Advanced Git Workshop

RAJU GANDHI

@LOOSELYTYPED

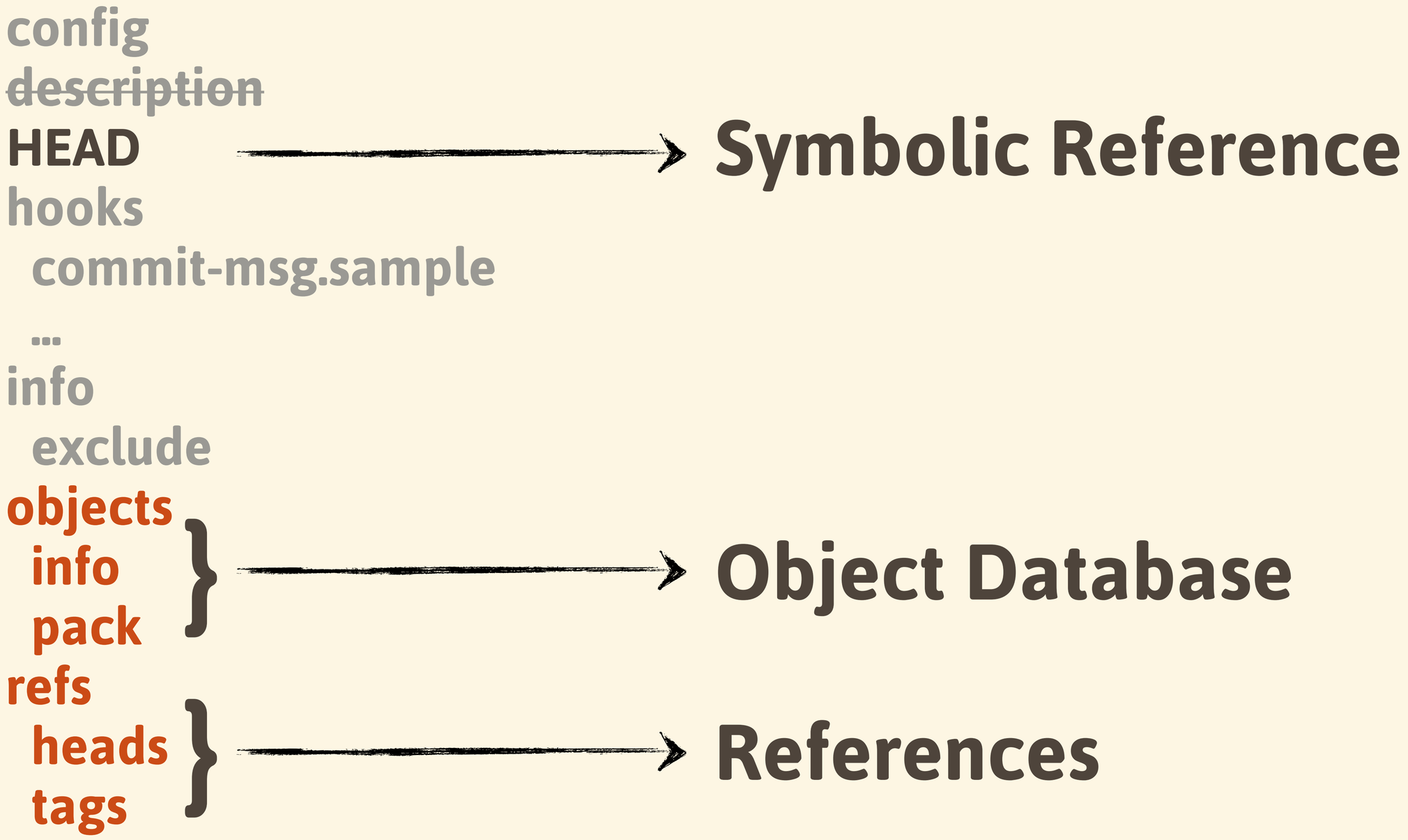
FOUNDER - DEFMACRO SOFTWARE

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

IMMUTABLE!

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!

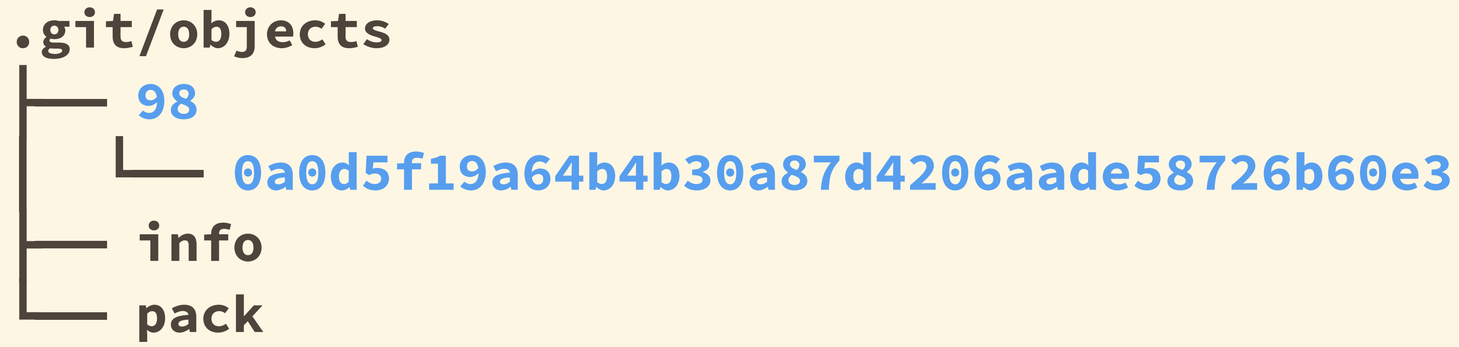
SHA-1 hash

`980a0d5f19a64b4b30a87d4206aade58726b60e3`

SHA-1

Blob

980a0d5f19a64b4b30a87d4206aade58726b60e3



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

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!

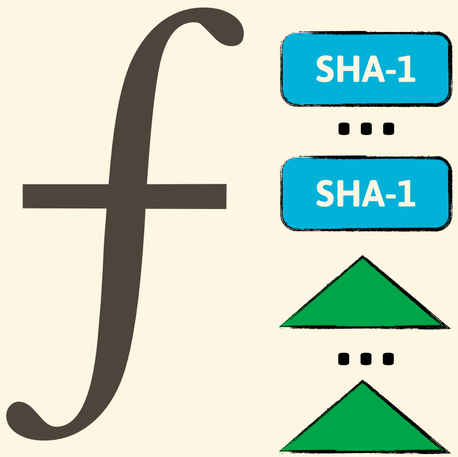
Suppose you have the following directory structure.

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

feat: initial implementation of Hello world
```

State of the working dir

(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 metadata

(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
```

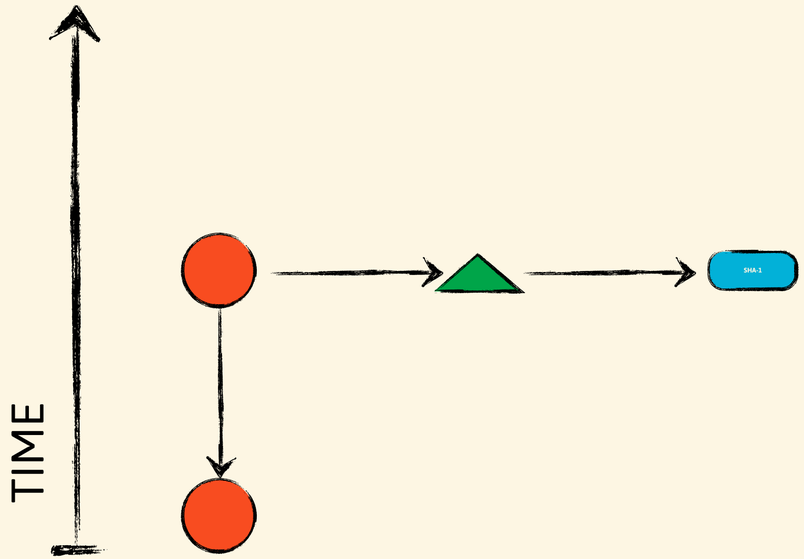
SHA-1 hash

```
ff4d71c72a91378206f6ce04e9153aeb02316737
```

Commit

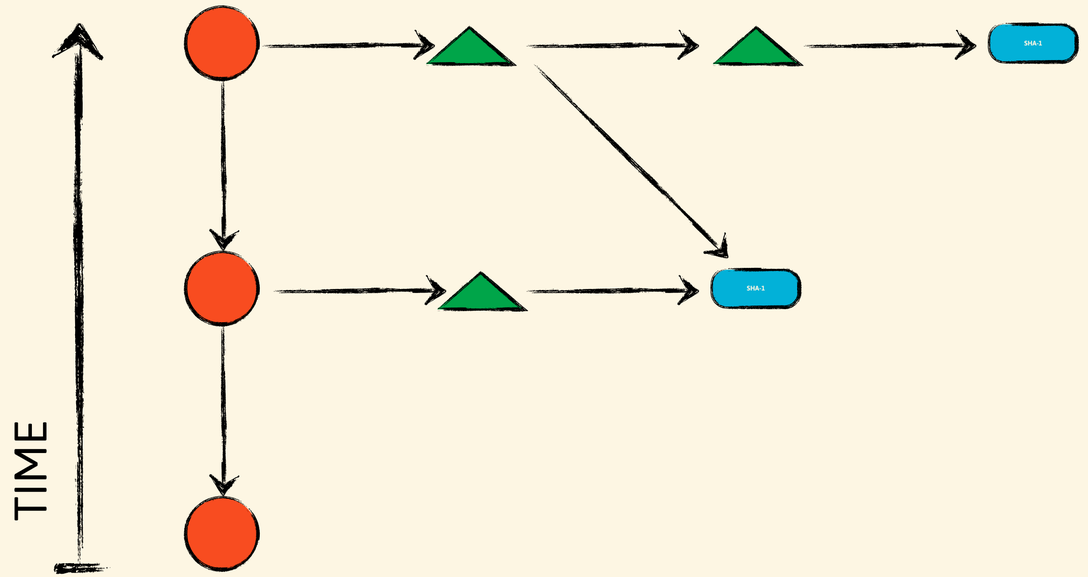
ff4d71c72a91378206f6ce04e9153aeb02316737

