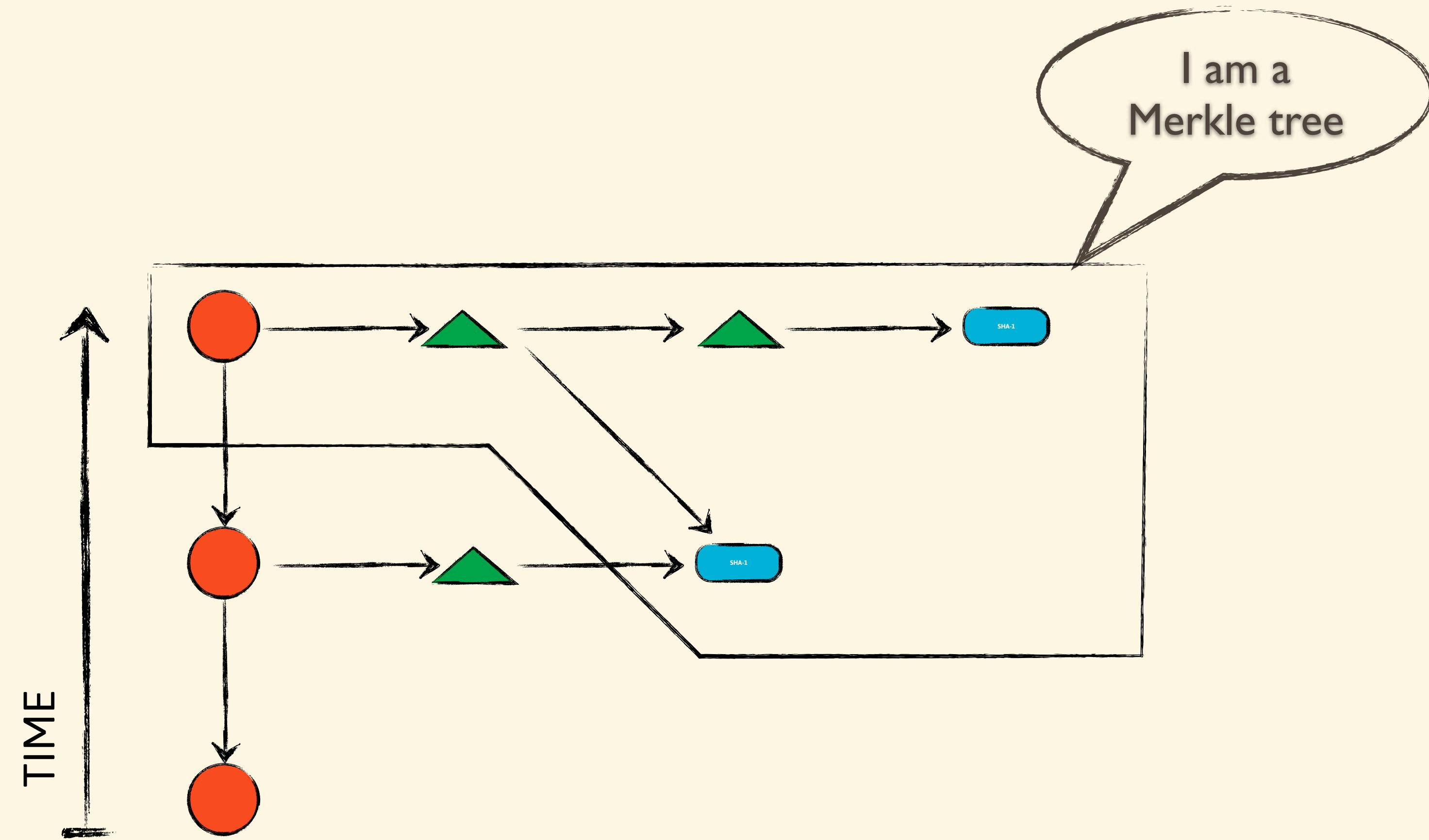
State of the working dir

```
author Raju <raju.gandhi@gmail.com> 1587841782 -0400
```

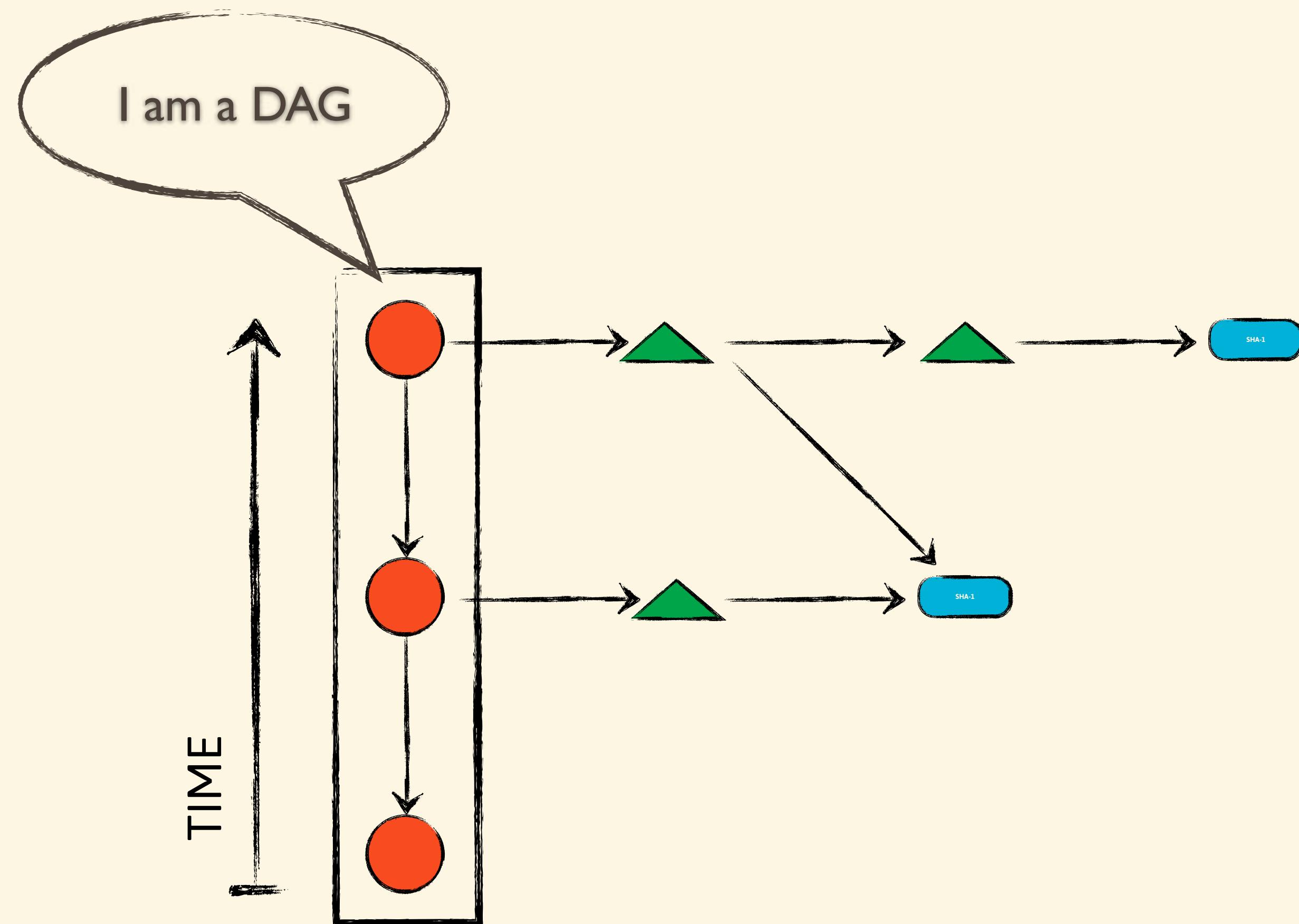
```
committer Raju <raju.gandhi@gmail.com> 1587841782 -0400
```

```
test: add a test for Hello World
```



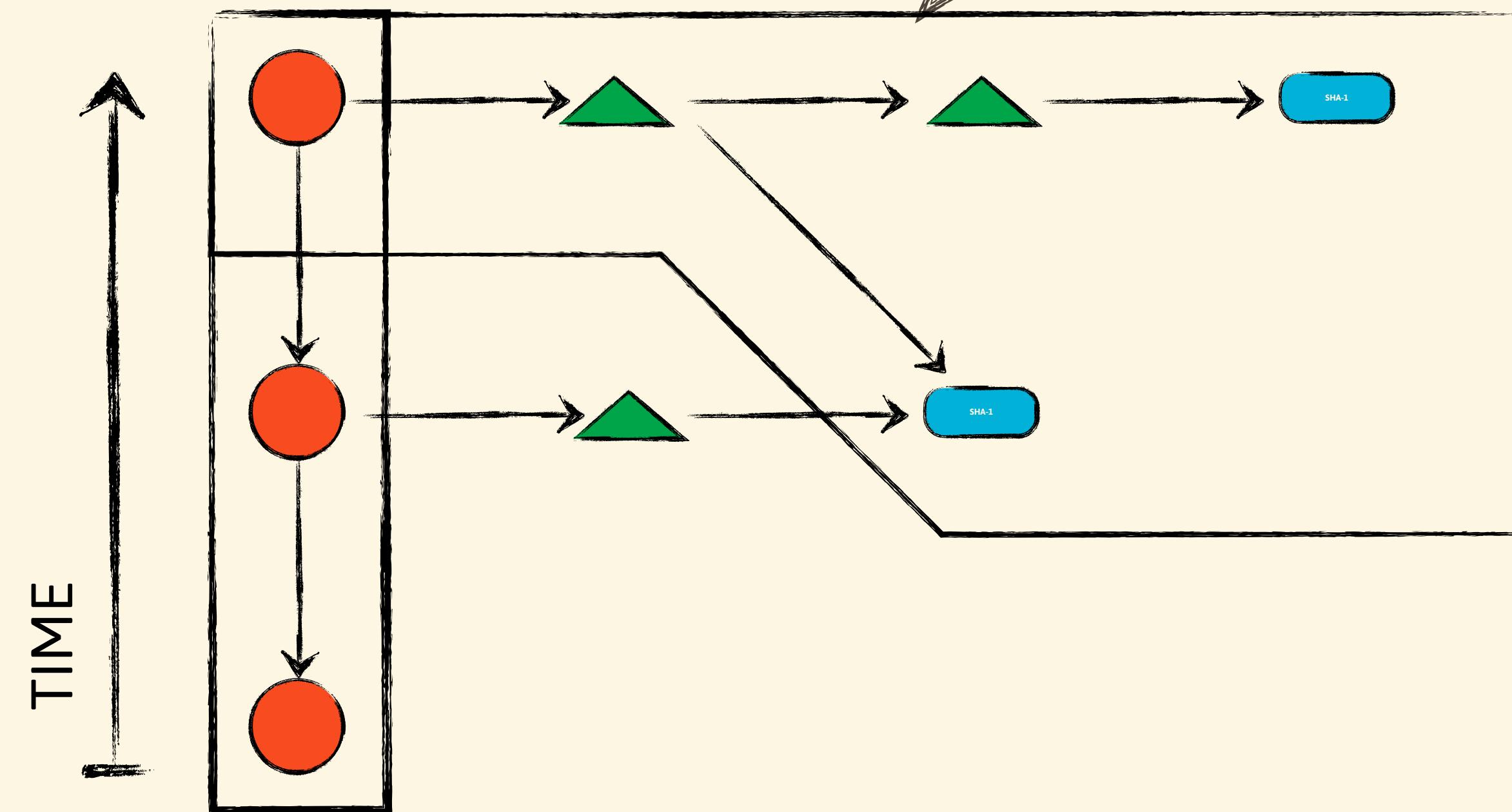


Merkle tree — A tree structure in which each leaf node is a hash of a block of data, and each non-leaf node is a hash of its children



DAG — A finite directed graph with no directed cycles

Together, we are a
Merkle-DAG



Branches / HEAD / Tags

config

description

HEAD

hooks

commit-msg.sample

...

