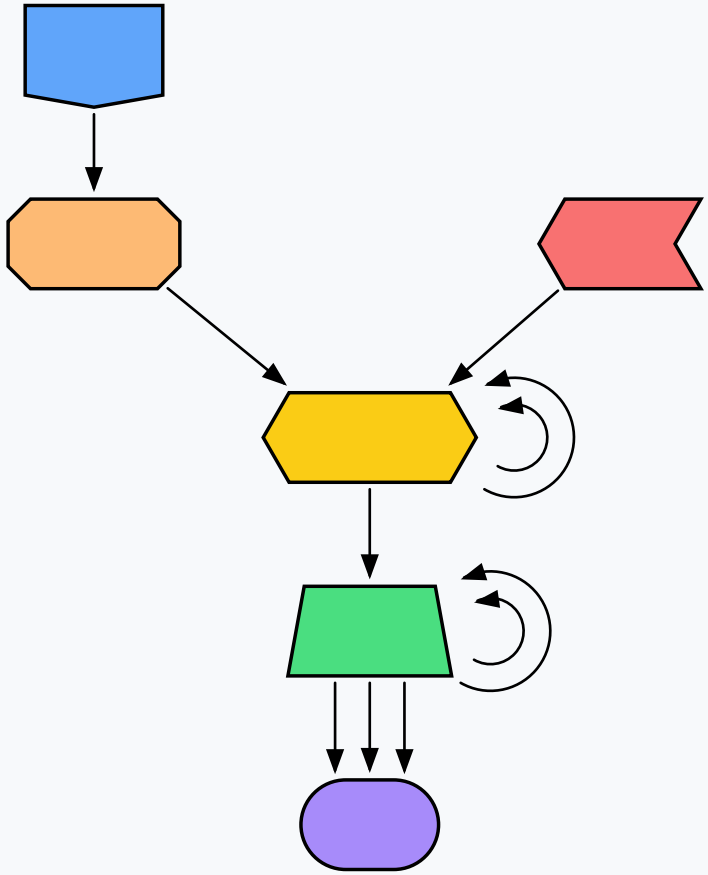
Git's internals

"Git is a content-addressable filesystem"

– The doc

Contents



“Git is fundamentally a content-addressable filesystem with a VCS user interface written on top of it”

— Also, the doc

Contents are called *objects*

- There are four different types of *objects*
- They all live in the `.git/objects` directory

List them with `git rev-list`!

Addresses are *SHA-1 hashes*

- They're computed based on the *content*
- Hence, you cannot pick a *key* yourself

Same *object*? Same *SHA-1 hash*!

Let's play with *porcelain*

```
git init
```

```
find .git/objects
```

Create a file : no *object*

```
echo "Hello" >> greetings.txt
```

```
find .git/objects
```

Stage it : one *object*

```
git add greetings.txt

find .git/objects
e9/65047ad7c57865823c7d992b1d046ea66edf78
```

And *commit* : two more *objects*

```
git commit -m "Initial commit"
```

```
[main (root-commit) f3c9648] Initial commit
1 file changed, 1 insertion(+)
create mode 100644 greetings.txt
```

```
find .git/objects
```

```
f3/c9648f6342b65f0e10972882fa722942bbcdfd
e9/65047ad7c57865823c7d992b1d046ea66edf78
8d/708e5316adbdc9e4e0f86c188f7f47e3ac6def
```

Let's check the *plumbing*

- `git cat-file -t $hash` prints the object's *type*
- `git cat-file -p $hash` pretty-prints its *content*

Commit

```
git commit -m "Initial commit"
```

```
[main (root-commit) f3c9648] Initial commit
1 file changed, 1 insertion(+)
create mode 100644 greetings.txt
```

Commit

```
git cat-file -t f3c96
```

```
commit
```

```
git cat-file -p f3c96
```

```
tree 8d708e5316adbdc9e4e0f86c188f7f47e3ac6def
author Pablo COVES <pablo.coves@pm.me> 1763754412 +0100
committer Pablo COVES <pablo.coves@pm.me> 1763754412 +0100

Initial commit
```