Commit



Git commits the entire tree per commit

Say Whattt???




```
(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

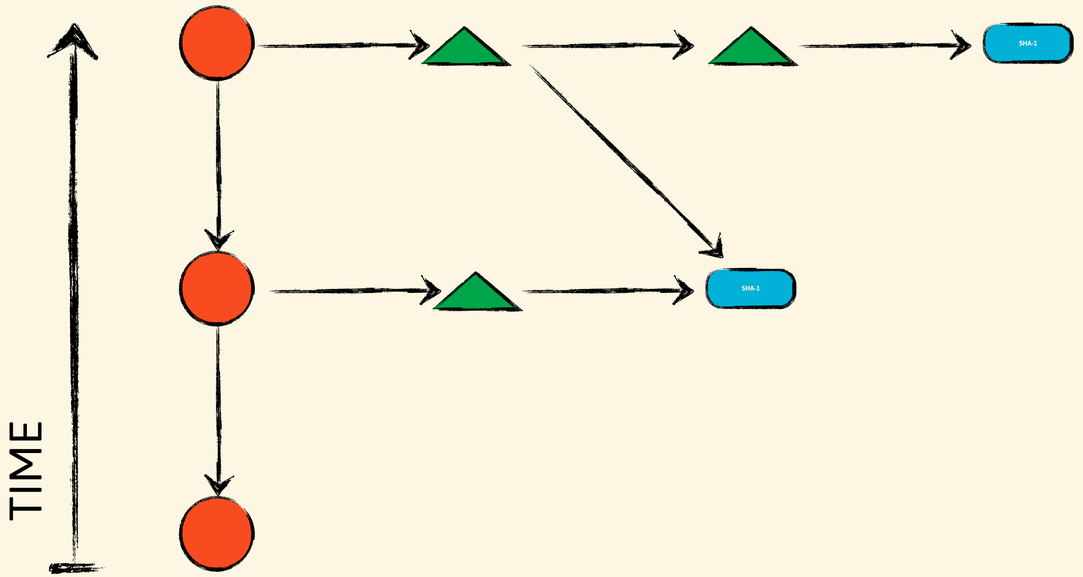
Exercise

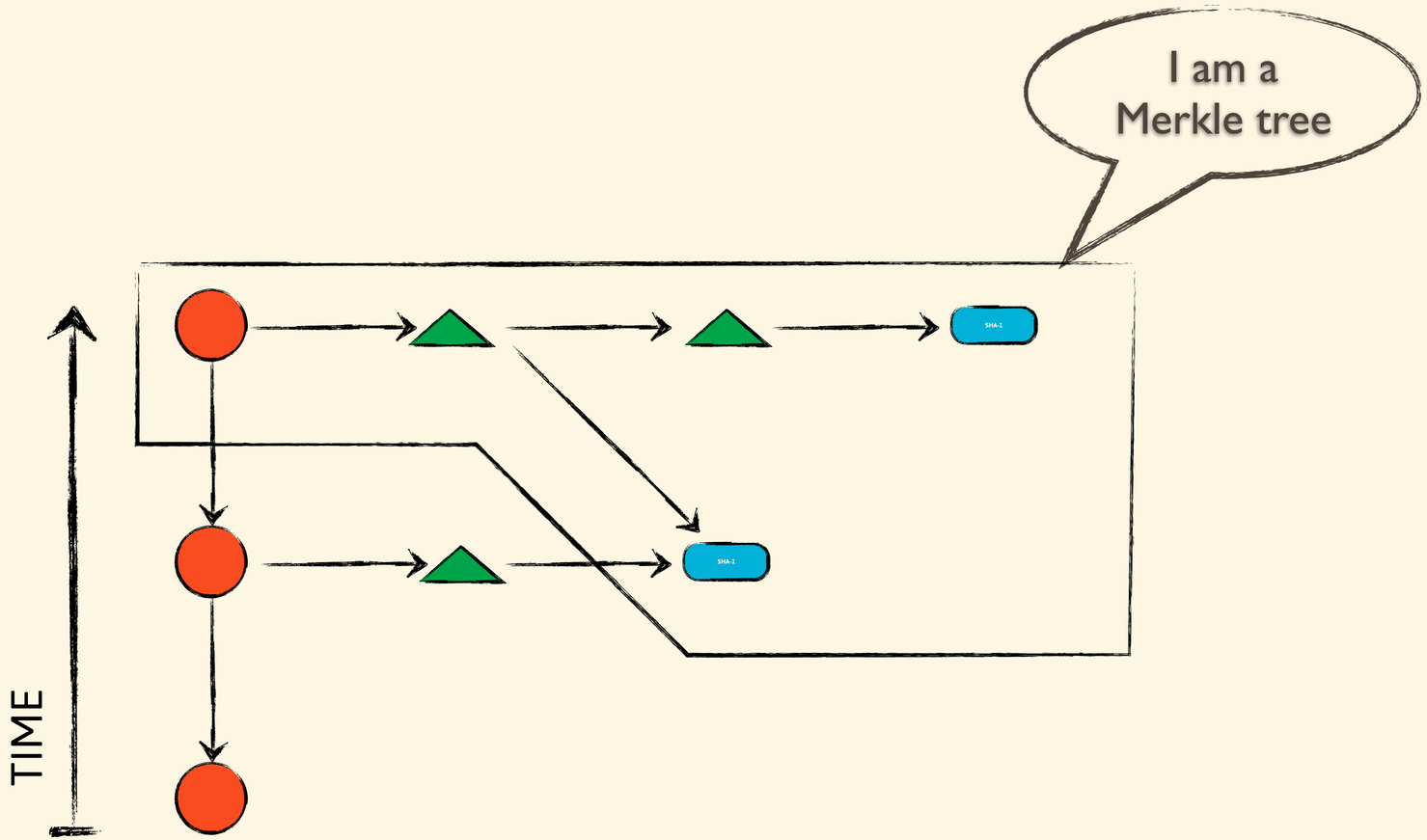
- Create a bunch of commits in your repository (Be sure to add several directories and subdirectories containing files)
- Inspect one or more commits using "git cat-file -p <commit-sha>" (You can use git log to find the ID of a commit)
- Use "git show <commit-sha>" and see what information it displays

(Most) Commits

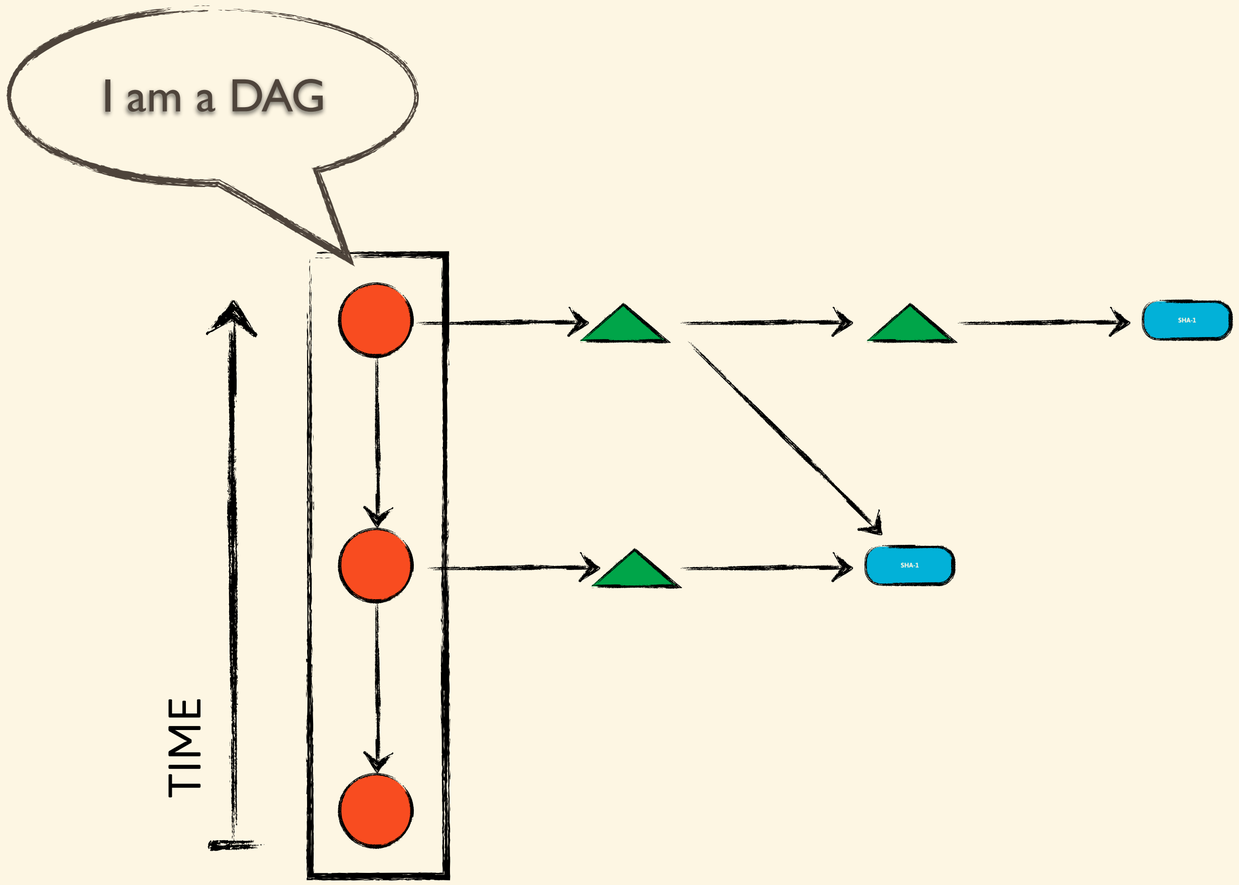
```
tree e6dab7bf1353cc6abf8f0f79811f26ff7da9c0e9
parent 08de1a6e2bc04f1758ea50799206cb26c55deb98
author Raju <raju.gandhi@gmail.com> 1587841782 -0400
committer Raju <raju.gandhi@gmail.com> 1587841782 -0400