info

exclude

objects

info

pack

refs

heads

tags

Symbolic Reference

References

Smile

to go ❤

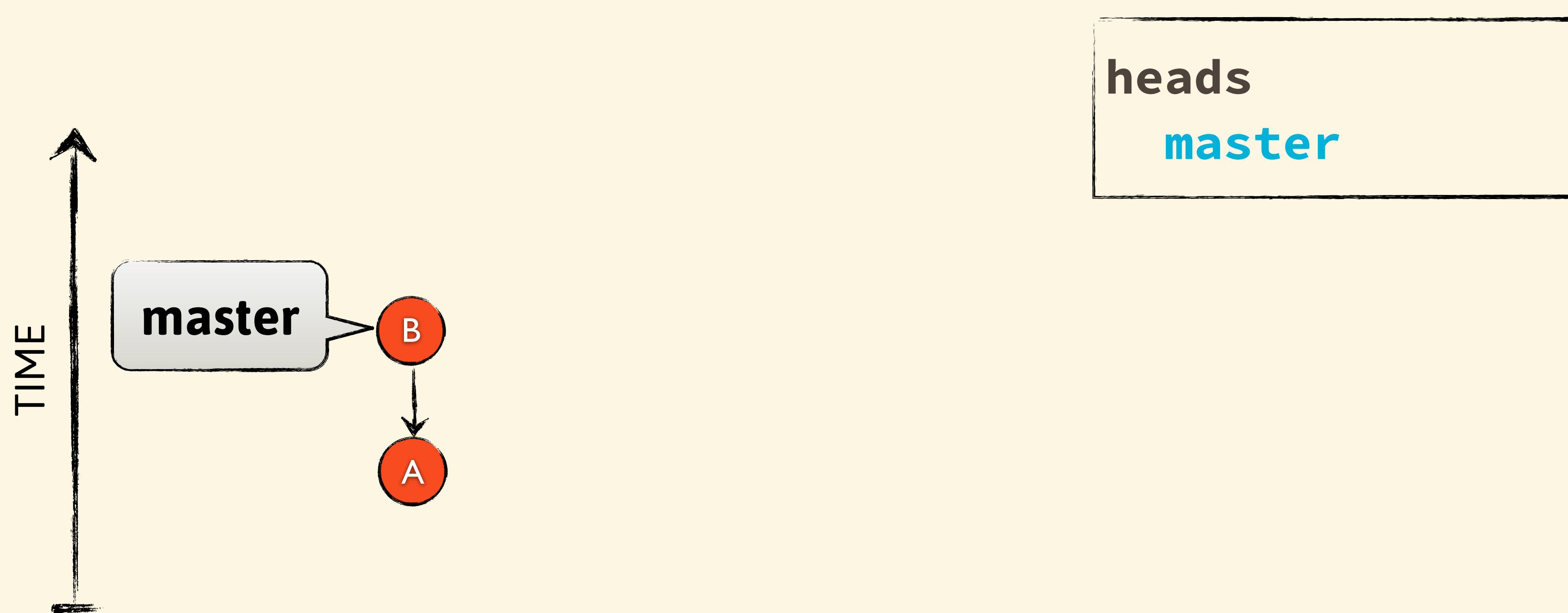
Branches
THE GIT WAY



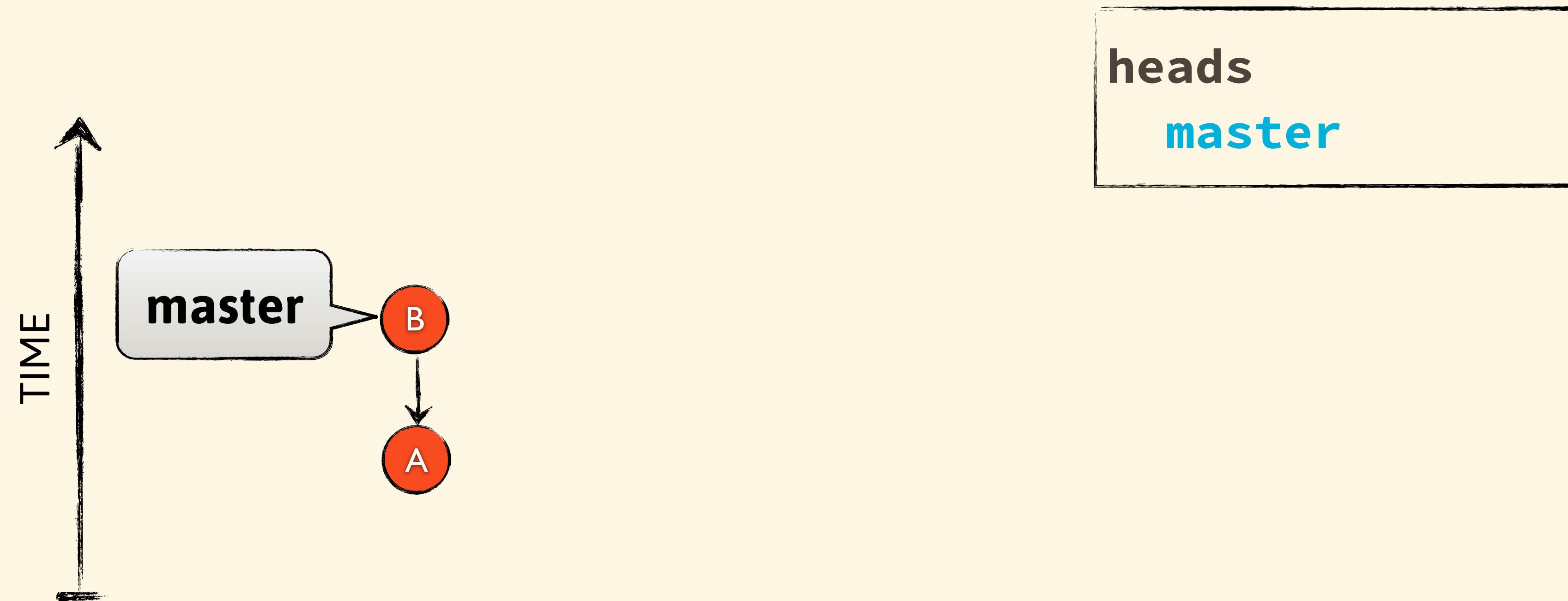
Branches

A named reference to a commit

Branches

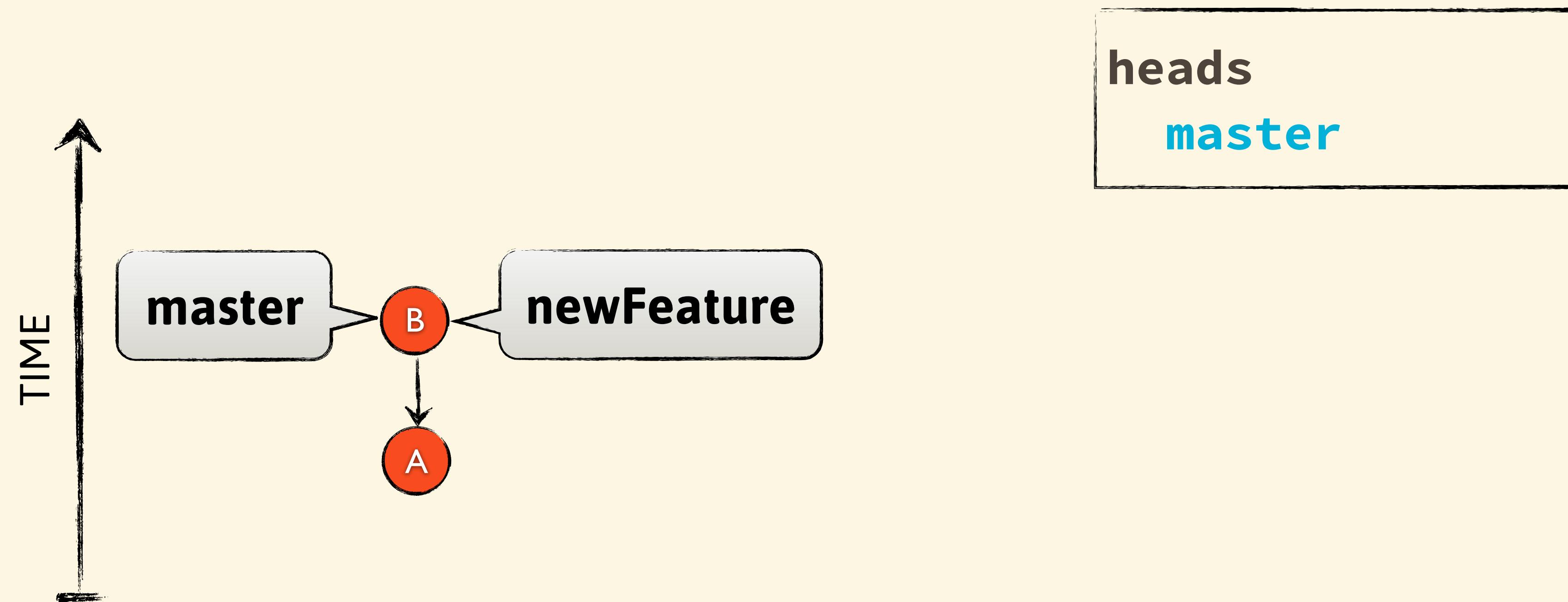


Branches



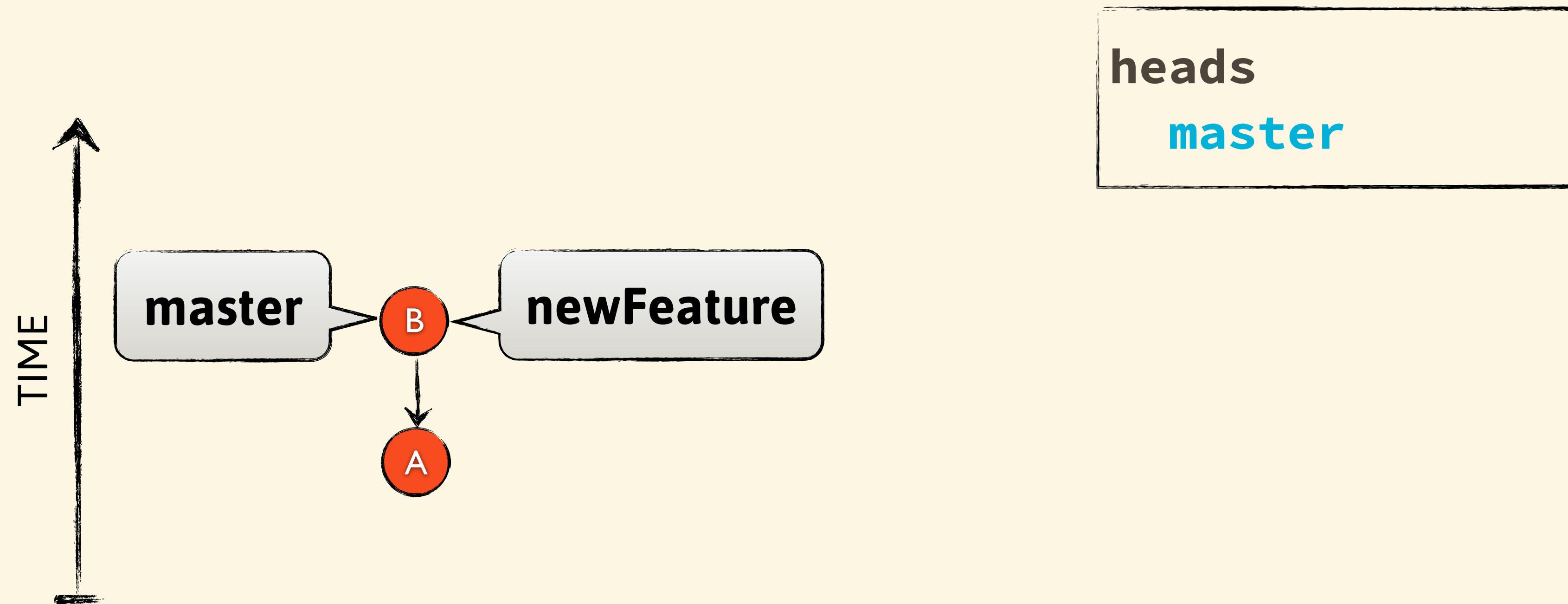
(master) > git switch -c newFeature