Tree

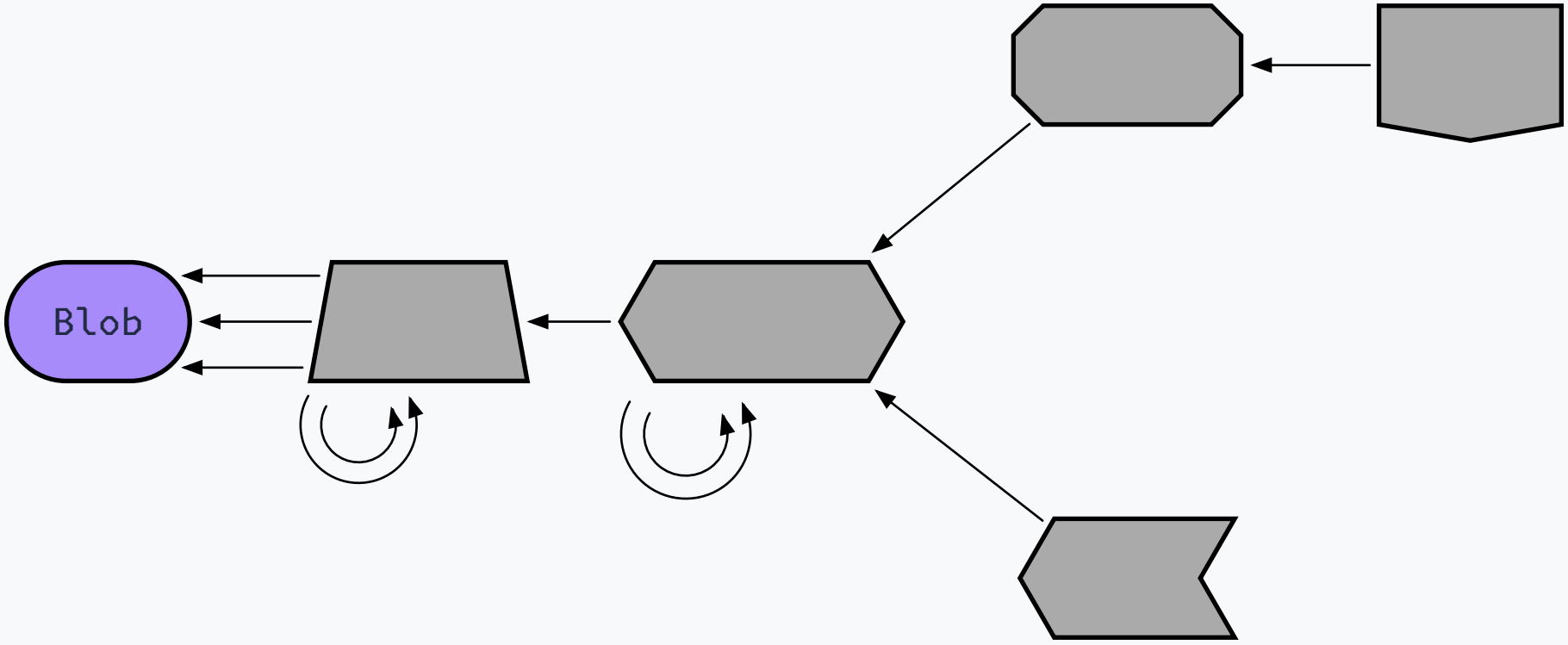
```
git cat-file -t 8d708e
tree

git cat-file -p 8d708e
100644 blob e965047ad7c57865823c7d992b1d046ea66edf78 greetings.txt
```

Blob

```
git cat-file -t e9650
blob

git cat-file -p e9650
Hello
```


Blob

Binary Large OBject

```
git init
echo "Hello" > greetings.txt
git add greetings.txt
```

```
find .git/objects
```

```
e9/65047ad7c57865823c7d992b1d046ea66edf78
```