test: add a test for Hello World
```

State of the working dir

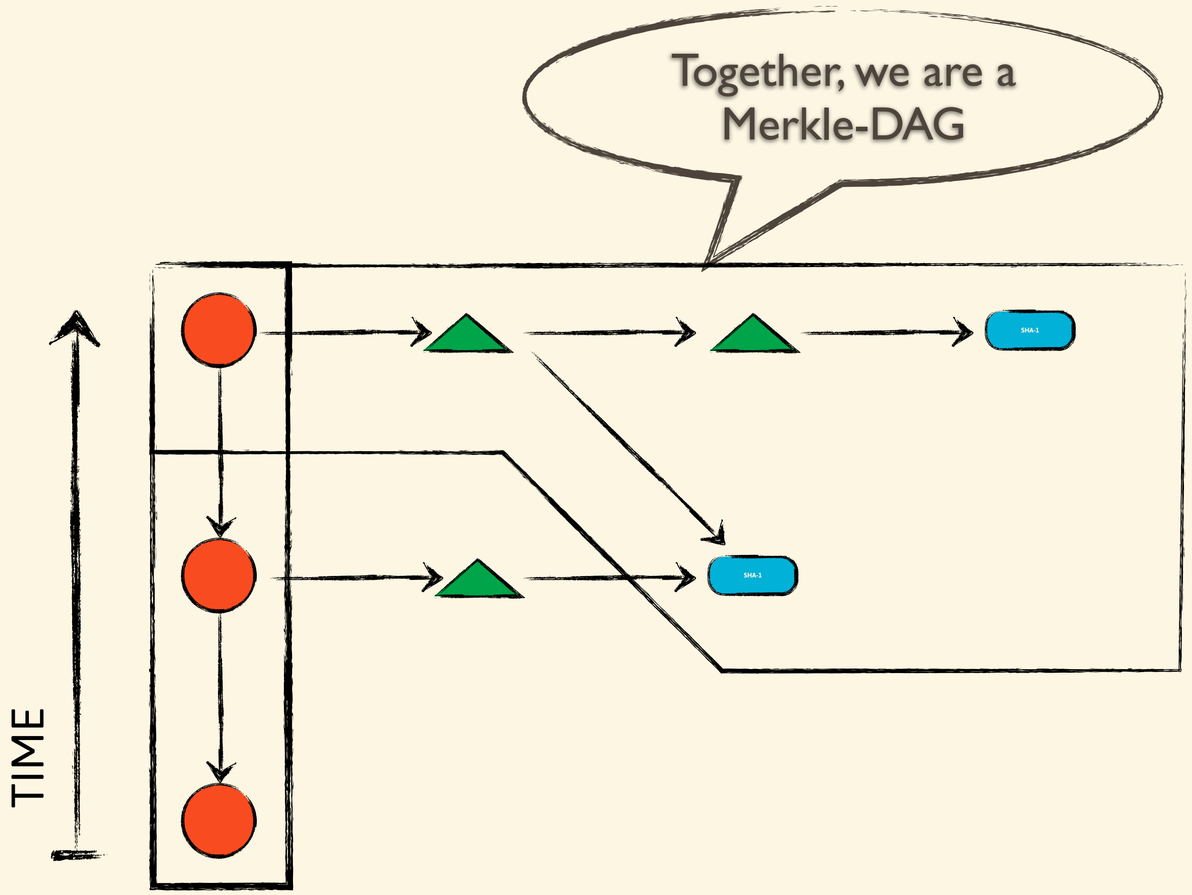

TIME
SHA-1
SHA-1



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*

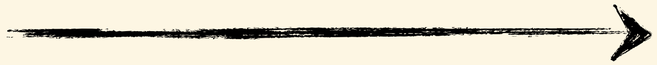


DAG — A finite directed graph with no directed cycles

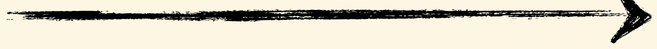

Together, we are a
Merkle-DAG
SHA-1
SHA-1
TIME

Branches / HEAD / Tags

config
~~description~~
HEAD
hooks
  commit-msg.sample
  ...
info
  exclude
objects
  info
  pack
refs
  heads
  tags



Symbolic Reference



References

Branches

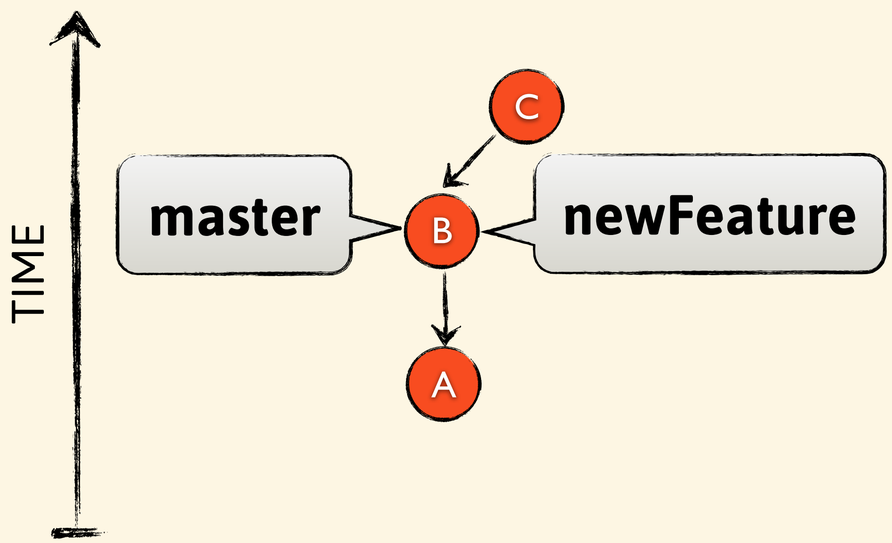
THE GIT WAY

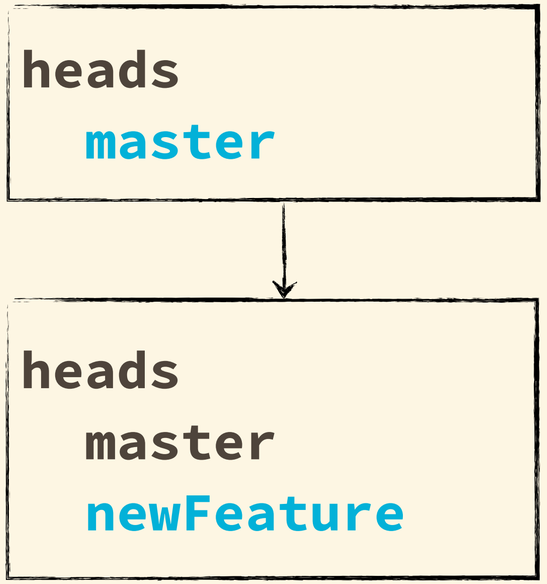
Image by 💛 Passt gut auf euch auf und bleibt gesund! 💛

Branches

A named reference to a commit

Branches



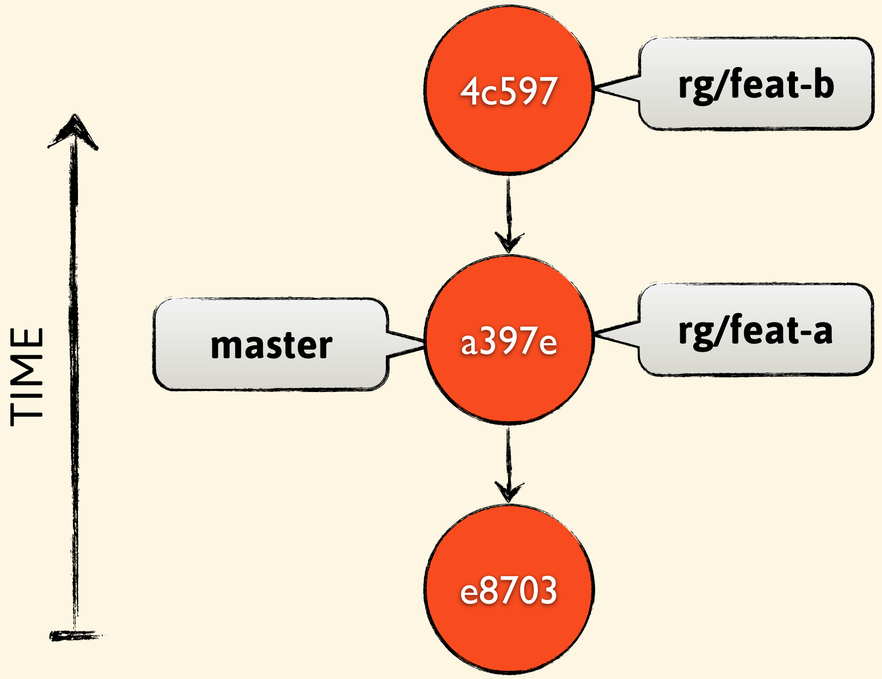



```
(master) > git switch -c newFeature
(newFeature) > (cd .git/refs/ && tree)
(newFeature) > git commit -m "Some Commit"
```

?

Quiz Time!

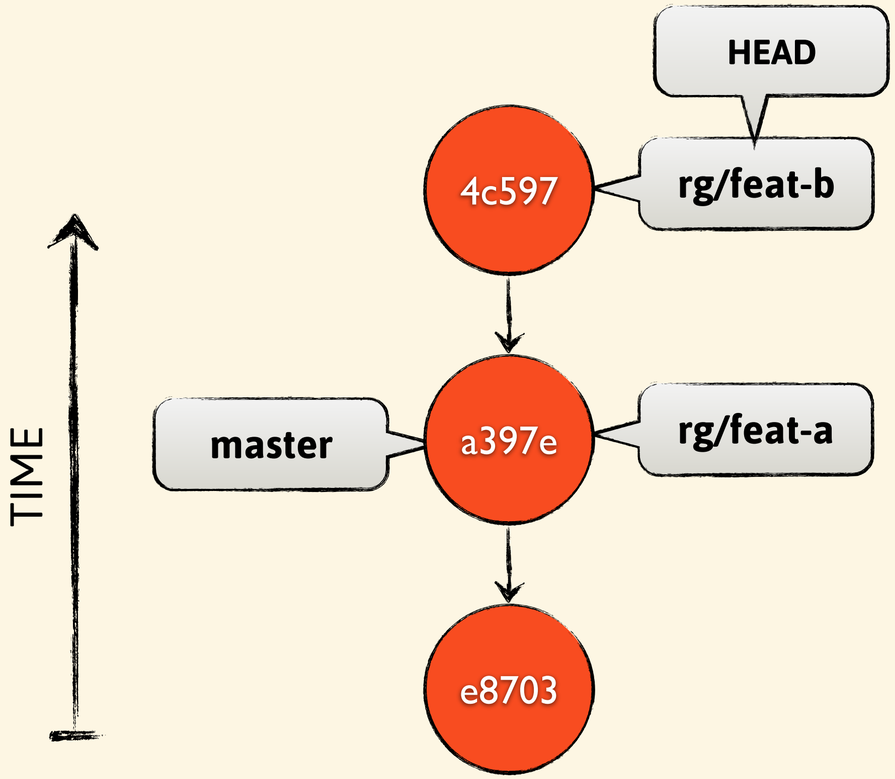
Given this DAG how many refs will you find under ".git/refs/heads" and what will be there respective contents?



?

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!!!

Photo by Tanner Mardis

Head

Points to the checked out commit

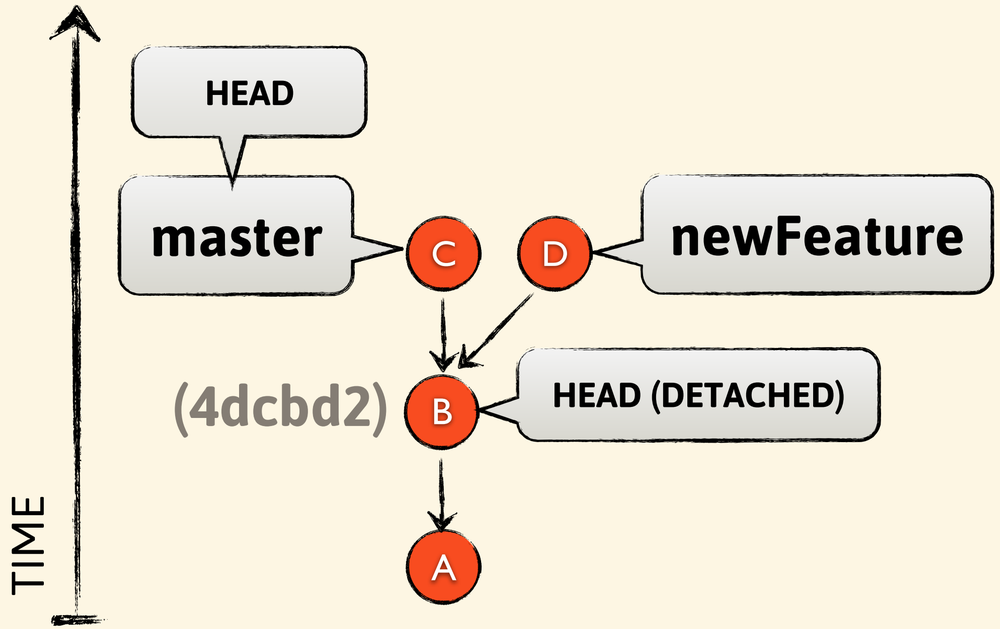
Head

The parent of the next commit

Head

Usually **the last commit on that branch**

Head

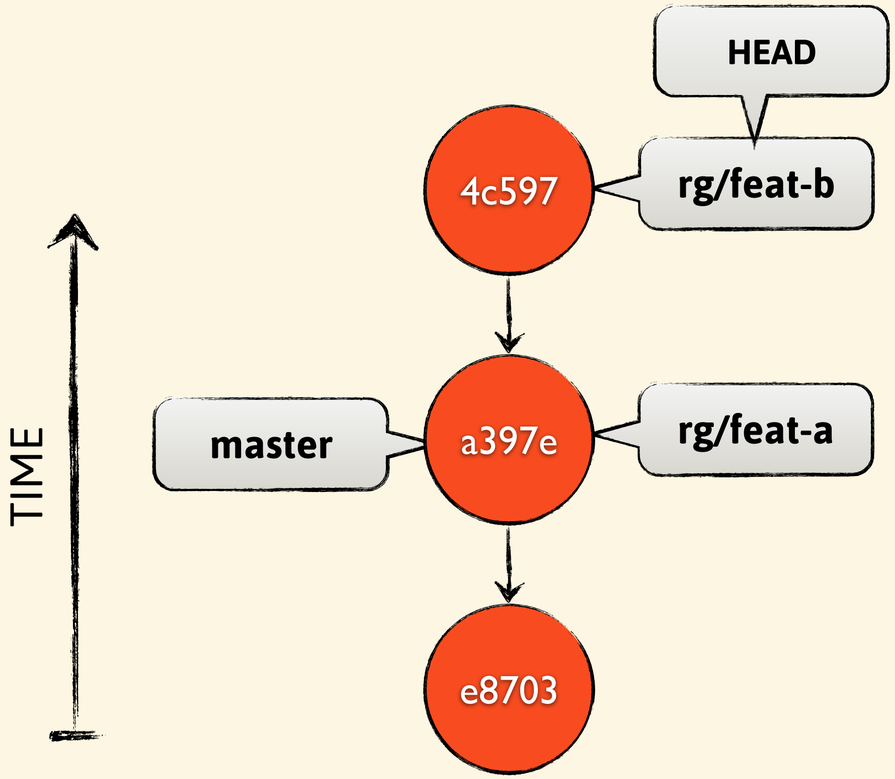