Branches



(master) > git switch -c newFeature

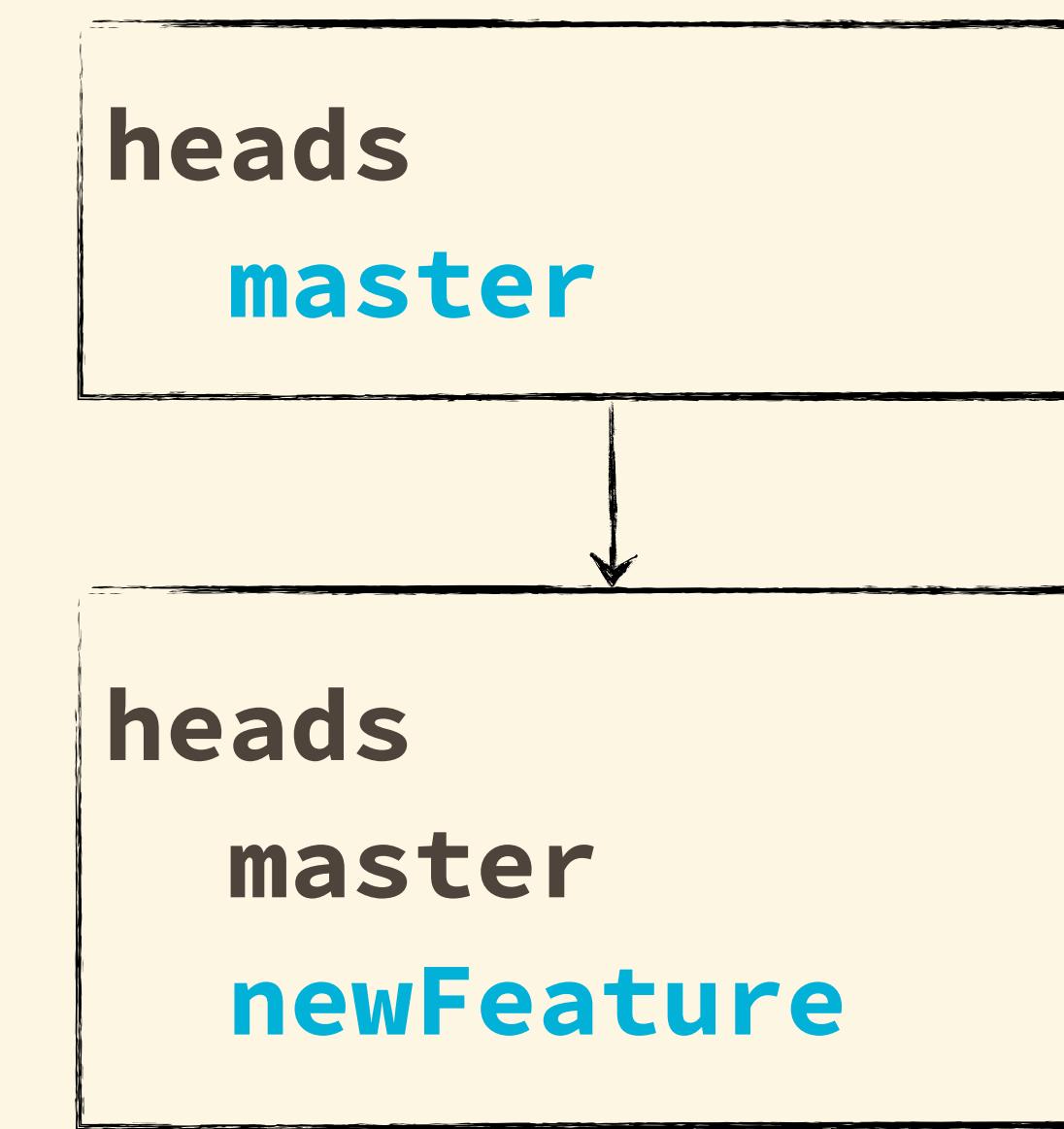
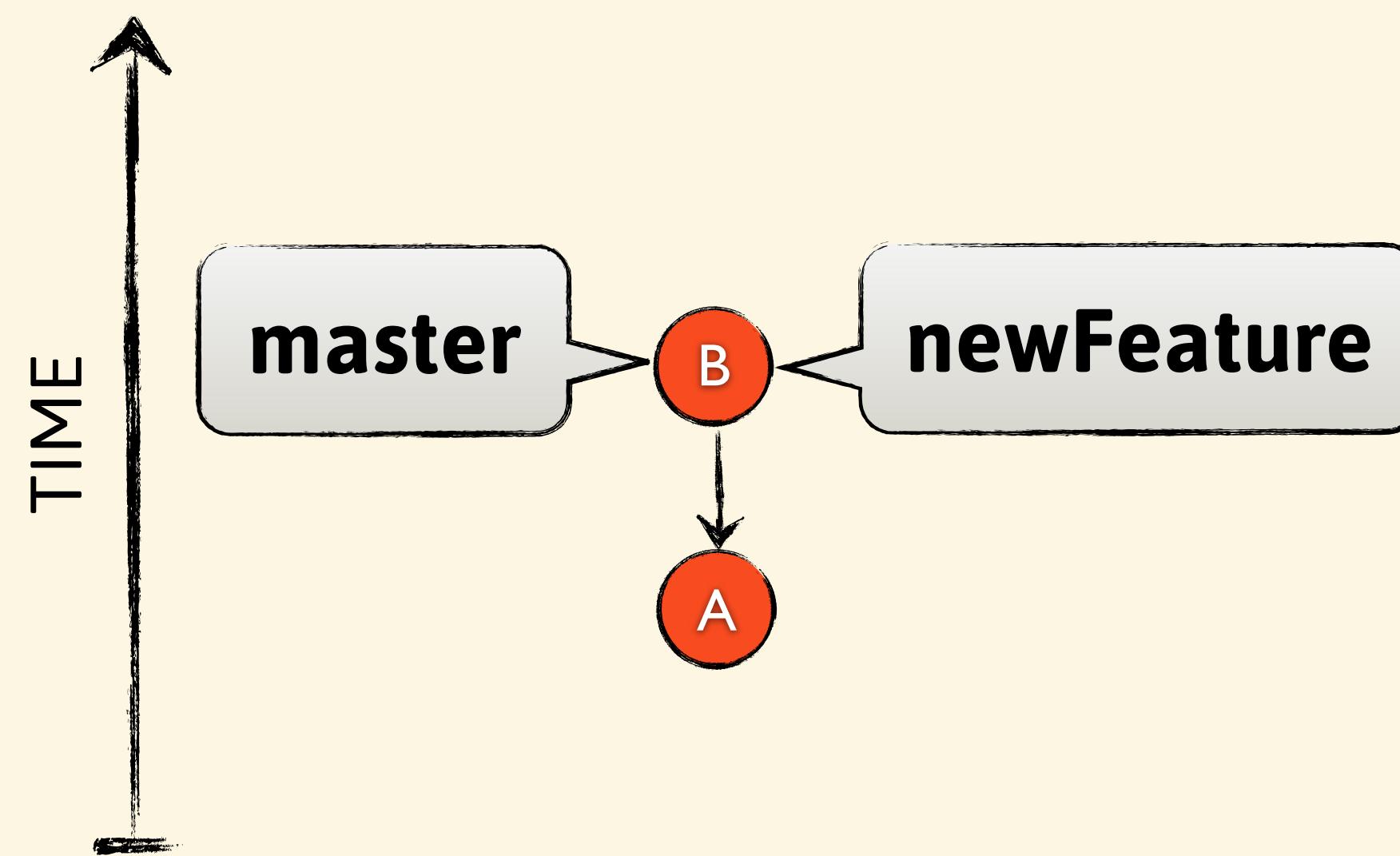
Branches



(master) > git switch -c newFeature

(newFeature) > (cd .git/refs/ && tree)

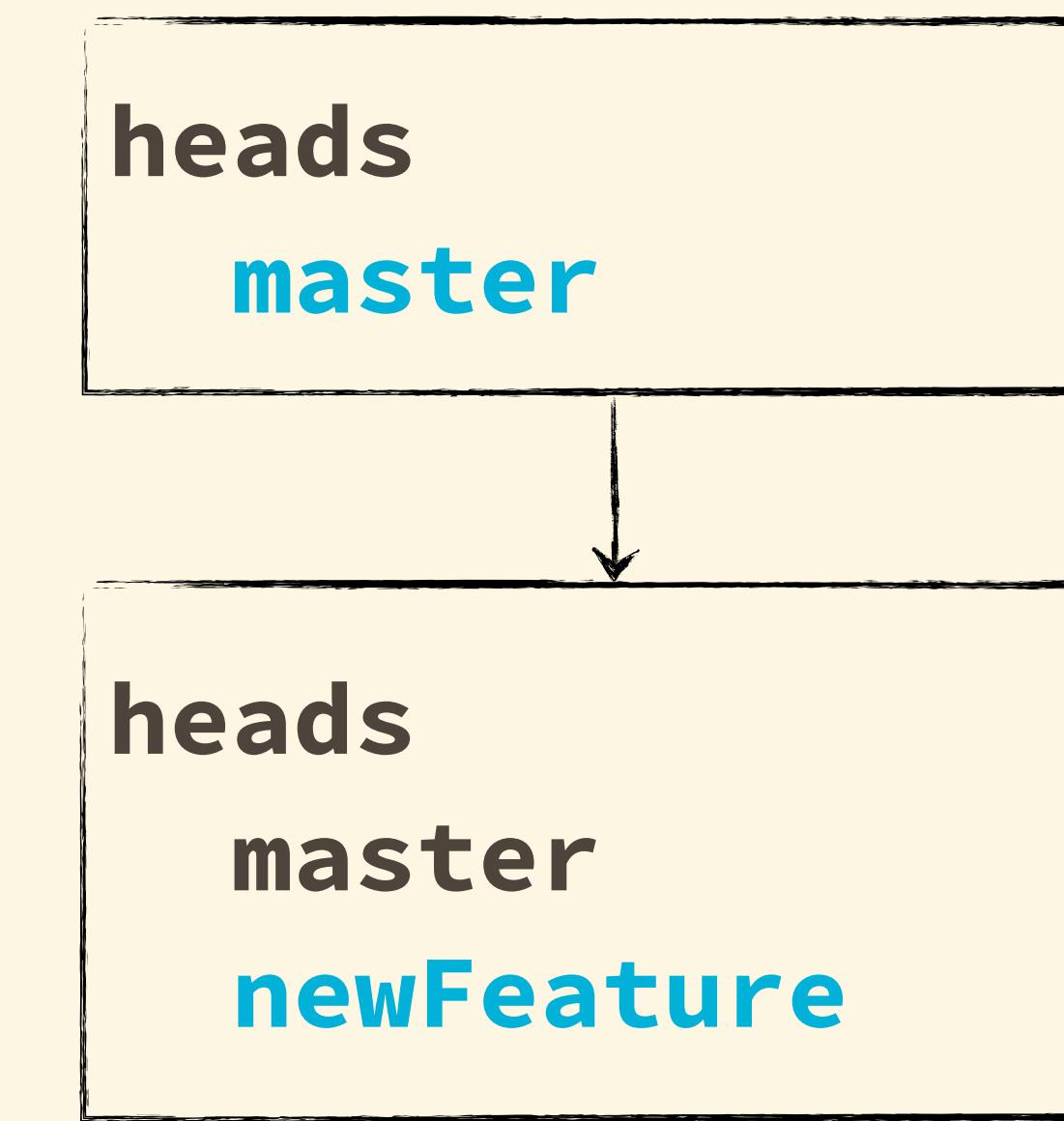
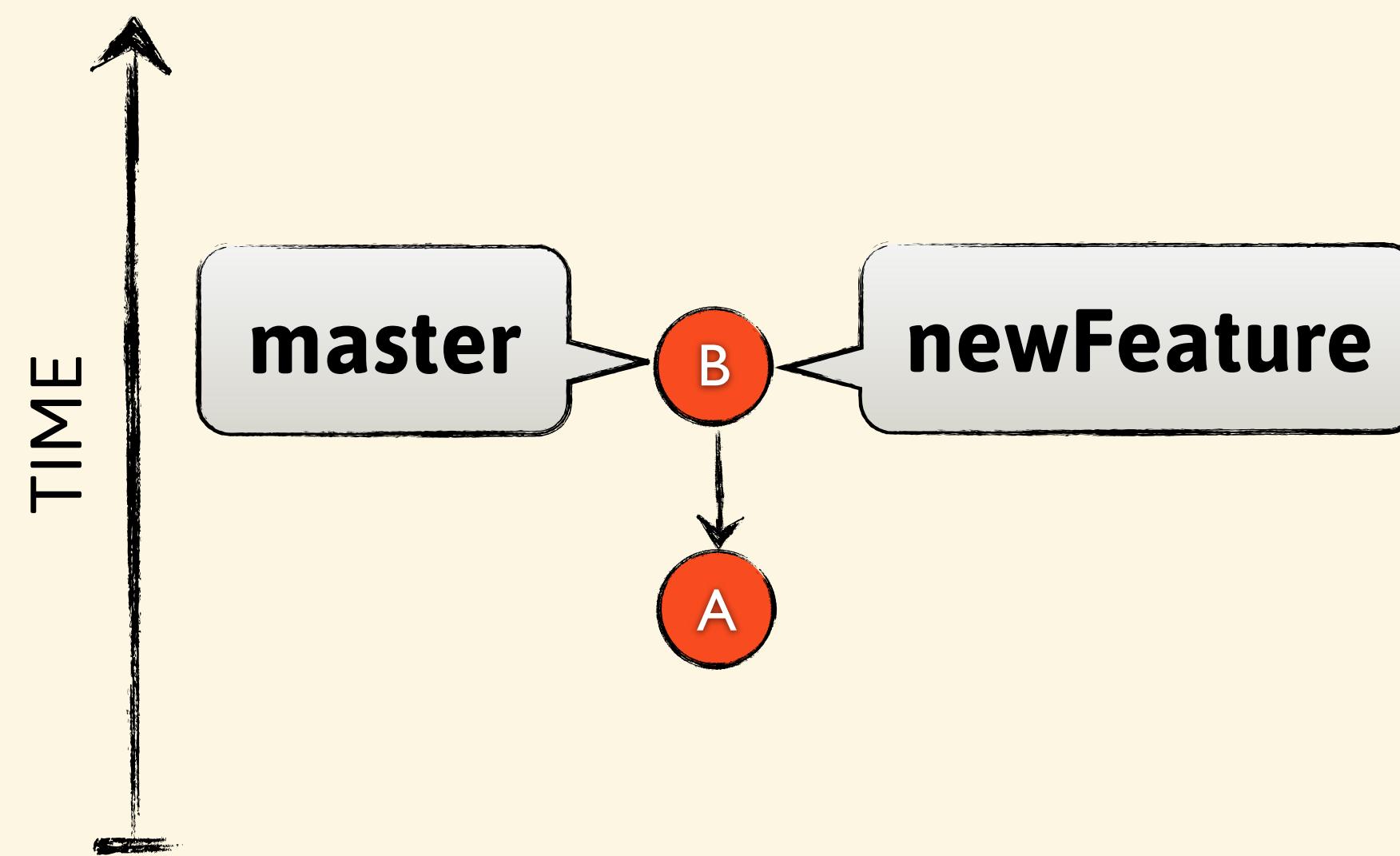
Branches



```
(master) > git switch -c newFeature
```