It's *compressed* using zlib

```
cat .git/objects/e9/65047ad7c57865823c7d992b1d046ea66edf78
xK0R0cH

cat .git/objects/e9/650[..] | zlib-flate -uncompress
blob 6Hello
```

It contains its *type* and *size*

```
git cat-file -t e9650
blob
```

```
wc -c < greetings.txt
6
```

`wc -c` outputs the size of its input in *bytes*

Let's double-check

```
echo -e "blob 6\0Hello" | zlib-flate -compress
xK0R0cH

cat .git/objects/e9/65047ad7c57865823c7d992b1d046ea66edf78
xK0R0cH
```

Blob's format : <type>< ><size><\0><content>

Address

```
echo -e "blob 6\0Hello" | sha1sum
e965047ad7c57865823c7d992b1d046ea66edf78

git hash-object --stdin <<< "Hello"
e965047ad7c57865823c7d992b1d046ea66edf78
```

What about the /?

```
find .git/objects
e9/65047ad7c57865823c7d992b1d046ea66edf78
```

It helps `git` process smaller directories

Also, FAT-32 can't handle more that 65,536 files in a single directory so it makes `git` usable on `windows` for big repositories

Bring Your Own Blob

1. Take a file's content
2. Prepend its size and `\0`
3. Prepend its type `blob`
4. Zip it at its `SHA-1` checksum

What's next?

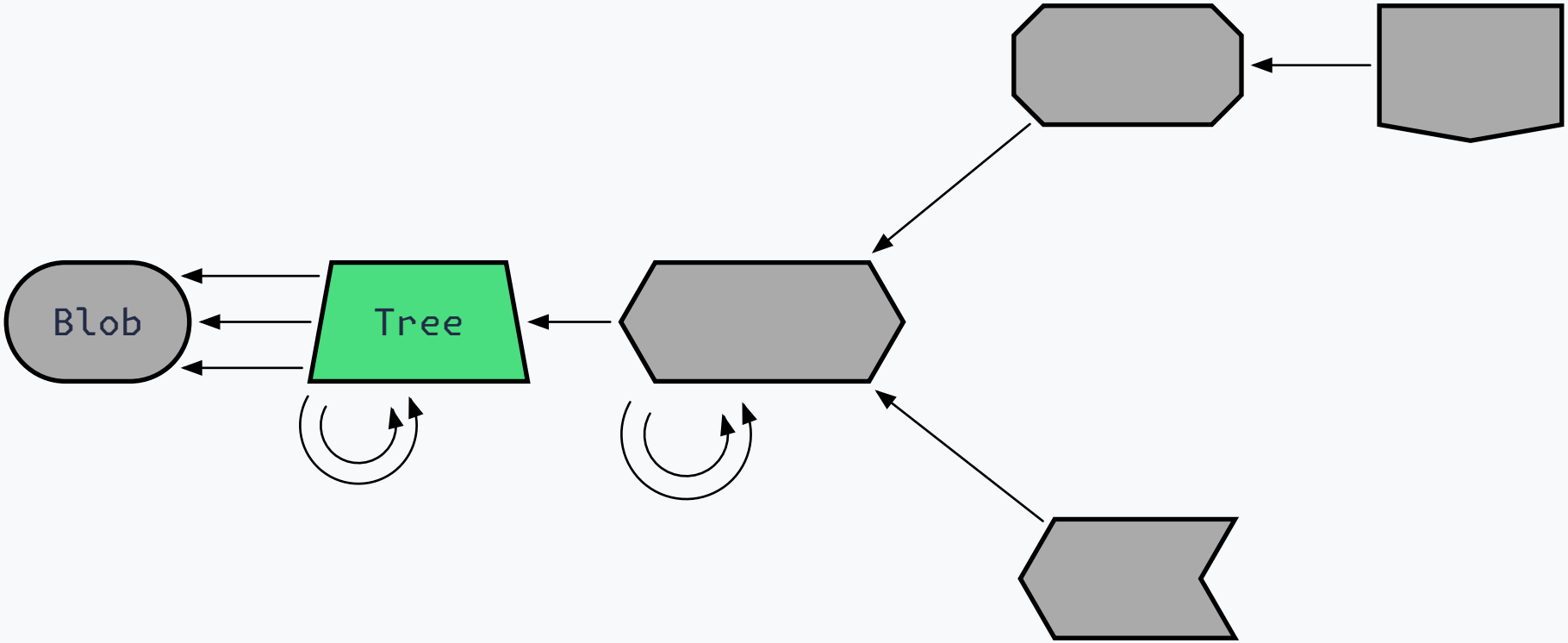
Did you spot the missing bits yet? 🤔

What's next?