```
(master) > git switch newFeature
(newFeature) > git switch master
(master) > git checkout 4dcbd2
```

?

Quiz Time!

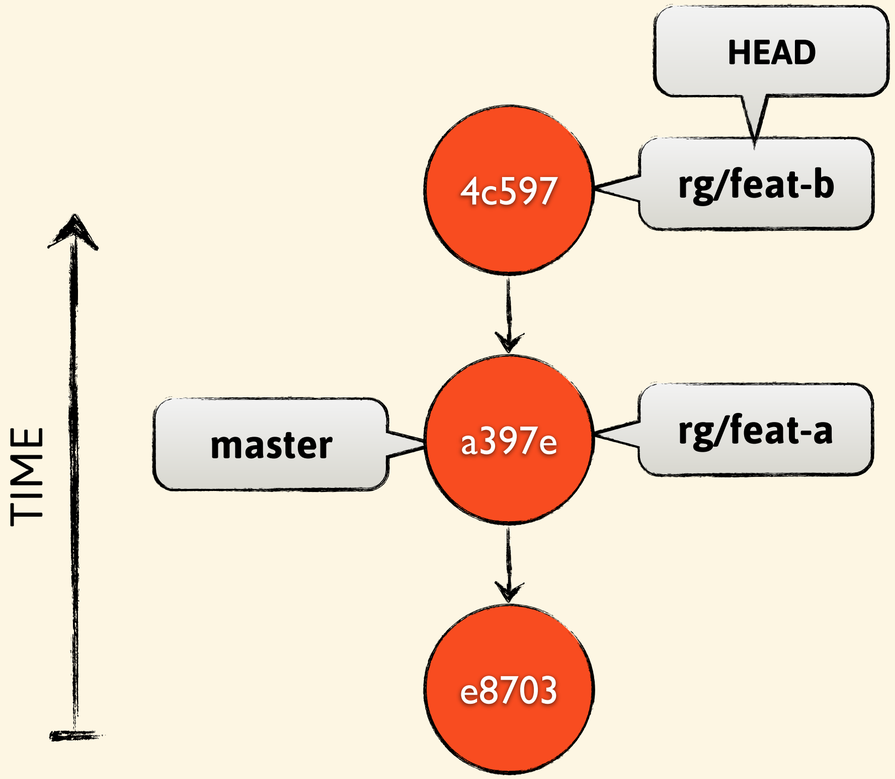
Given this DAG what will be the contents of .git/HEAD file?



?

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

Photo by Brett Jordan

Tags

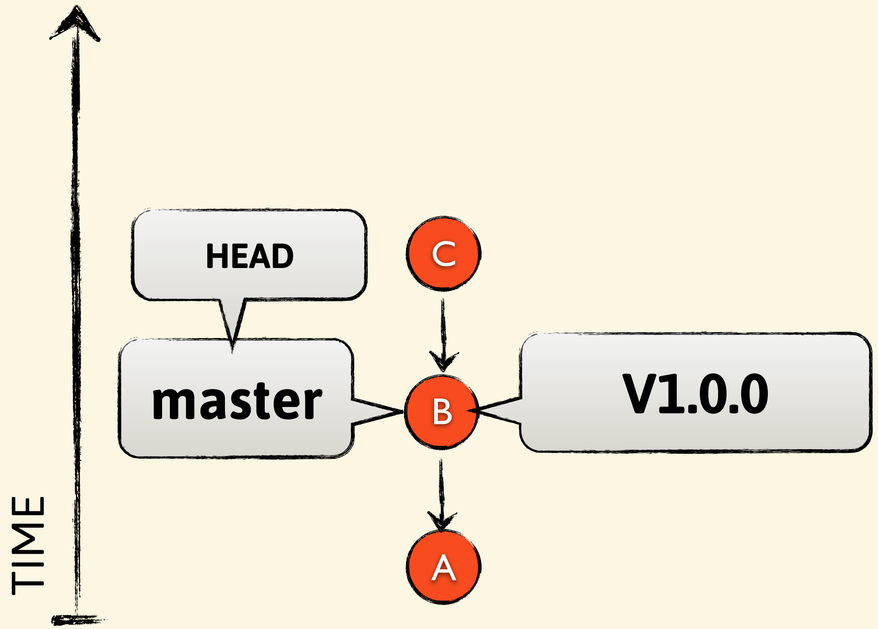
Points to a particular commit

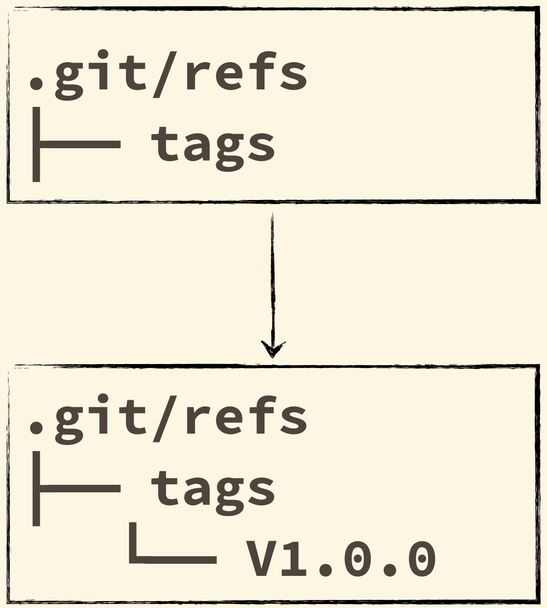
Tags

Never moves*

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

Head






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

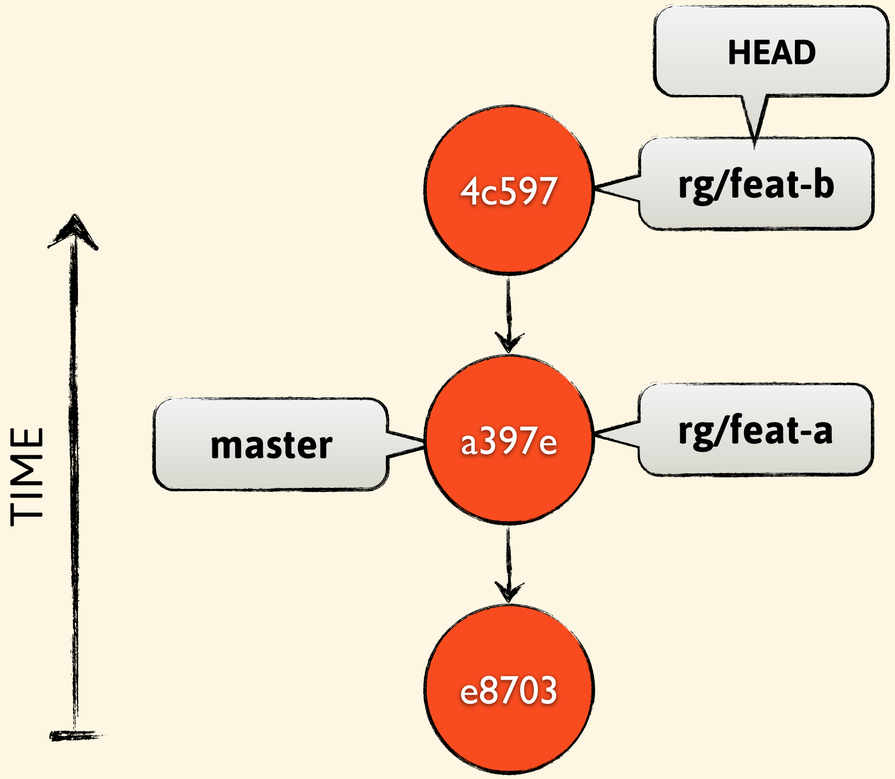
?

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*

gc.reflogExpire defaults to 90 days

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

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
#
```


Gah!!!

KEEP CALM USE AND STASH

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

```
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
#
```

```
git stash
```

```
git status
```

```
# On branch master
nothing to commit, working directory clean
```


Yay!!!

Stash

Stores the index and working tree state

Resets to the current commit!

Stash

Is just like a commit

Does not participate in the log

Stash

```
git stash

git stash save <message>
```

Stash

```
git stash list

git stash apply/pop
```

Quiz Time!

This is your "git status"

```
› git status
On branch master
Changes to be committed:
        new file:   src/HelloWorld.spec.js
Changes not staged for commit:
        modified:   src/HelloWorld.js
Untracked files:
        src/lib.js
```

- [ ] What will "git stash save 'some message'" do? What will "git status" report after stashing?

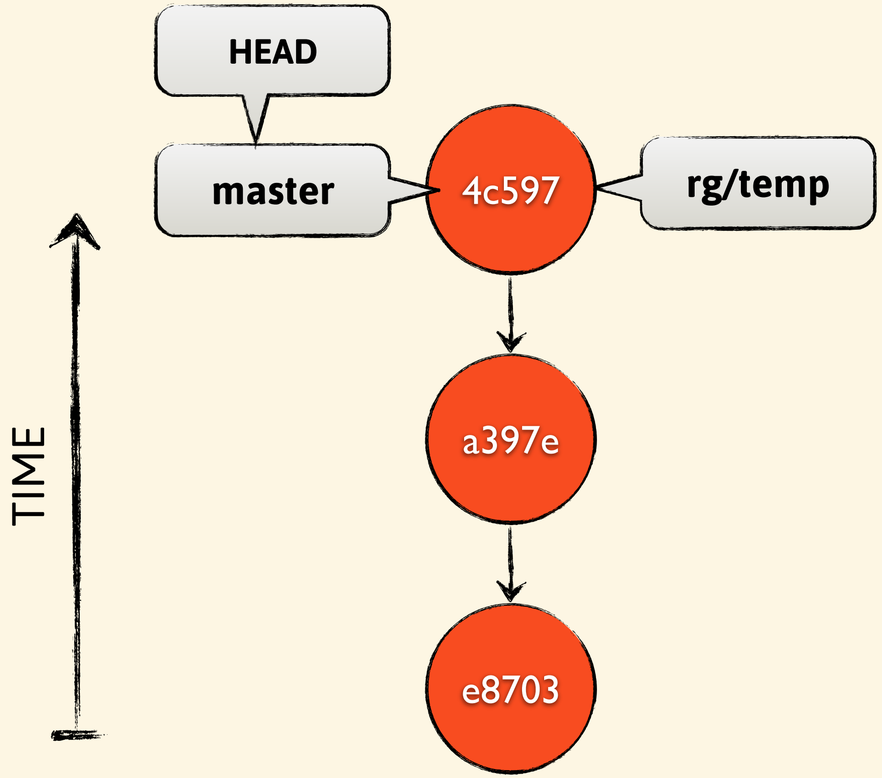
Exercise

- Edit some committed files, add a few new files, and add some of the new files to the index
- Stash your work using "git stash save 'Some message'" — Check "git status"
- Perform a "git stash apply" followed by a "git stash list"
- Then perform a "git stash pop" followed by a "git stash list" - What's the difference?