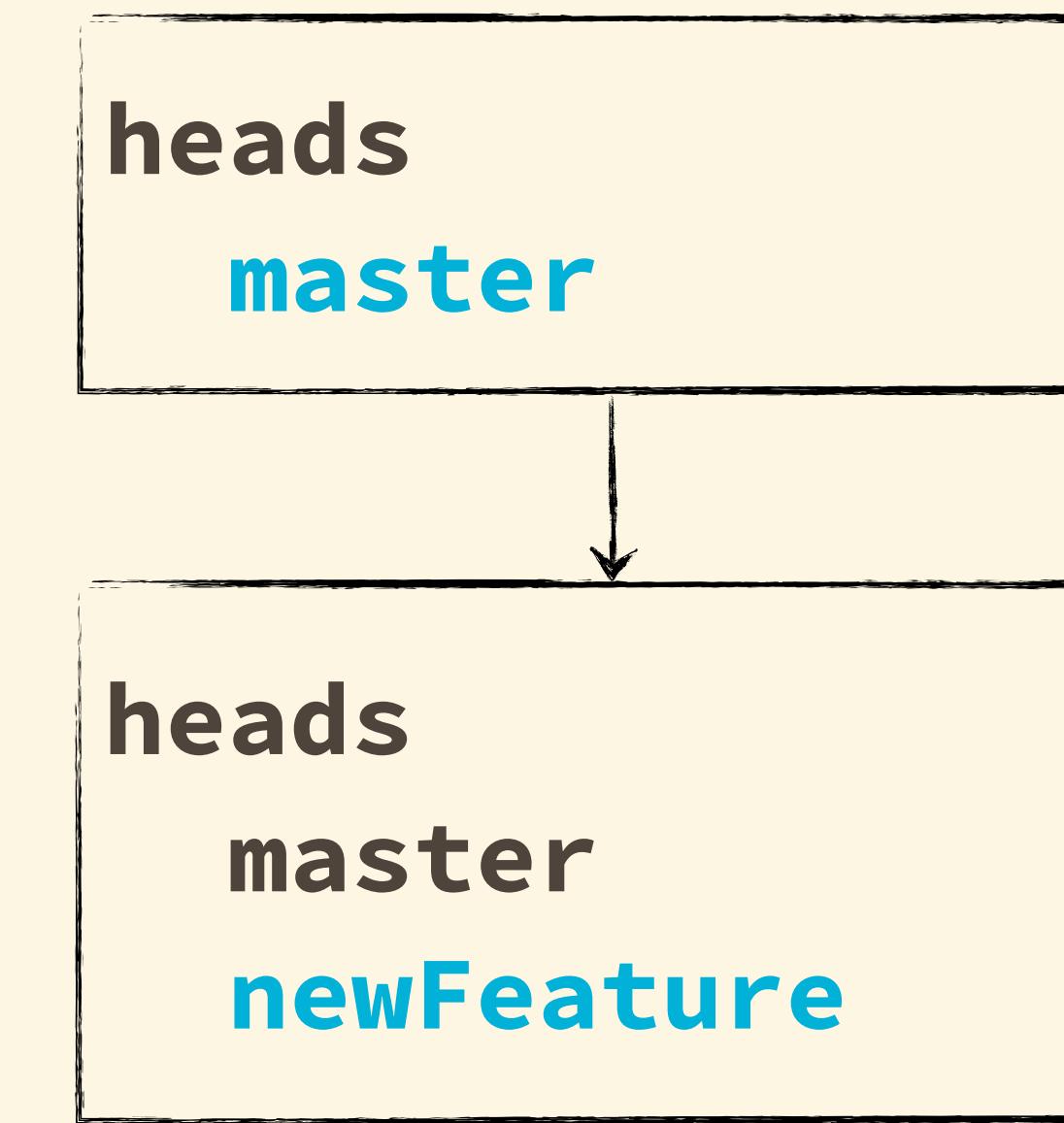
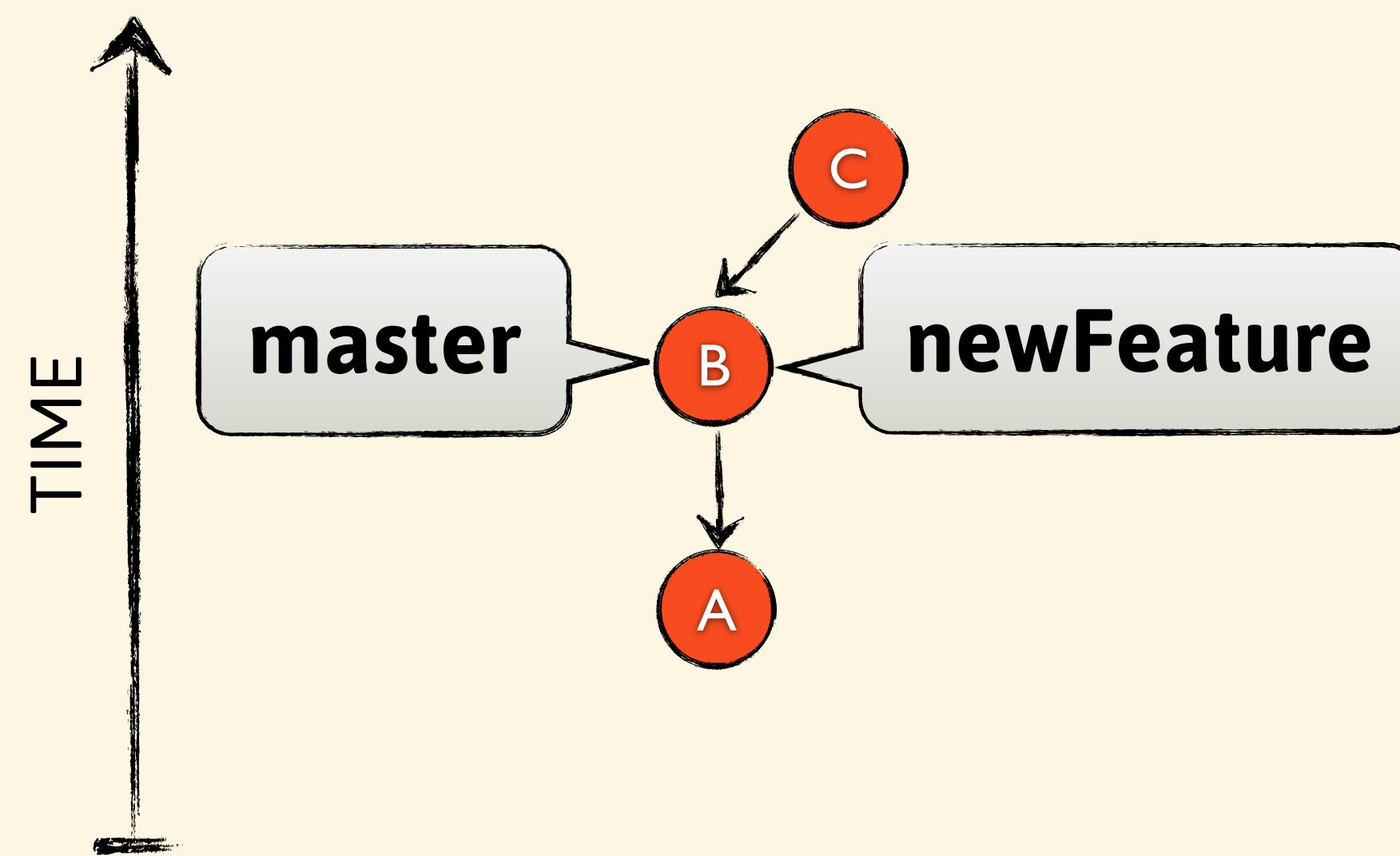
```
(newFeature) > (cd .git/refs/ && tree)
```

Branches



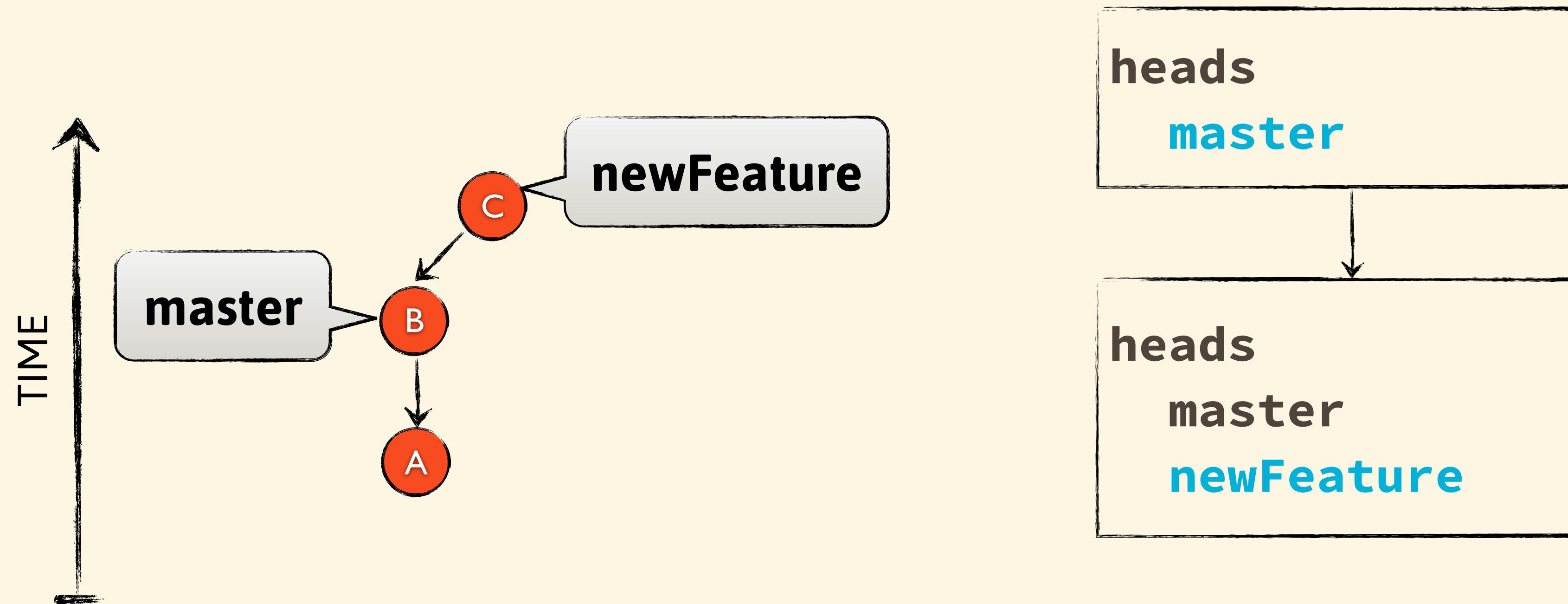
```
(master) > git switch -c newFeature
(newFeature) > (cd .git/refs/ && tree)
(newFeature) > git commit -m "Some Commit"
```

Branches



```
(master) > git switch -c newFeature  
(newFeature) > (cd .git/refs/ && tree)  
(newFeature) > git commit -m "Some Commit"
```

Branches

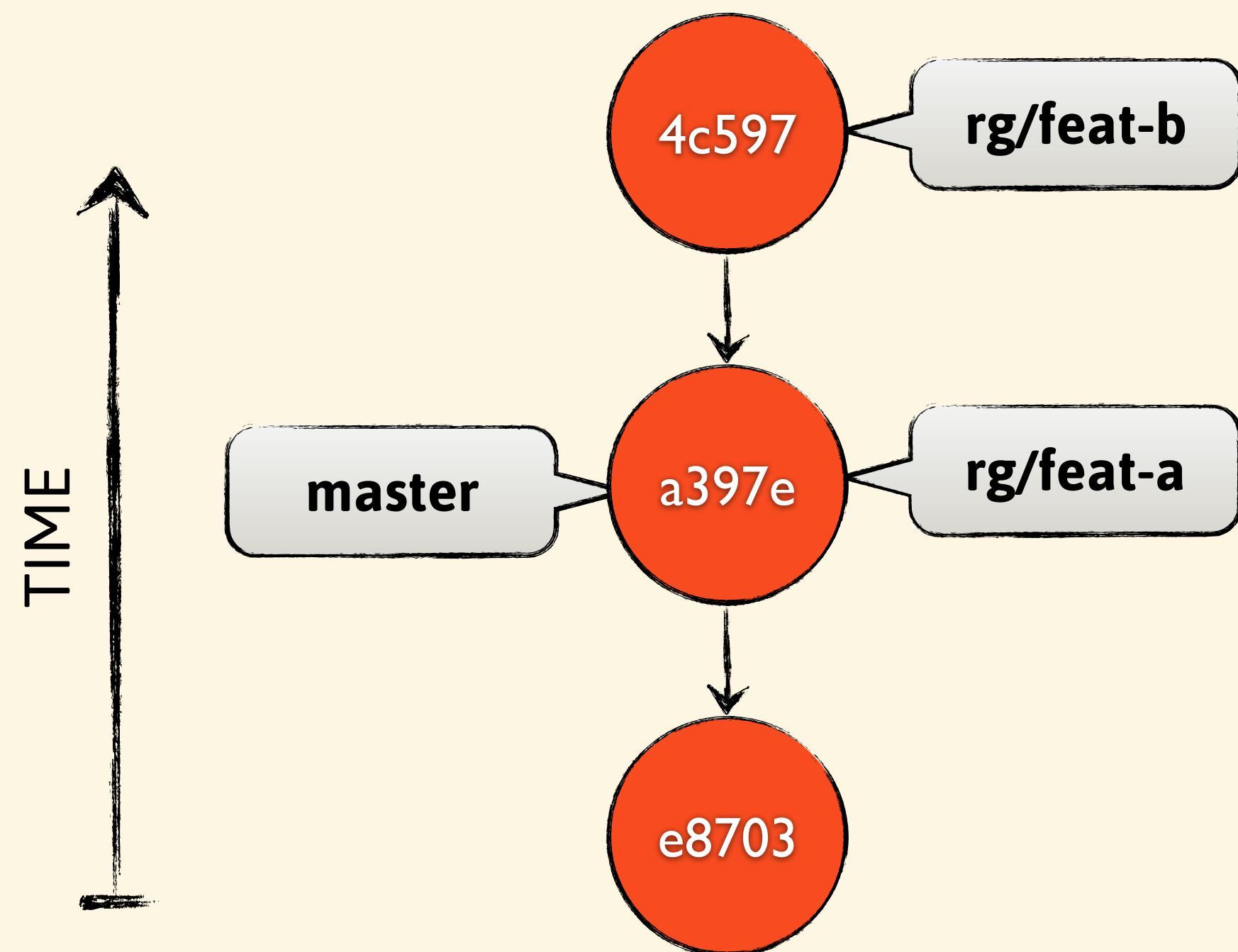


```
(master) > git switch -c newFeature  
(newFeature) > (cd .git/refs/ && tree)  
(newFeature) > git commit -m "Some Commit"
```

?

Quiz Time!

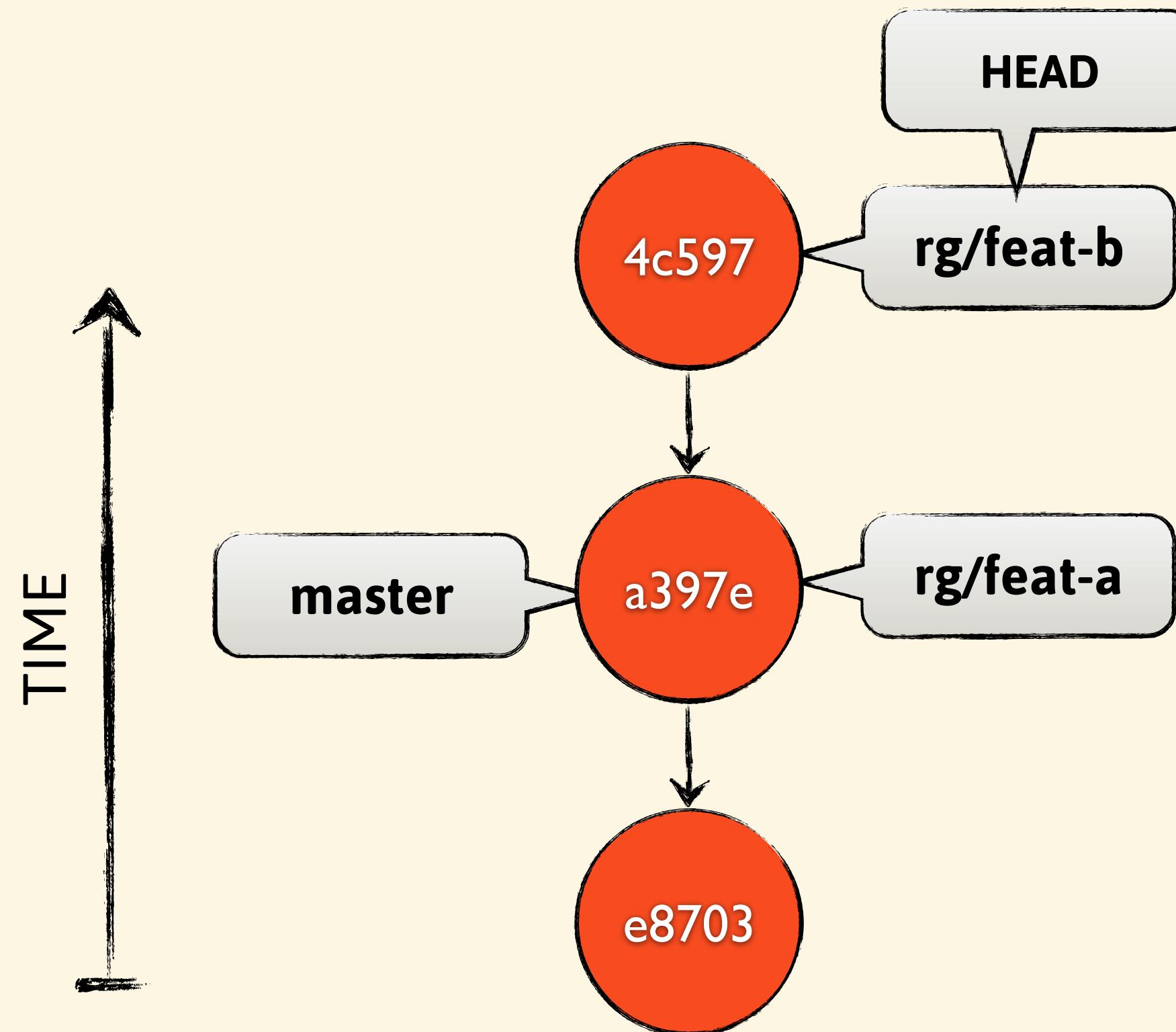
Given this DAG how many refs will you find under ".git/refs/heads" and what will be their respective contents?

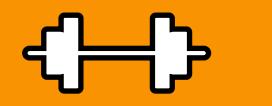


?

Quiz Time!

Given this DAG what will happen if you run "git branch -d rg/feat-a?





Exercise

- Create a branch in your repository and check the contents of the respective refs file under ".git/refs/heads"**
- Make a few commits on that branch and continuously inspect the respective refs file**

Branches

A branch implies
WORK-IN-PROGRESS!!!