Quiz Time!

- [ ] Can you think of a better workflow than using stashes?

A better workflow




```
(master) > git branch rg/temp
(master) > git reset HEAD~1
```

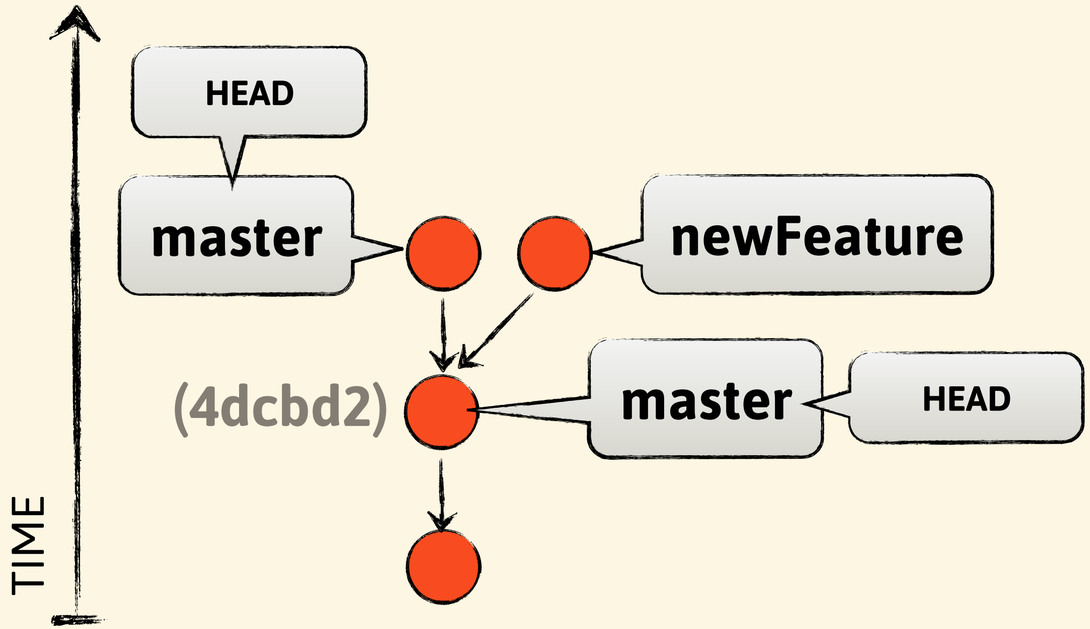
Undo

SORTA …

Reset

Moves the HEAD and branch

Reset --soft




```
(master) > git reset --soft 4dcbd2
```

Quiz Time!

- [ ] Do you think it's prudent to use "git reset" on integration branches?

Exercise

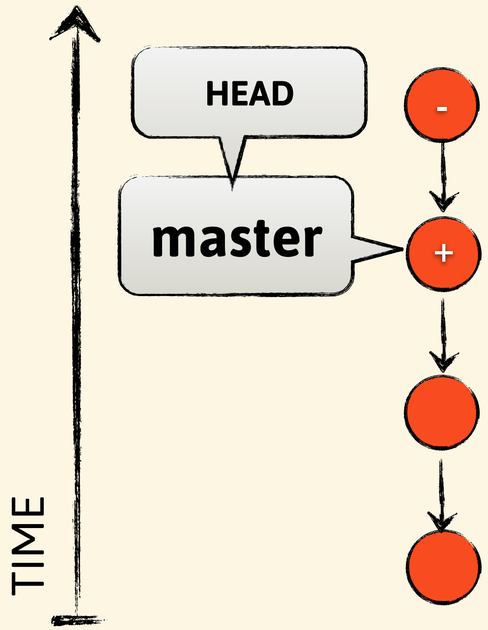
- Create two files in your repository — Let's call them A.txt, and B.txt
- Add A.txt to the index (Be sure to check your status here)
- Perform a "git reset HEAD~1"
  - Check "git status" — Explain what you see
- Use reflog to set your branch back to where it was (git reset <previous_sha>)
- Perform a "git reset --hard HEAD~1"
  - Check "git status" — Explain what you see

NEVER RESET PUBLIC COMMITS

Revert

Undos the commit and create a new commit

Revert




```
(master) > git revert HEAD
```

Exercise

- Perform a "git --no-pager log --oneline" and find a commit
- Perform a "git show <commit-sha>" to see what the delta was
- Perform a "git revert <commit-sha>" followed by a "git --no-pager show HEAD" to see if the delta is the opposite of the original commit

Merge

Happily, forever

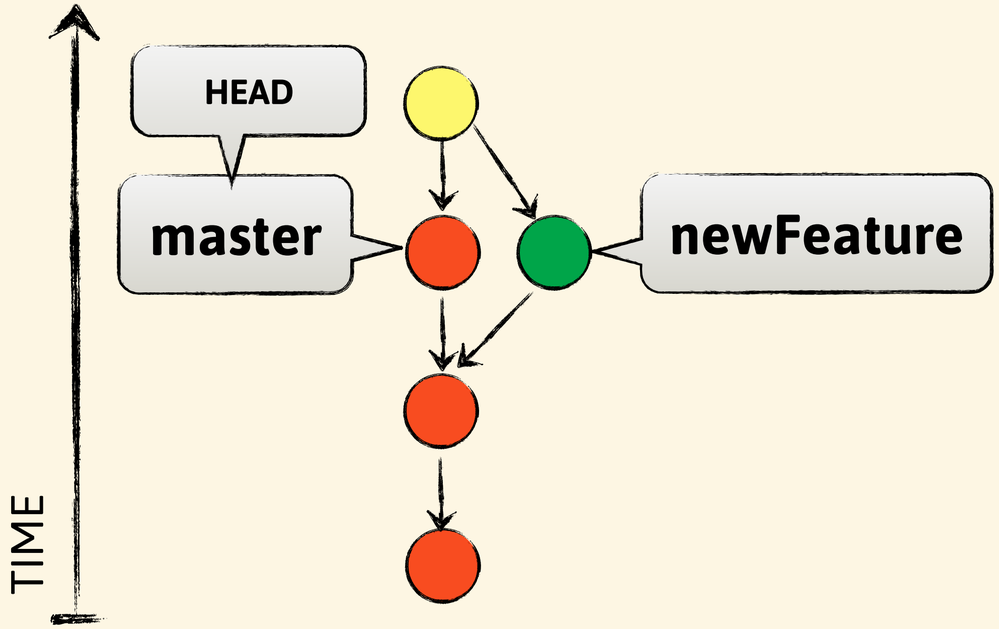
Merge

Joins two or more commits

Merge

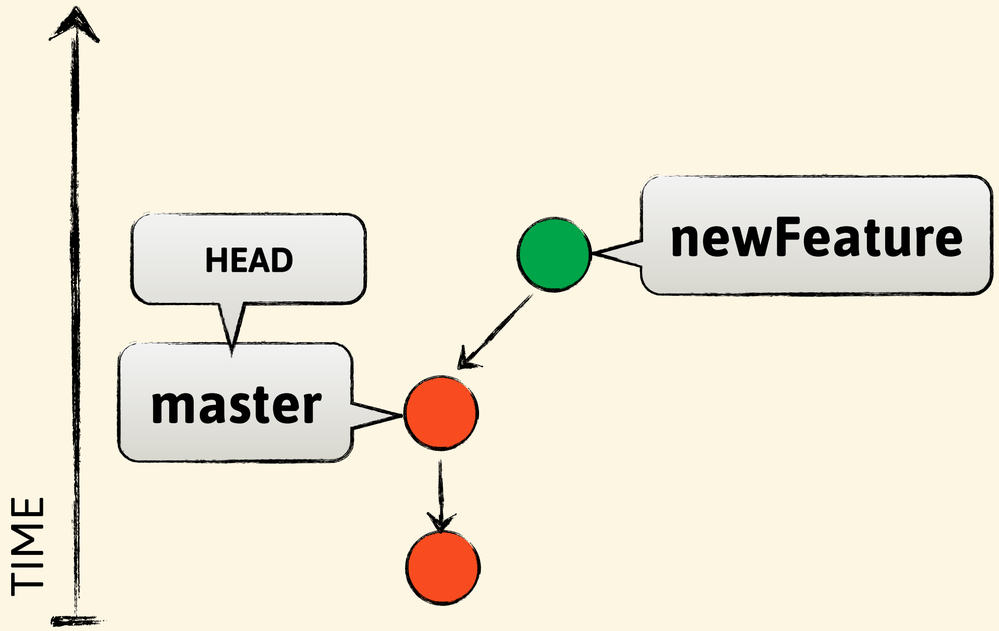
(May) Create a child commit

Merge




```
(master) > git merge newFeature
```

(Fast-forward) Merge




```
(master) > git merge newFeature
```

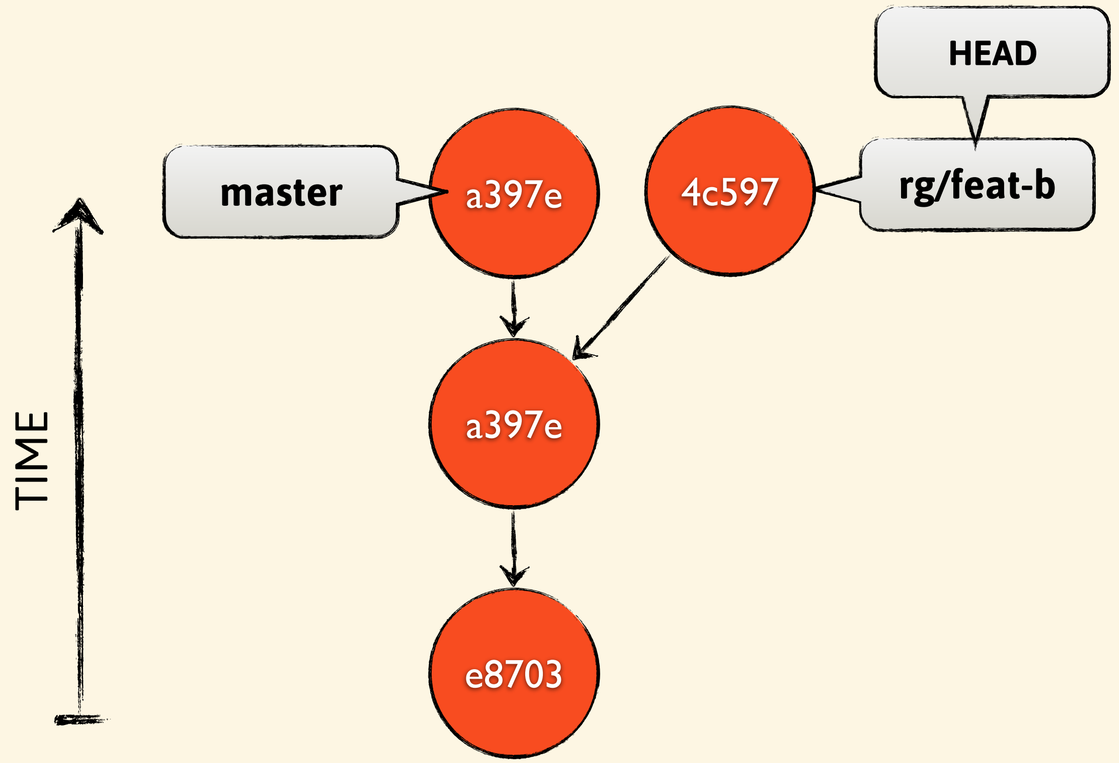
USE MERGE FOR PUBLIC HISTORY

?

Quiz Time!

Suppose your DAG looks like this. Will "git merge master" be a fast-forward merge? Explain your answer.

- ☐ Yes
- ☐ No



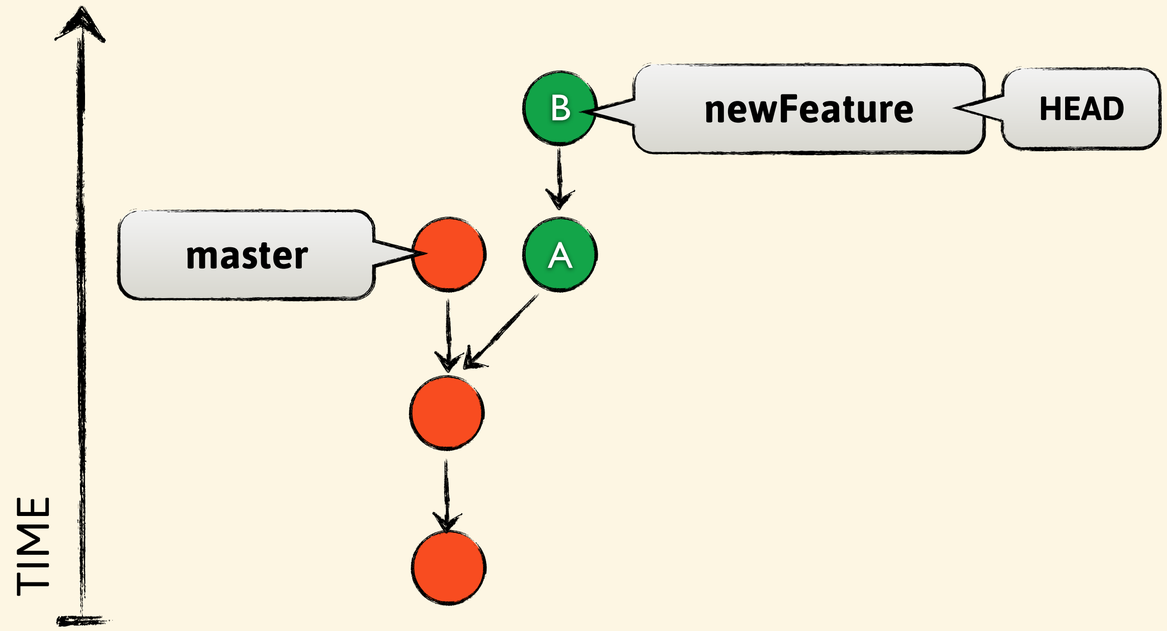
Rebase

The swiss army knife

Rebase

Relocates a branch to a new parent

Rebase

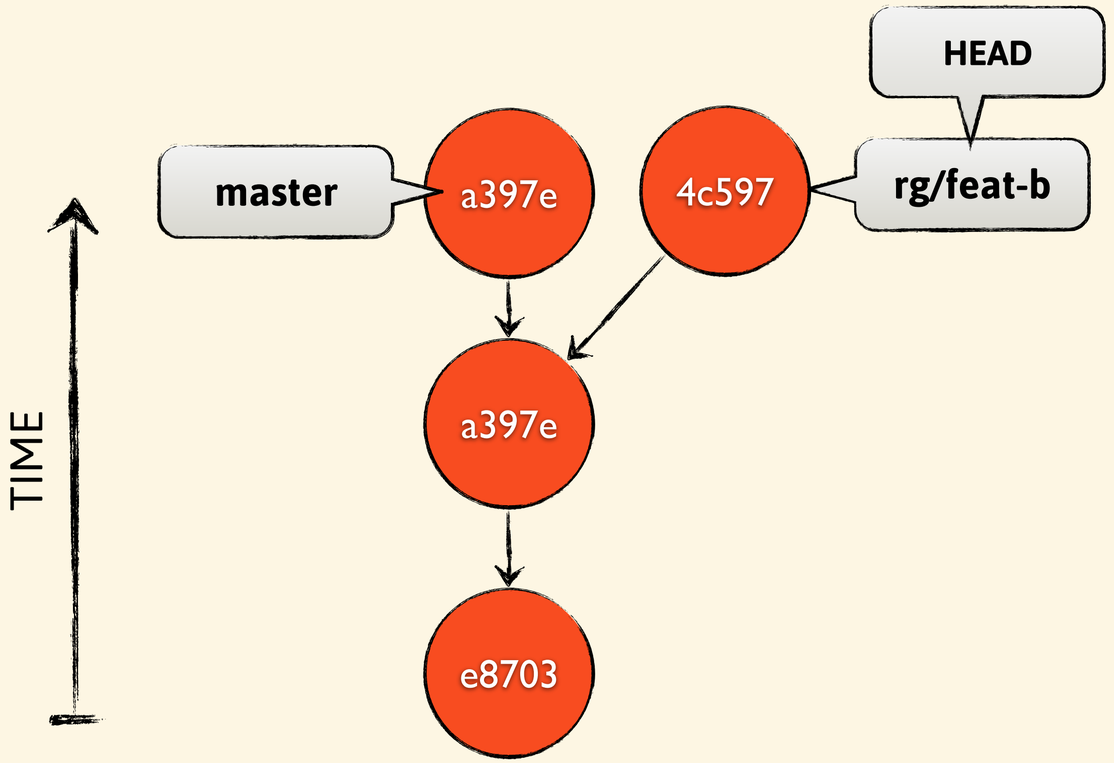



```
(newFeature) > git rebase master
```

USE **REBASE** FOR **PRIVATE COMMITS**

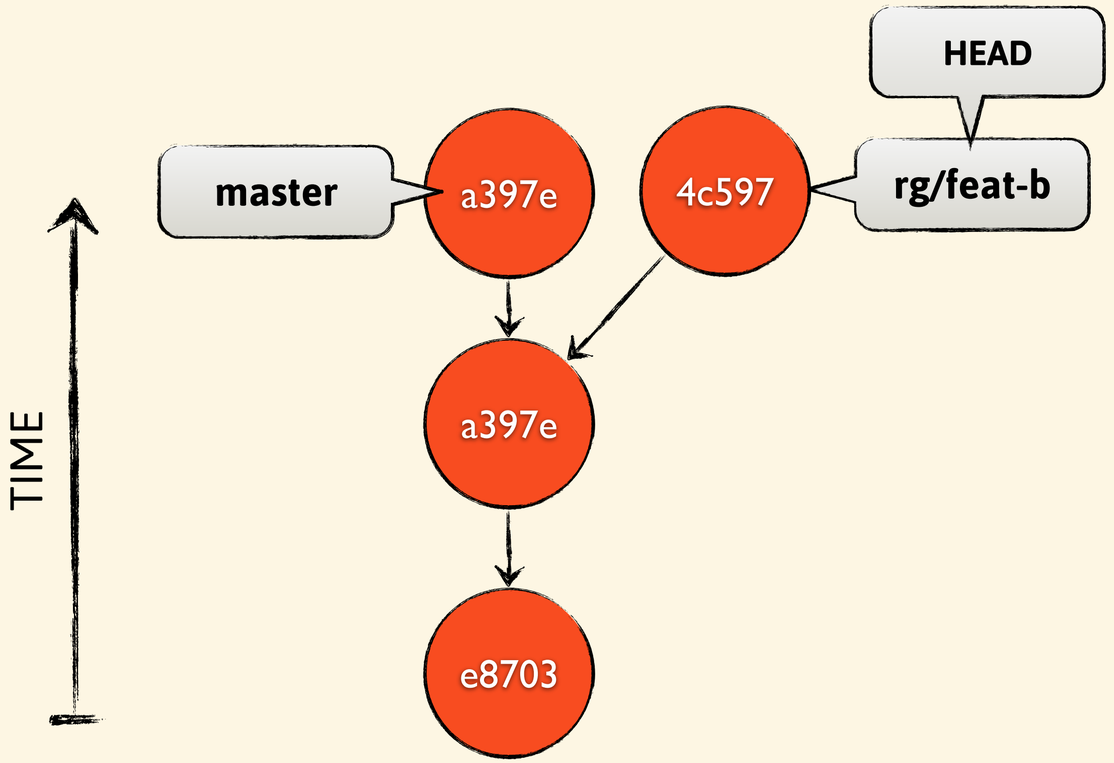
Exercise

Make it so that your DAG looks like this. Then perform a "git rebase master". Observe "git --no-pager log --oneline -n 10"



Exercise

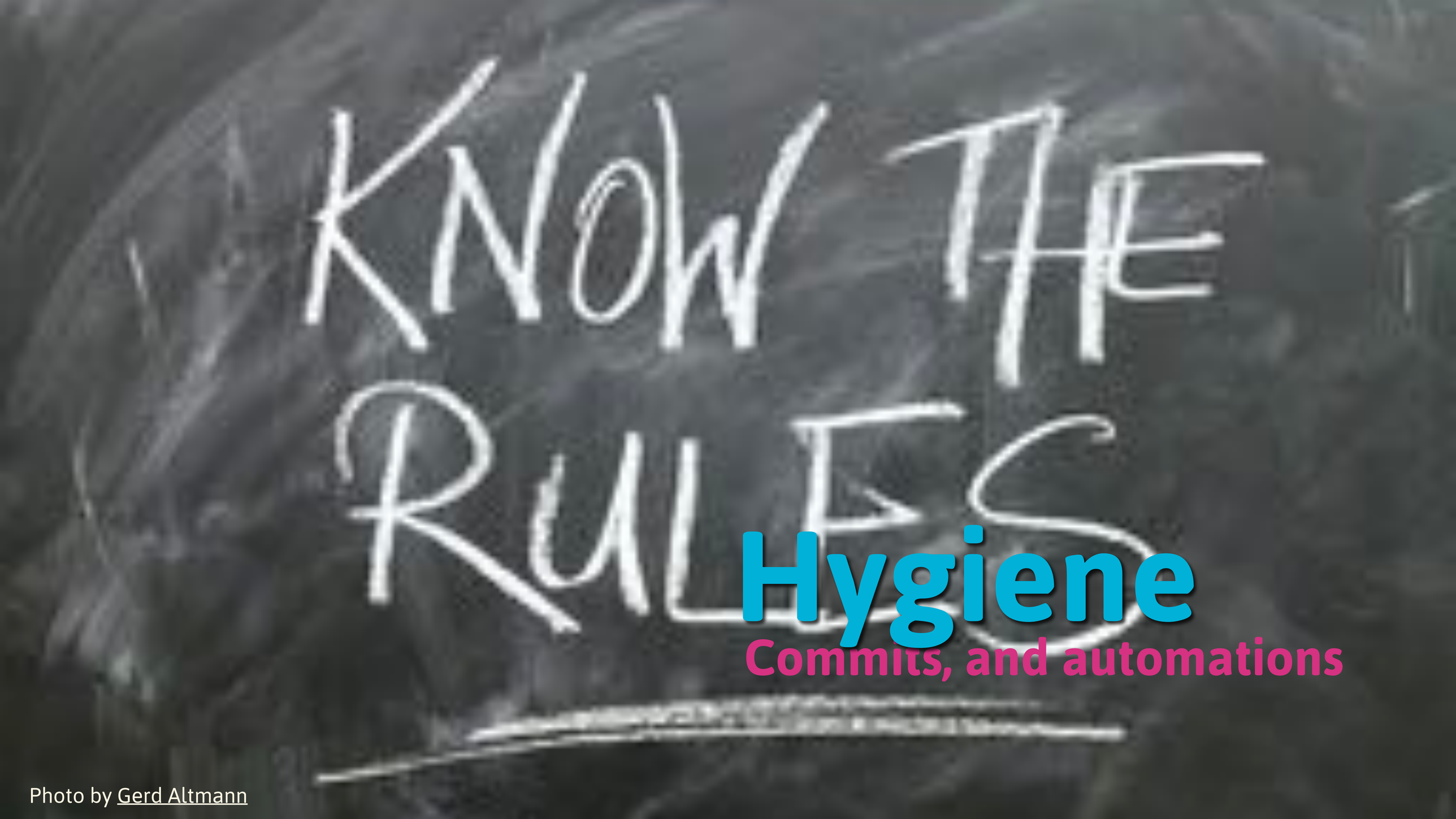
Make it so that your DAG looks like this. Then perform a "git rebase master". Observe "git --no-pager log --oneline -n 10"