**HEAD
NOT LOSING IT :)**

Head

Points to the **checked** out commit

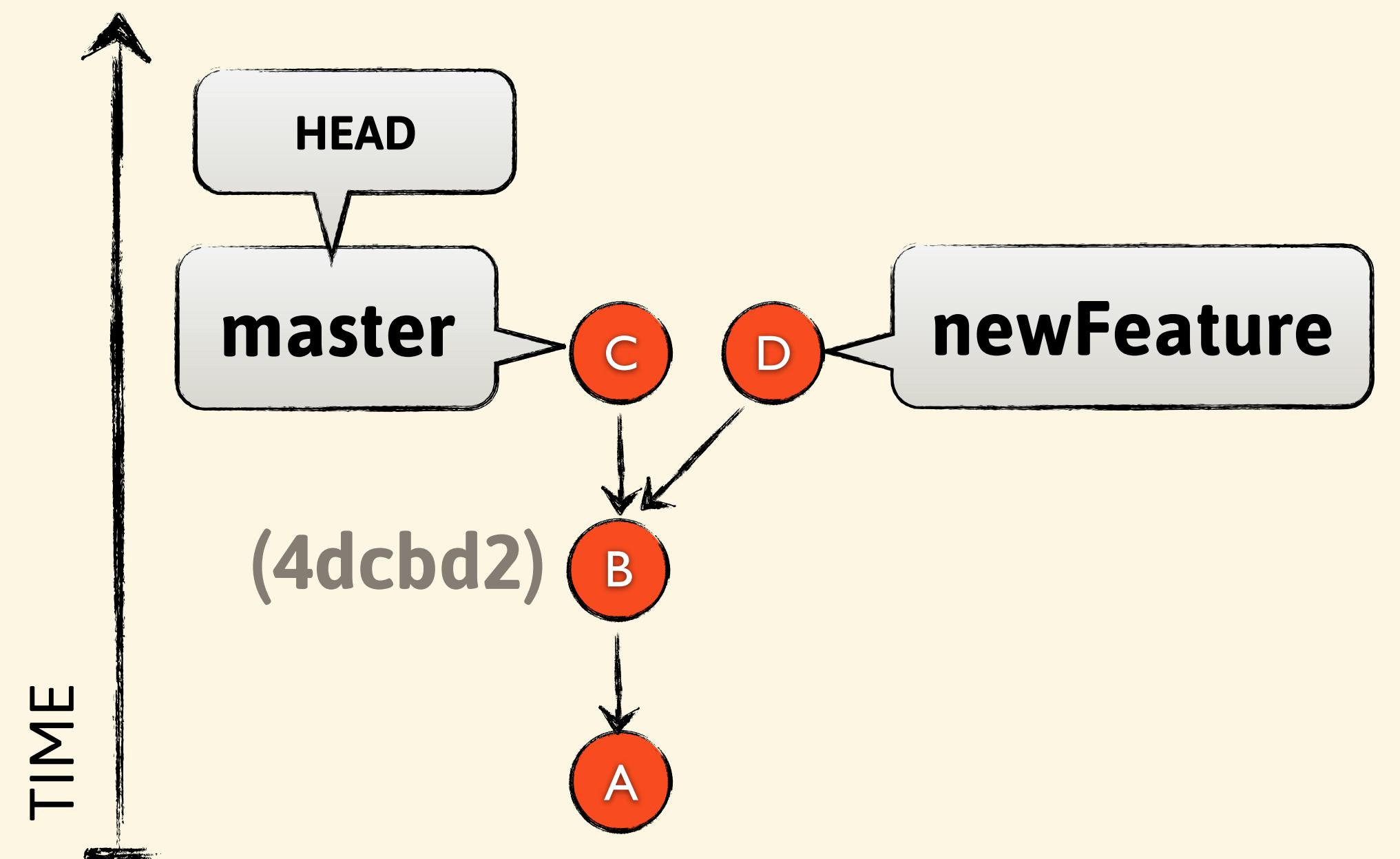
Head

The parent of the next commit

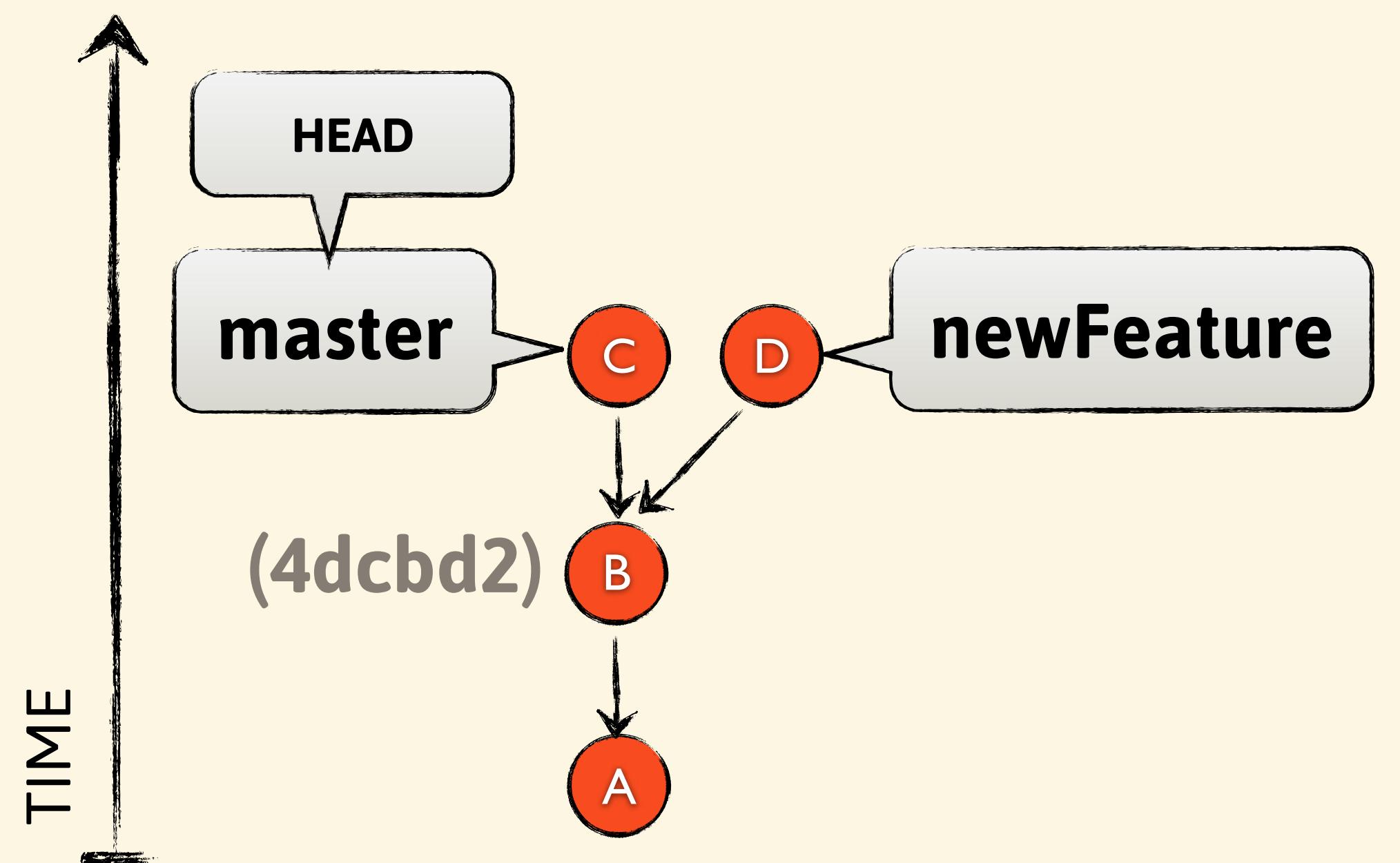
Head

Usually the last commit on that branch

Head

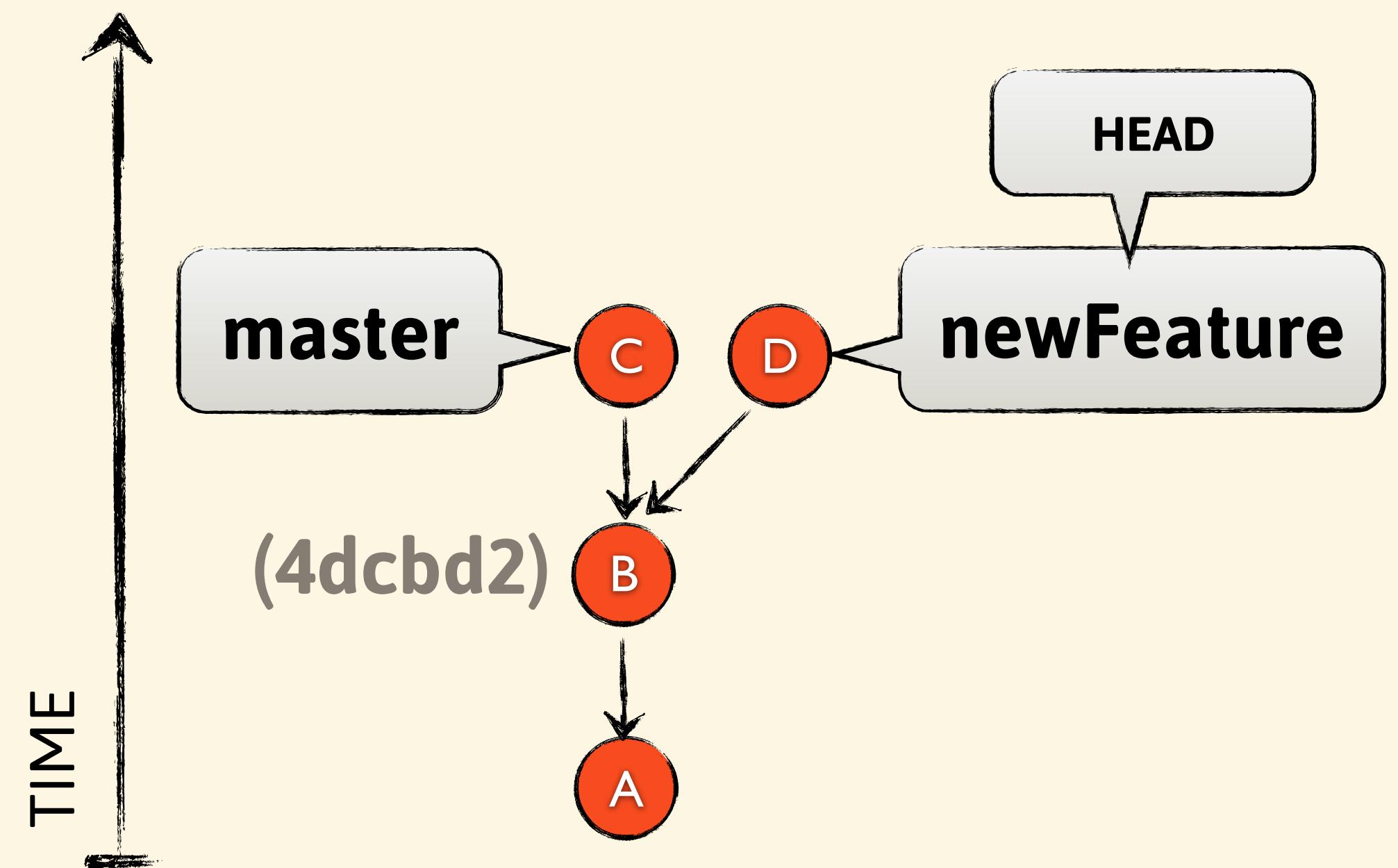


Head



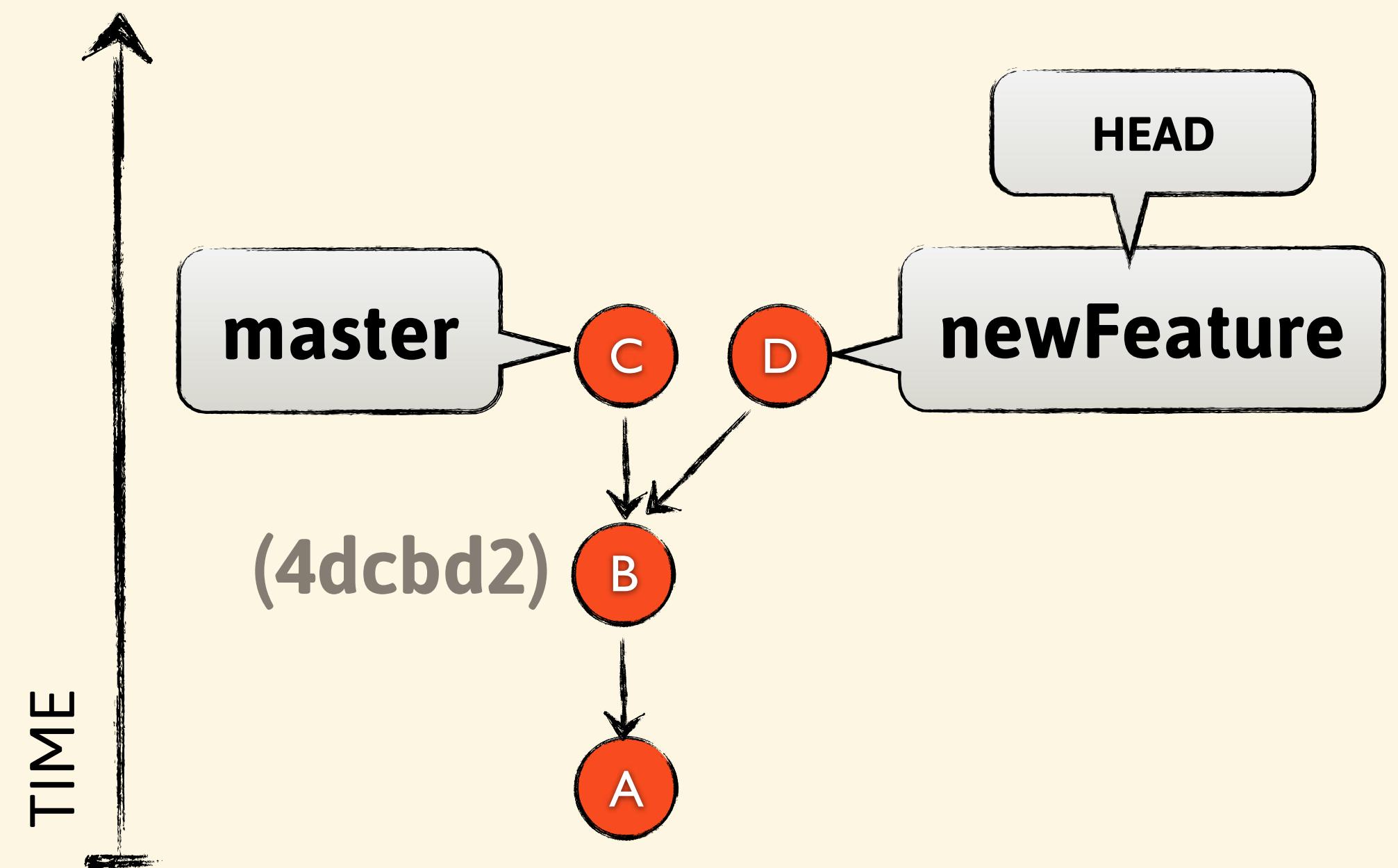
(master) > git switch newFeature

Head



(master) > git switch newFeature

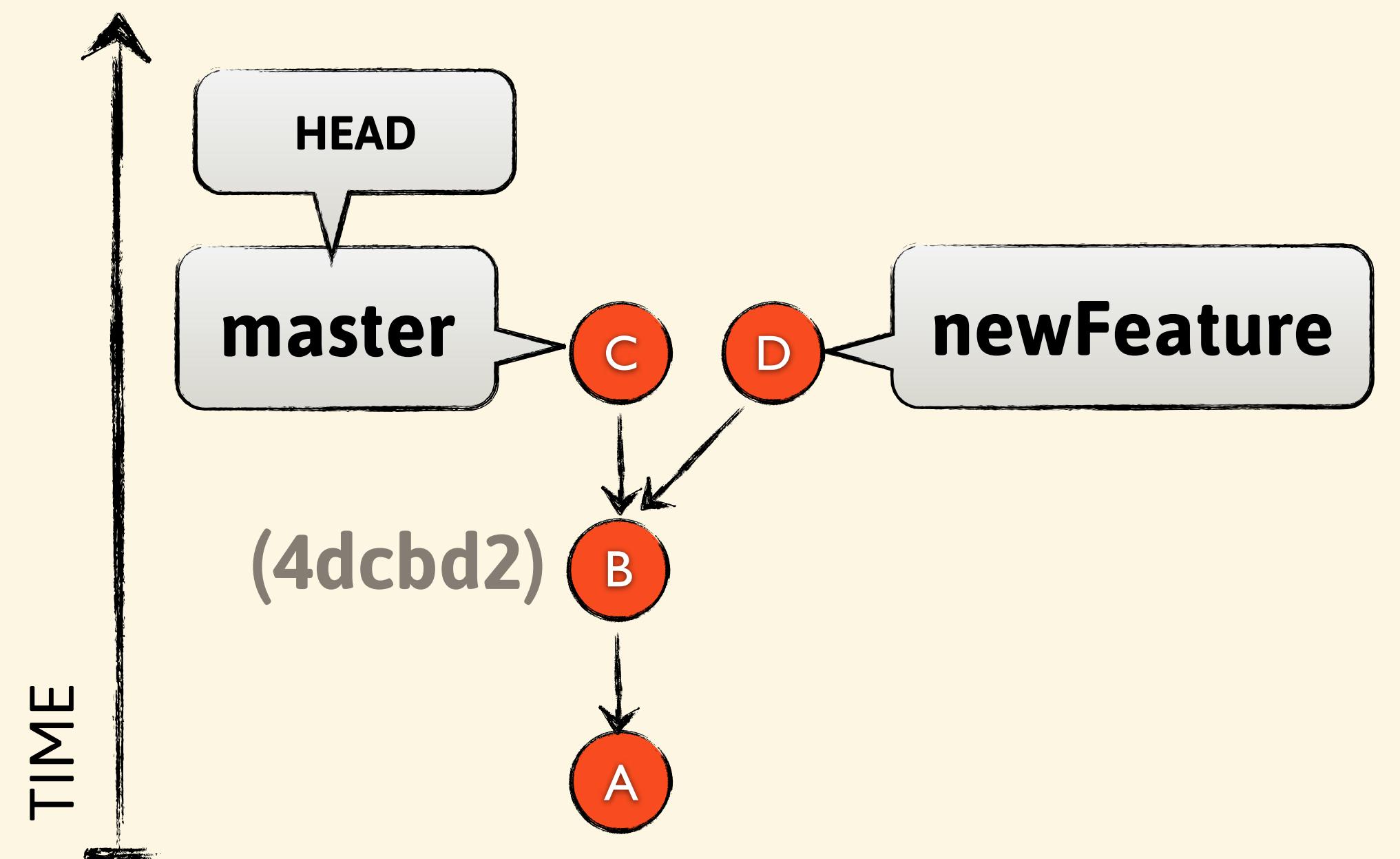
Head



(master) > git switch newFeature

(newFeature) > git switch master