Exercise

- Switch to any of your feature branches (or create one) - Add one or more commits
- Attempt a "git rebase -i master"
  - Try editing the message of one of your commits
- See the results of "git --no-pager log --oneline -n 10"

- Hints
  - Git will stop the rebase at the commit you are editing. You can use "git commit --amend" to edit the commit message
  - If you get in a bad state you can use "git rebase --abort"



Hygiene

Commits, and automations

Photo by Gerd Altmann

```
1786586747 (origin/g3) fix(core): Refresh transplanted views at insertion point only (#35968)
45c09416ed refactor(ngcc): move `PathMappings` to separate file to avoid circular dependency (#36626)
4779c4b94a fix(ngcc): handle `ENOMEM` errors in worker processes (#36626)
793cb328de fix(ngcc): give up re-spawing crashed worker process after 3 attempts (#36626)
966598cda7 fix(ngcc): support recovering when a worker process crashes (#36626)
772ccf0d9f feat(ngcc): support reverting a file written by `FileWriter` (#36626)
ff6e93163f refactor(ngcc): keep track of transformed files per task (#36626)
dff5129661 refactor(ngcc): notify master process about transformed files before writing (#36626)
e367593a26 refactor(ngcc): support running callback before writing transformed files (#36626)
16039d837e refactor(ngcc): rename `TaskQueue#markTaskCompleted()` to `markAsCompleted()` (#36626)
4665c35453 feat(ngcc): support marking an in-progress task as unprocessed (#36626)
```

```
<type>(<scope>): <subject>
<BLANK LINE>
<body>
<BLANK LINE>
<footer>
```

10.0.0-next.4 (2020-04-29)

Bug Fixes

- **compiler:** normalize line endings in ICU expansions (#36747) (70b82f7), closes #36725
- **core:** attempt to recover from user errors during creation (#36381) (3d82ac7), closes #31221
- **core:** handle synthetic props in Directive host bindings correctly (#35568) (527deea), closes #35501
- **language-service:** disable update the `@angular/core` module (#36783) (dd049ca)
- **localize:** include legacy ids when describing messages (#36751) (4799867)
- **ngcc:** recognize enum declarations emitted in JavaScript (#36550) (89c5890), closes #35584

Features

- **router:** allow CanLoad guard to return UrlTree (#36610) (00e6cb1), closes #26531 #38306

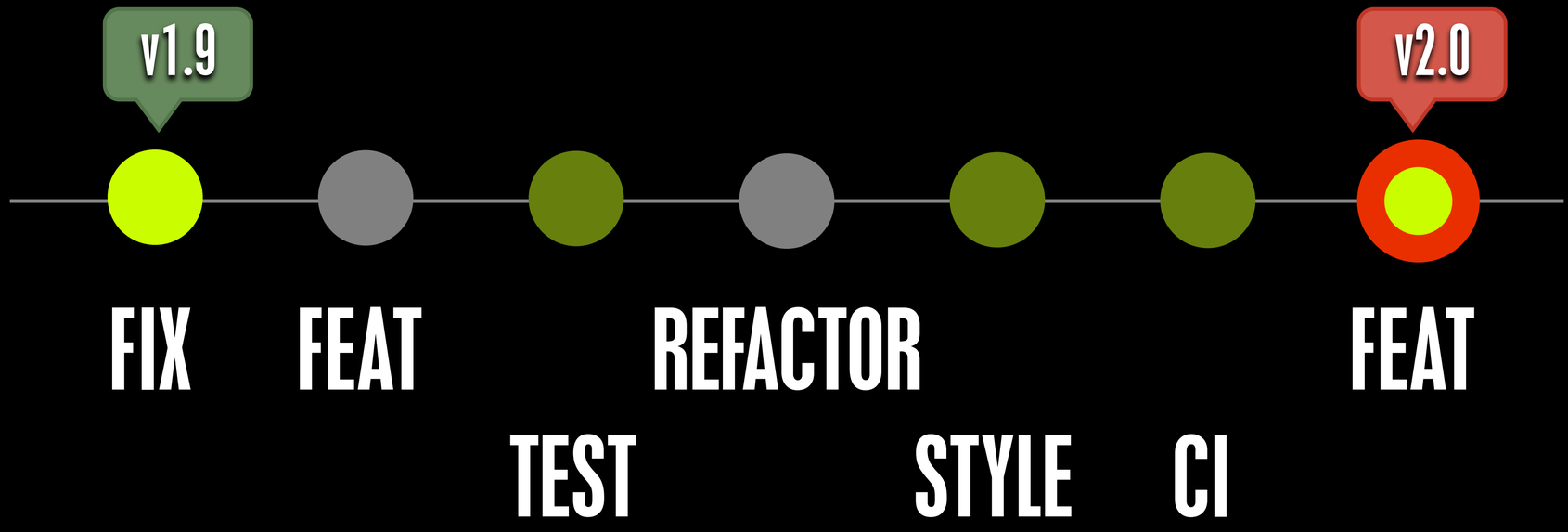
9.1.4 (2020-04-29)

Bug Fixes

- **core:** attempt to recover from user errors during creation (#36381) (d743331), closes #31221
- **core:** handle synthetic props in Directive host bindings correctly (#35568) (b73899a), closes #35501
- **language-service:** disable update the `@angular/core` module (#36783) (d3a77ea)
- **localize:** include legacy ids when describing messages (#36751) (aa84cd6)
- **ngcc:** recognize enum declarations emitted in JavaScript (#36550) (c440165), closes #35584

10.0.0-next.3 (2020-04-22)

GIT BISECT



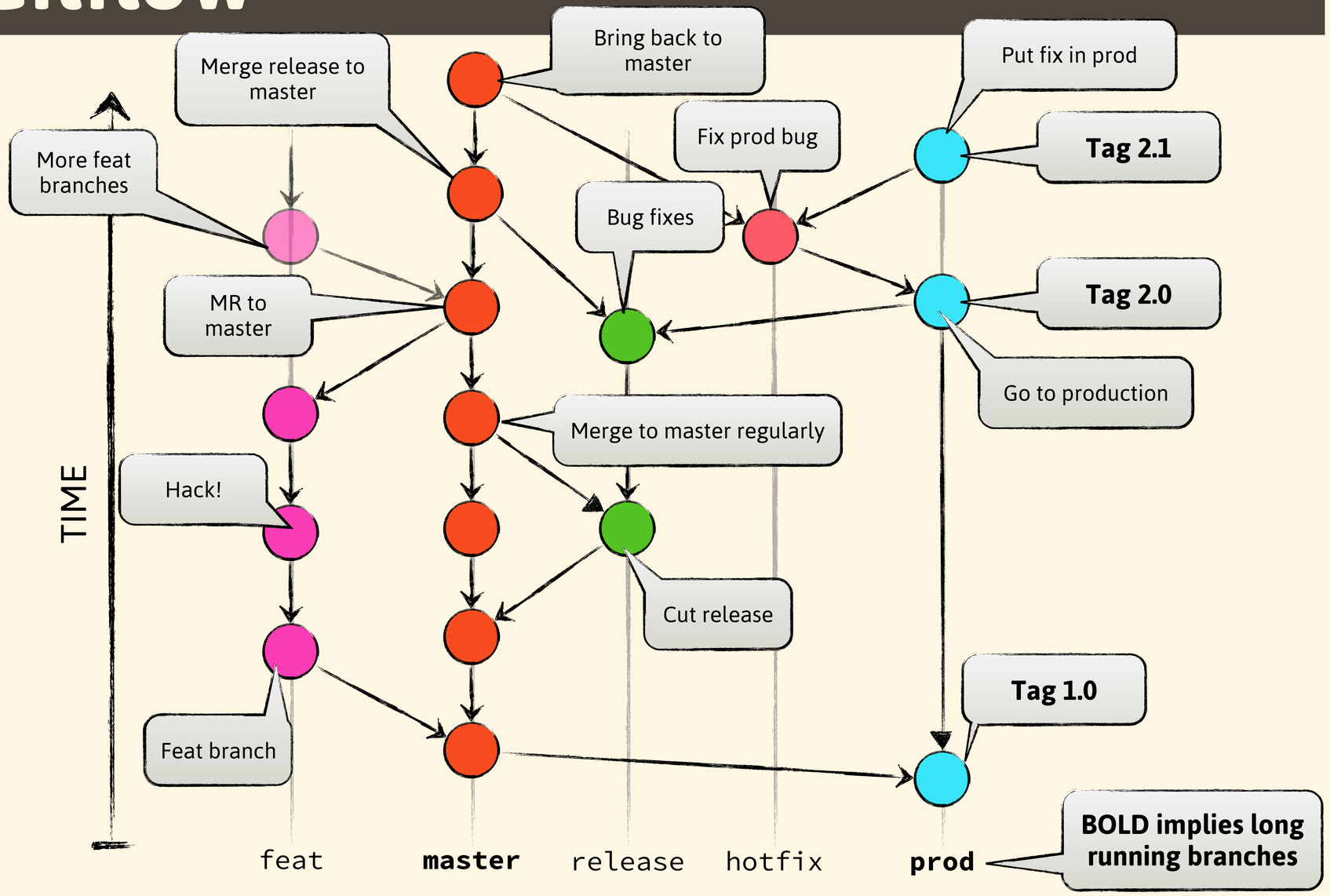
Conventional Commits

A specification for adding human and machine readable meaning to commit messages

Basic Summary | Full Specification | Contributing


Workflows
Strategies for your team

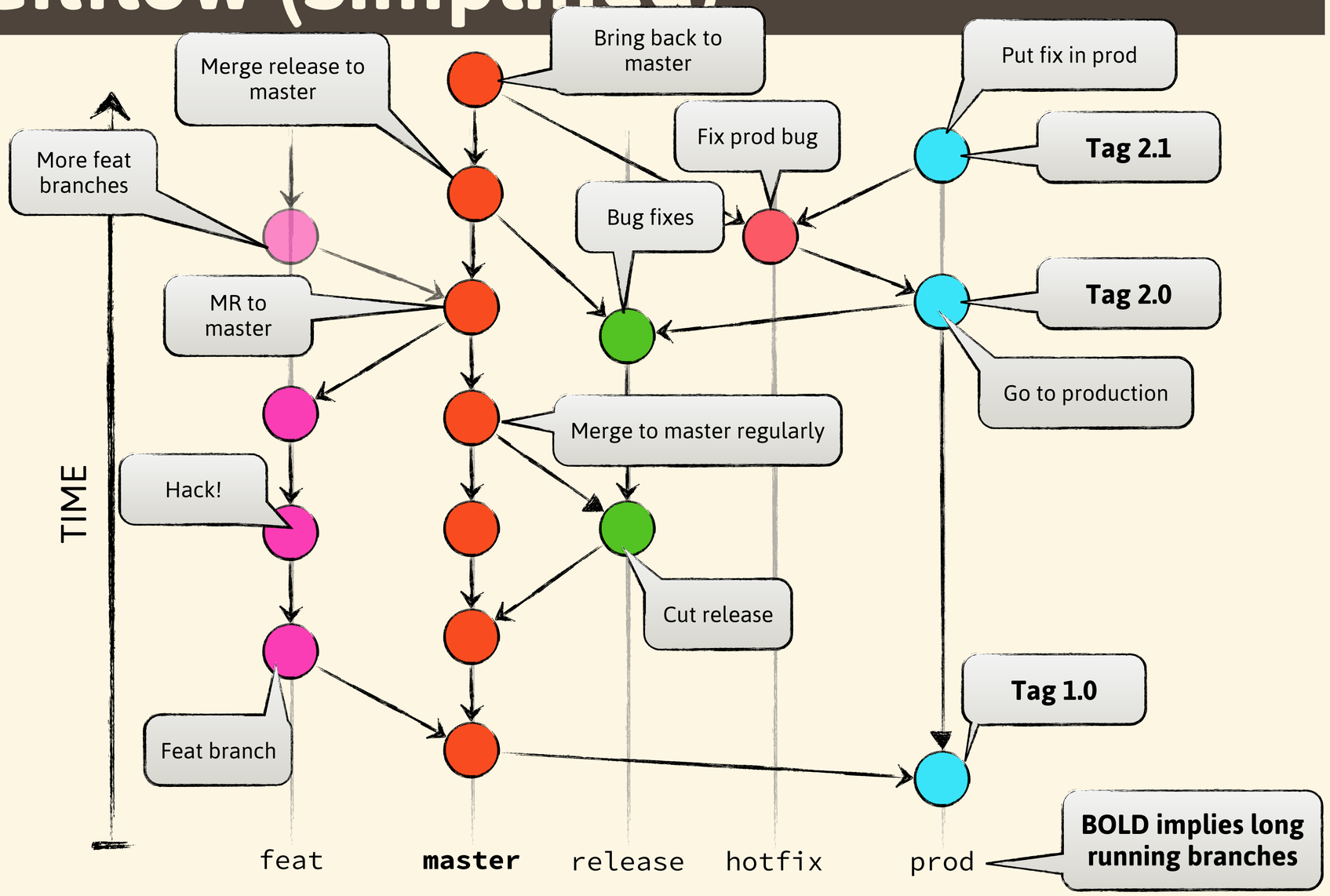
Gitflow


Merge release to master
Bring back to master
Put fix in prod
More feat branches
Fix prod bug
Tag 2.1
Bug fixes
MR to master
Tag 2.0
Go to production
Merge to master regularly
Hack!
TIME
Cut release
Tag 1.0
Feat branch
BOLD implies long running branches
feat
master
release
hotfix
prod

Gitflow

- Pros
  - Adapts well to large teams
  - Allows for a traditional "promote to production" model
  - Third party tooling available
- Cons
  - Encourages multiple long running branches
  - *May* encourage "big-bang" releases

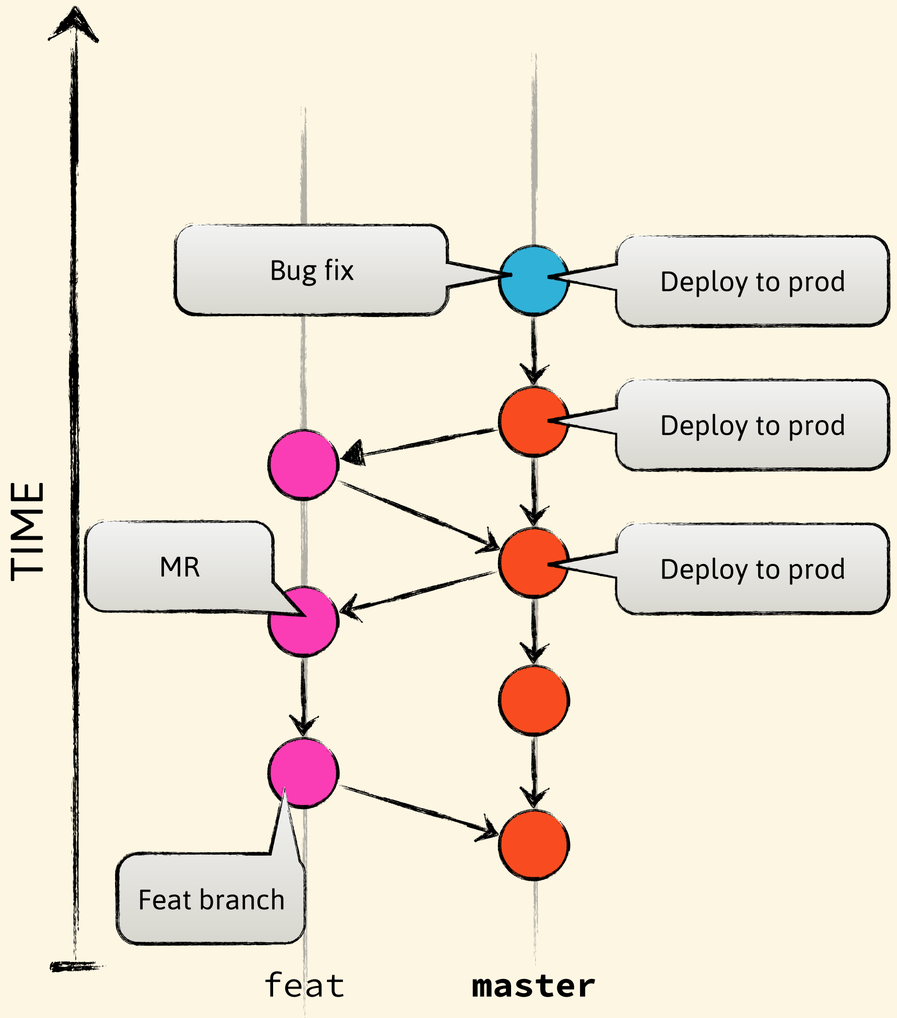
Gitflow (Simplified)



Gitflow (Simplified)

- Pros
  - Same as traditional Gitflow
  - Only **one** long running branch
- Cons
  - Same as Gitflow

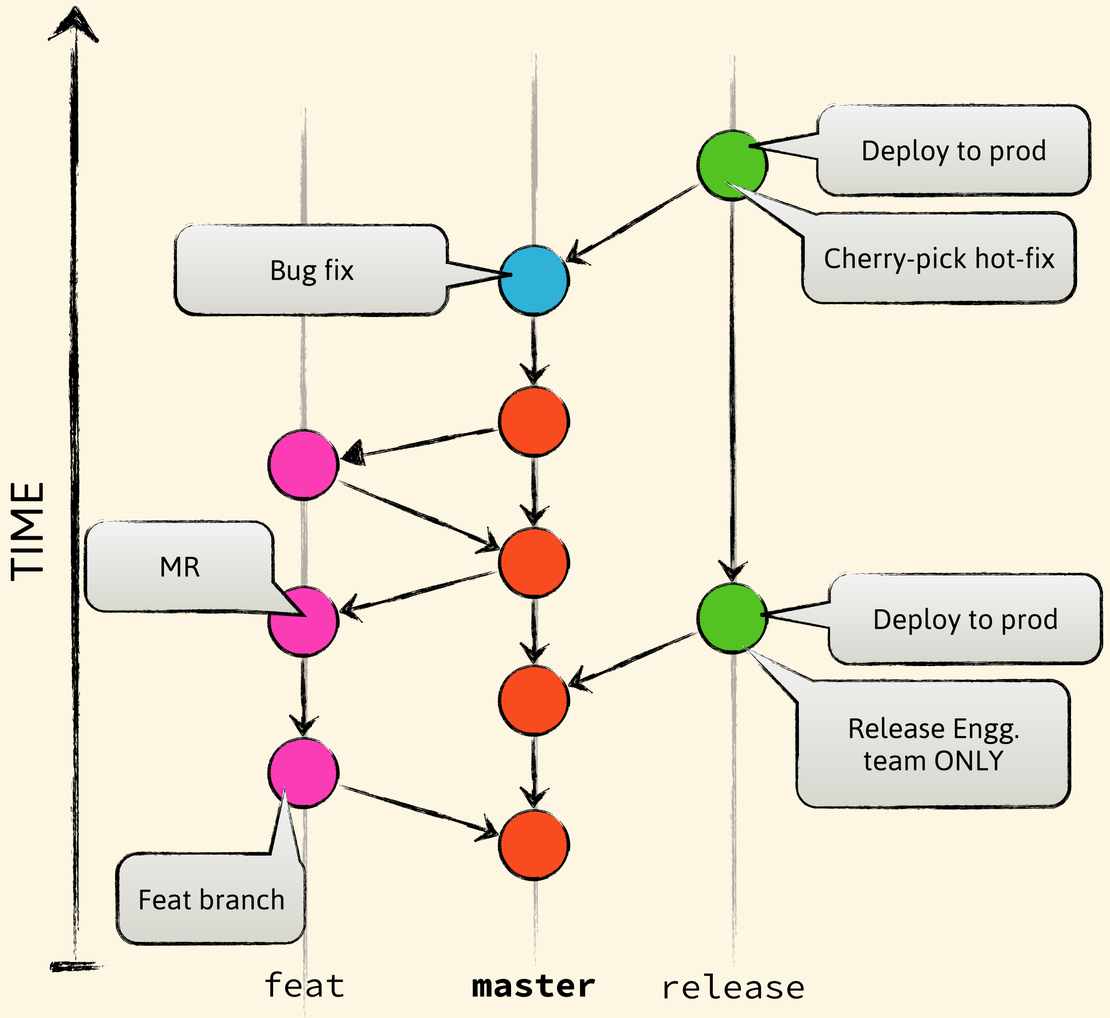
Trunk-based (CD model)



Trunk-based (CD model)

- Pros
  - Extremely simple and lightweight model
  - Allows for very small lead times
  - Lends very well to Continuous Delivery (and if possible, Continuous Deployments)
- Cons
  - Simple does not imply easy — Necessitates concepts like "branching by abstraction" and feature-toggles
- See Trunk Based Development and Branch by Abstraction

Trunk-based (Branch for release)



Trunk-based (Branch for release)

- Pros
  - Same as Trunk-Based
  - Scales for **very** large teams
- Cons
  - Same as Trunk-Based
  - Hot-fixes need to be cherry-picked
- See Trunk Based Development and Branch by Abstraction

THANKS

Credits

- Asap font
- Theme blatantly stolen from Git's homepage

Resources

- Pro Git by Scott Chacon
- Git from the bottom up by John Wiegley
- Git for Computer Scientists by Tommi Virtanen
- Git Core Concepts by Ted Naleid