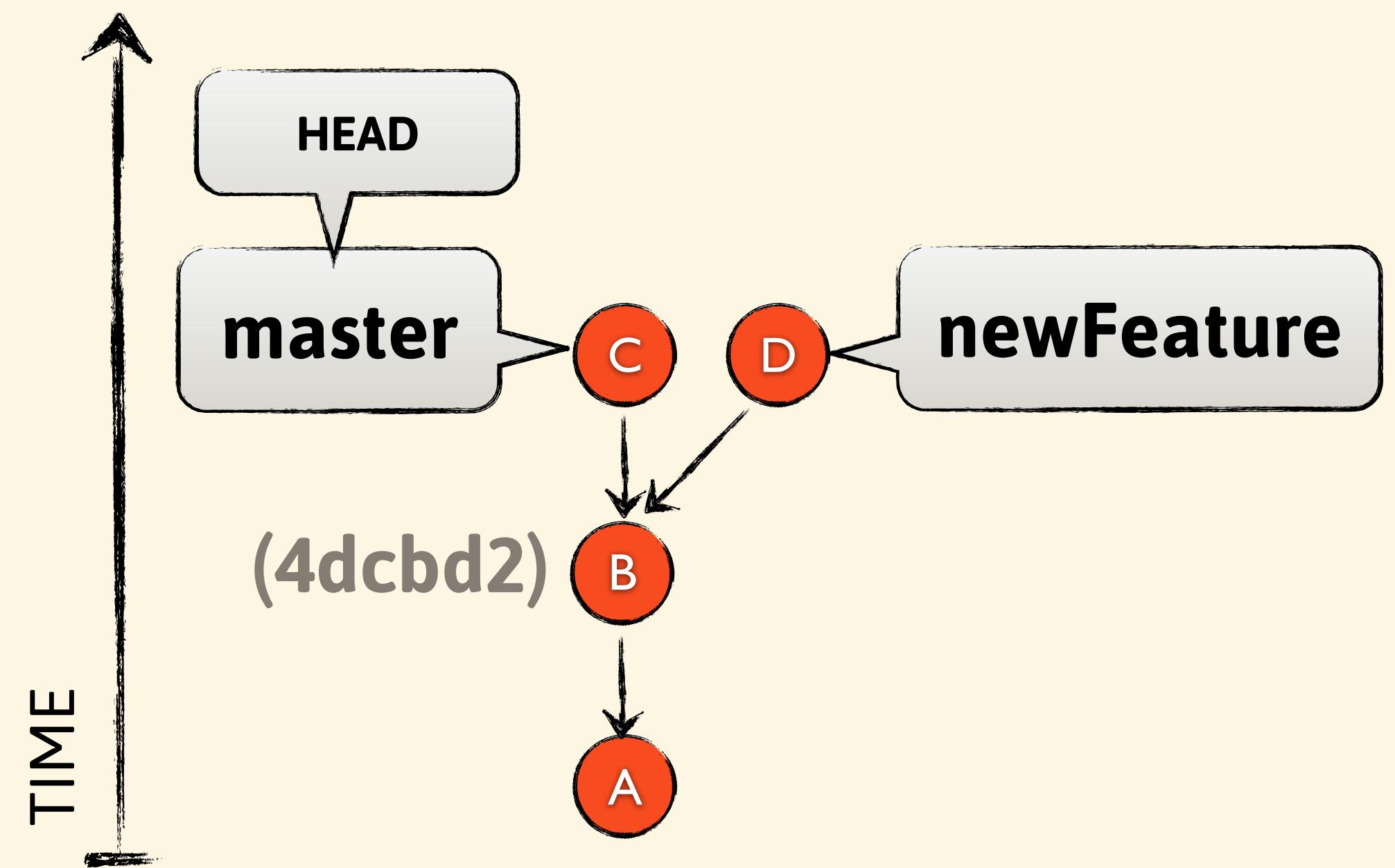
Head



(master) > git switch newFeature

(newFeature) > git switch master

Head

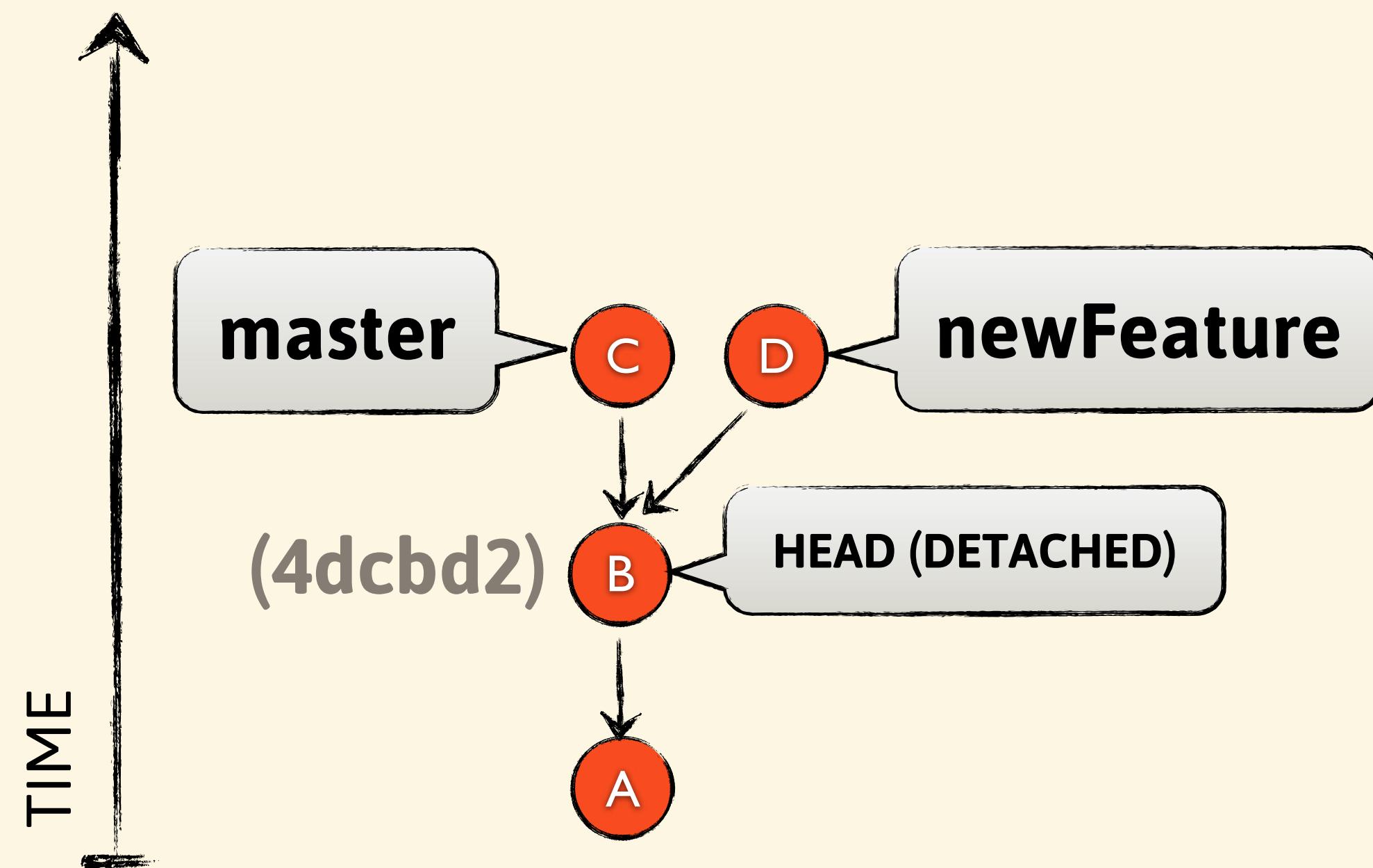


(master) > git switch newFeature

(newFeature) > git switch master

(master) > git checkout 4dcbd2

Head



(master) > git switch newFeature

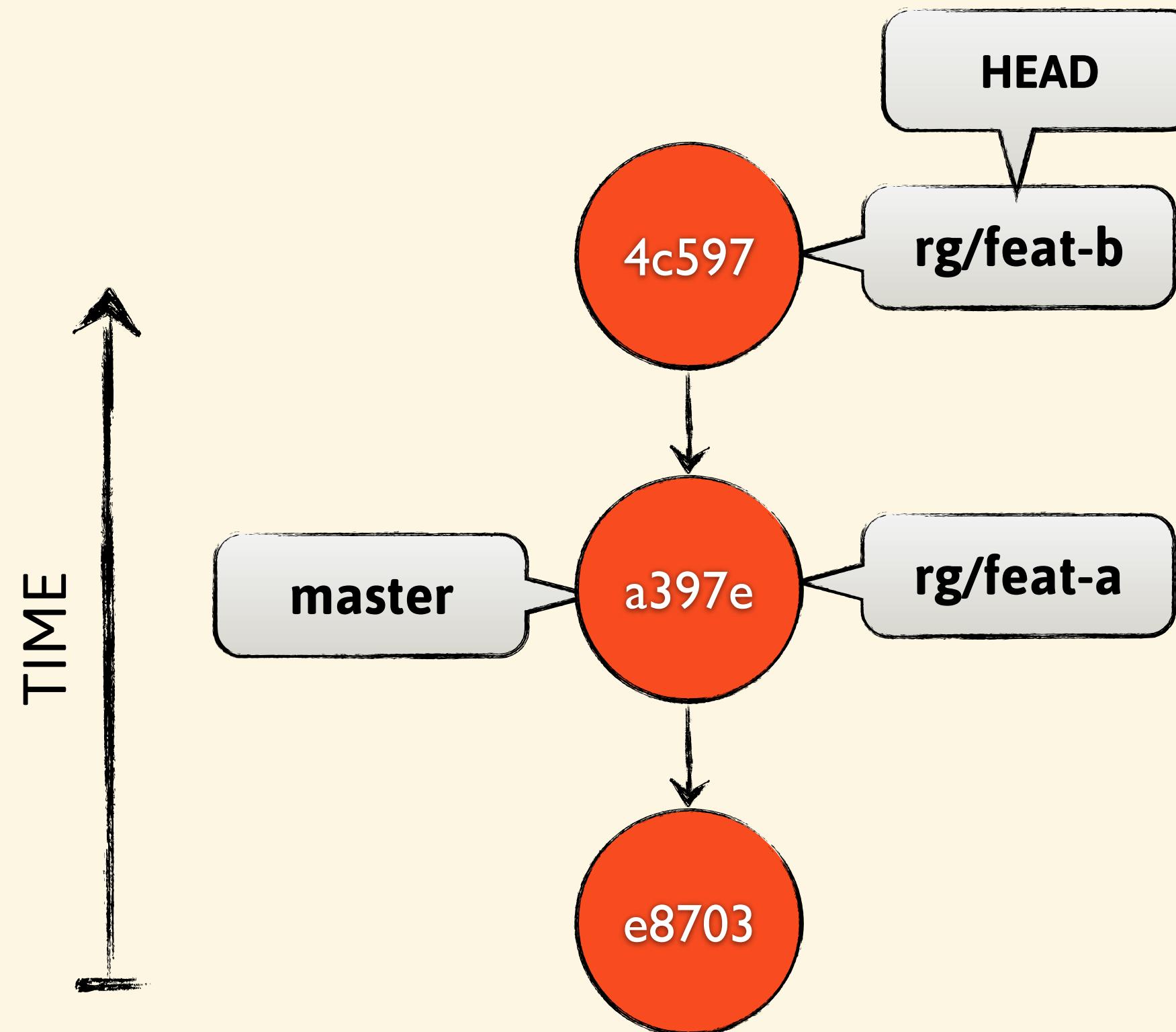
(newFeature) > git switch master

(master) > git checkout 4dcbd2

?

Quiz Time!

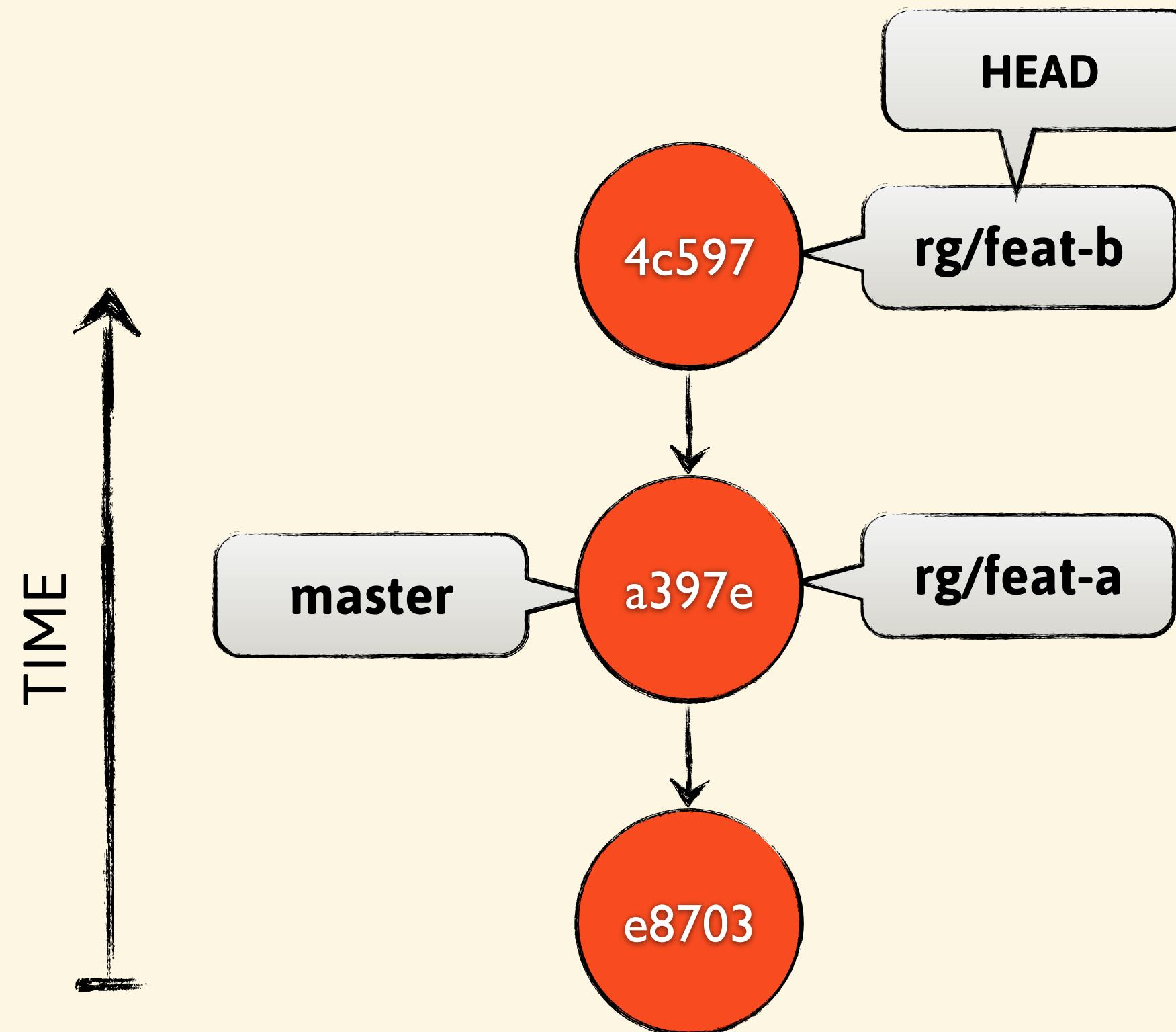
Given this DAG what will be the contents of .git/HEAD file?

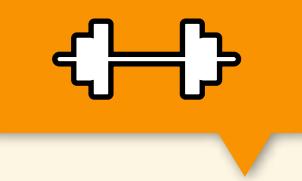


?

Quiz Time!

Given this DAG what will change when you run "git switch rg/feat-a"?





Exercise

- Use "git switch" to switch between branches in your repository and keep interrogating ".git/HEAD" to see what it does
- Find a random commit using "git log" and then "git checkout <commit-sha>". Carefully read the helpful message Git provides
- Create a new branch (using "git switch -c <branch-name>") from the checked out commit
 - Then make a (few) commits on that branch
 - See if you can draw out what your DAG looks like right now
 - What will be the contents of the HEAD file after all this?



TAGS

Remember me forever

Tags

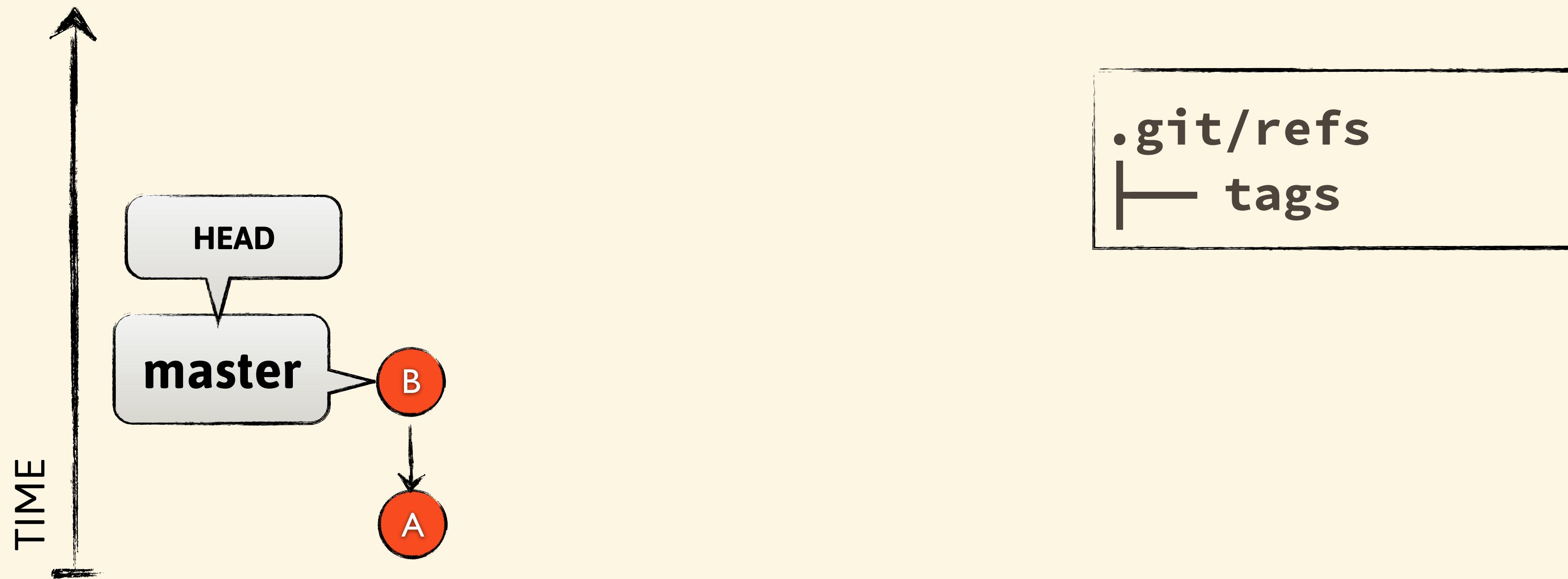
Points to a **particular commit**

Tags

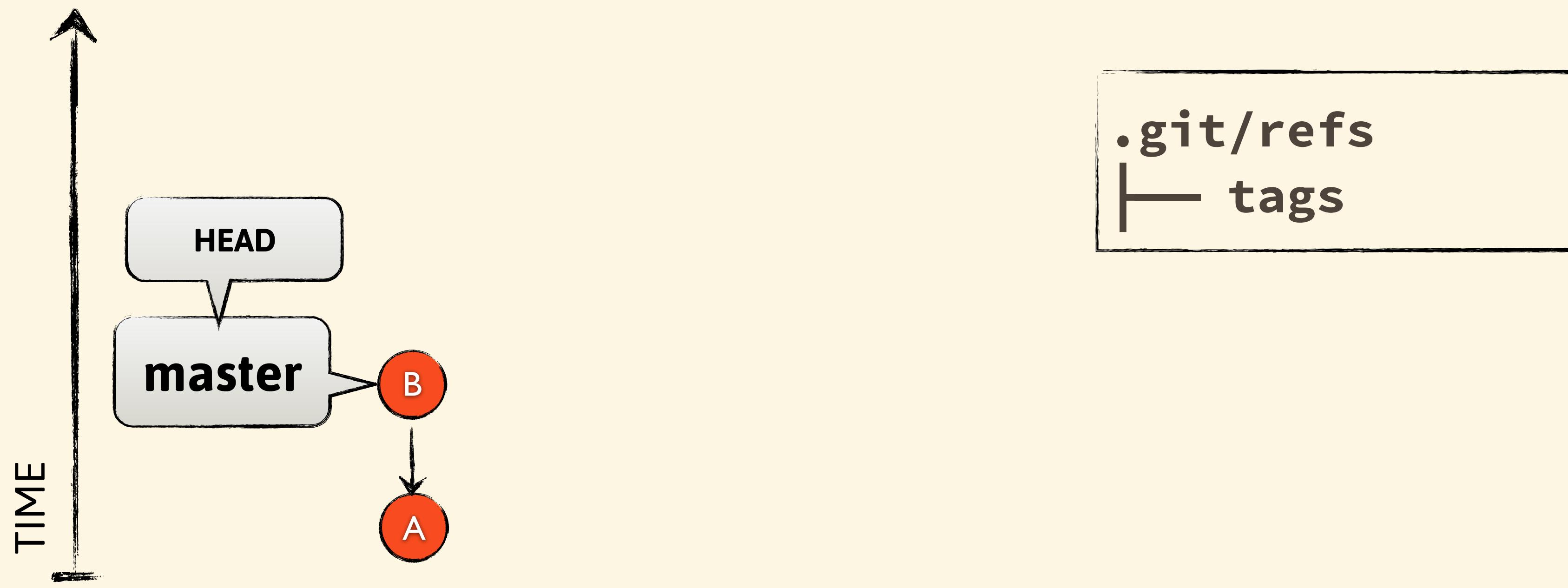
Never moves*

You can "force" a tag to be moved using "-f"

Head



Head



(master) > git tag v1.0.0

Head



(master) > git tag V1.0.0

Head



(master) > git tag V1.0.0

Head



(master) > git tag V1.0.0

(newFeature) > git commit ...

Head



(master) > git tag V1.0.0

(newFeature) > git commit ...

Head



```
(master) > git tag V1.0.0
```

```
(newFeature) > git commit ...
```

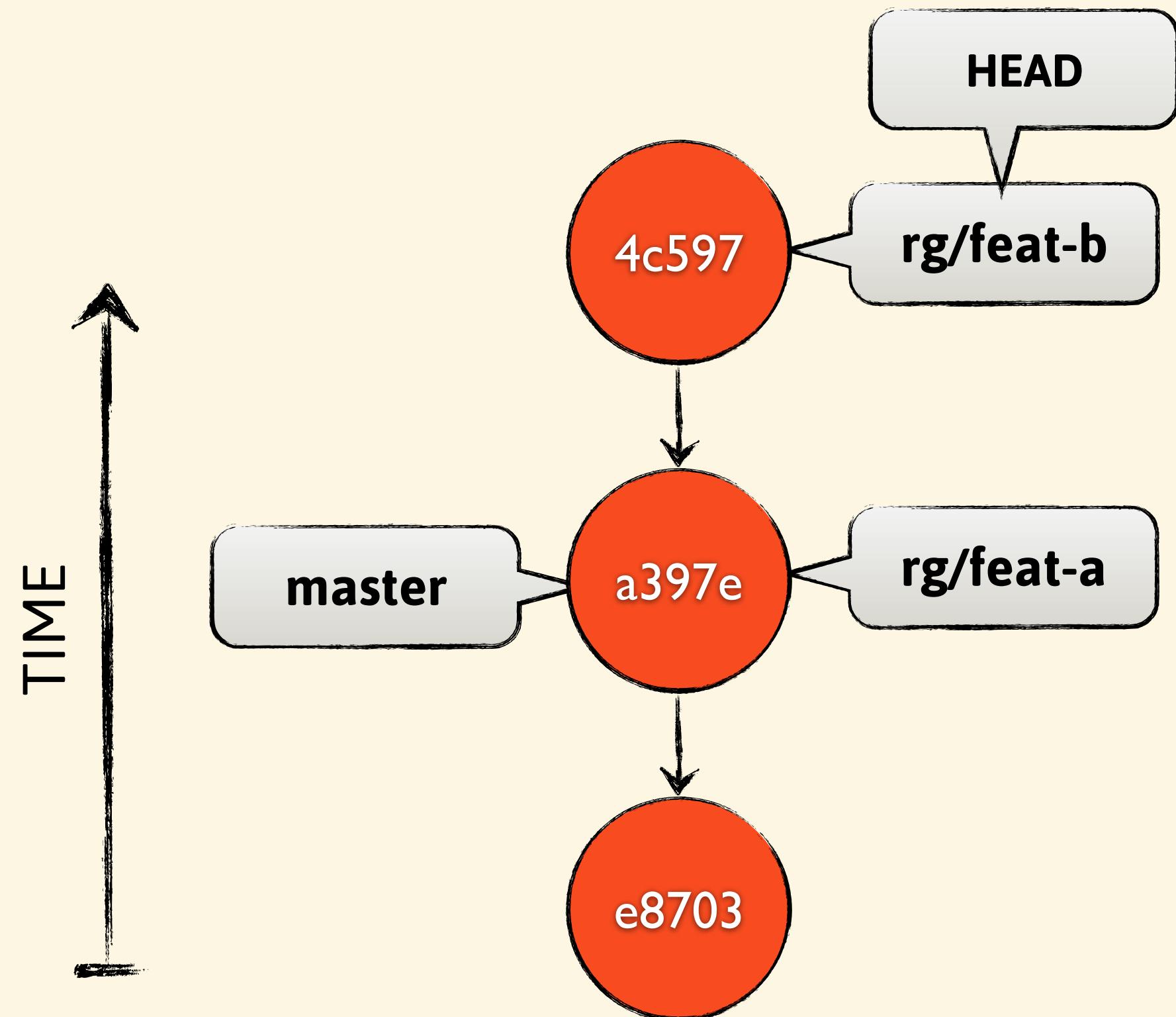
?

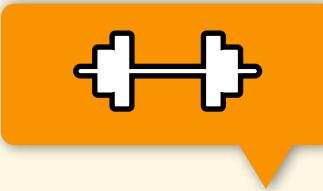
Quiz Time!

Suppose your DAG looks like this.
For each step listed at the bottom:

- What files will be created and what will be their contents?
- What will the DAG look like after each execution.

- "git tag v3"
- "git switch master" and "git tag v1"
- "git tag v2 rg/feat-a"





Exercise

- Use "git tag <some-tag>" to tag the HEAD commit. Then inspect ".git/refs/tags" to find the file and inspect it's comments
- Find a random commit in your graph using "git --no-pager log --oneline" and then use the "git tag <tag-id> <commit-id>" to create a new tag. View your log again with "git --no-pager log --oneline"



Reflog

Know where you have been

Reflog

List of branch tips

Updated every time **HEAD** moves

Reflog

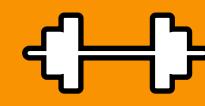
Your Safety Net*

Reflog

```
> git reflog
e837f19 (HEAD -> master) HEAD@{0}: checkout: moving from rg/add-test to master
b74c8ef (rg/add-test) HEAD@{1}: commit: test: add test file
e837f19 (HEAD -> master) HEAD@{2}: checkout: moving from master to rg/add-test
e837f19 (HEAD -> master) HEAD@{3}: commit: feat: add HelloWorld implementation
ad6af2a HEAD@{4}: commit (initial): docs: introduce a README
```

KEEP CALM USE
AND REFLOG

<http://www.keepcalm-o-matic.co.uk/p/keep-calm-and-use-git-reflog/>



Exercise

- Use "git --no-pager reflog" to inspect your activity
- Perform more than one edit/commits and inspect reflow again
- Use "git switch" to switch branches
- Delete the other branch using "git branch -d <branch-name>"
- Try recreating the deleted branch by using the reflog

Stash

Pseudo-Commits

hack hack hack

hack hack hack

Oh No! Production Bug

hack hack hack

Oh No! Production Bug

git status

hack hack hack

Oh No! Production Bug

git status

```
# On branch getCommits
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   src/com/looselytyped/galore/bla.clj
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:  .gitignore
#       modified:  src/com/looselytyped/galore/commit.clj
#       modified:  src/com/looselytyped/galore/fs.clj
#
```