There's no metadata, only content!

No path, no type, no permissions…

How does `git` know *where* to write *what*?


Blob
Tree

Tree

```
git init
echo "Hello" > greetings.txt
git add greetings.txt
git commit -m "Initial commit"
```

```
find .git/objects
```

```
f3/c9648f6342b65f0e10972882fa722942bbcdfd
e9/65047ad7c57865823c7d992b1d046ea66edf78
8d/708e5316adbdc9e4e0f86c188f7f47e3ac6def
```

What was that again?

```
git cat-file -t 8d708e
tree

git cat-file -p 8d708e
100644 blob e965047ad7c57865823c7d992b1d046ea66edf78 greetings.txt
```

Oh yeah, *type*, *size*, *content*

```
cat .git/objects/8d/708e[..]  | zlib-flate -uncompress
```

```
tree 41100644 greetings.txtezxe<}+nnx
```

Almost

The actual format is as follow

```
<type>< ><size><\0>(<mode>< ><path><\0><sha1-binary>)*
```

Since dealing with binary format is not convenient, we'll stick to pretty-printing for now

A *mode*?

Mode	Type	Description
100644	blob	Regular file
100755	blob	Executable file
120000	blob	Symbolic link
040000	tree	Directory
160000	commit	Submodule

Point to another *tree*?

Yep! And this is how `git` handles nested directories!

Let's see how it's done

```
mkdir foo
echo Autruche > foo/bar
echo Autruche > foo/baz
git add foo
git commit -m "feat: add foo directory"
```

```
[main 71dbf7e] feat: add foo directory
2 files changed, 2 insertions(+)
create mode 100644 foo/bar
create mode 100644 foo/baz
```

Pop quiz!

How many `git` objects are stored?

```
find .git/objects
```

```
.git/objects/f3/c9648f6342b65f0e10972882fa722942bbcdfd
.git/objects/fe/12d8007e7a6d5310abb583fd82fefde9d28861
.git/objects/7b/a1e7a055fba65387cbc061d45c9ad584796f4d
.git/objects/19/c1ba5d8ac0828d5ea56ad21238240e0f9389b2
.git/objects/e9/65047ad7c57865823c7d992b1d046ea66edf78
.git/objects/8d/708e5316adbdc9e4e0f86c188f7f47e3ac6def
.git/objects/71/dbf7e44b95e9419a0f040da129ed21f428deaf
```

7

Files `bar` and `baz` have the same content.

Remember: same content, same blob!

Same old same old

```
git commit -m "feat: add foo directory"
```

```
[main 71dbf7e] feat: add foo directory
2 files changed, 2 insertions(+)
create mode 100644 foo/bar
create mode 100644 foo/baz
```

Commit

```
git cat-file -t 71dbf
```

```
commit
```

```
git cat-file -p 71dbf
```

```
tree fe12d8007e7a6d5310abb583fd82fefde9d28861
parent f3c9648f6342b65f0e10972882fa722942bbcdfd
author Pablo COVES <pablo.coves@pm.me> 1763754961 +0100
committer Pablo COVES <pablo.coves@pm.me> 1763754961 +0100

feat: add foo directory
```

Tree

```
git cat-file -t fe12d
```

```
tree
```

```
git cat-file -p fe12d
```

```
040000 tree 19c1ba5d8ac0828d5ea56ad21238240e0f9389b2 foo
100644 blob e965047ad7c57865823c7d992b1d046ea66edf78 greetings.txt
```

Tree

```
git cat-file -t 19c1b
```

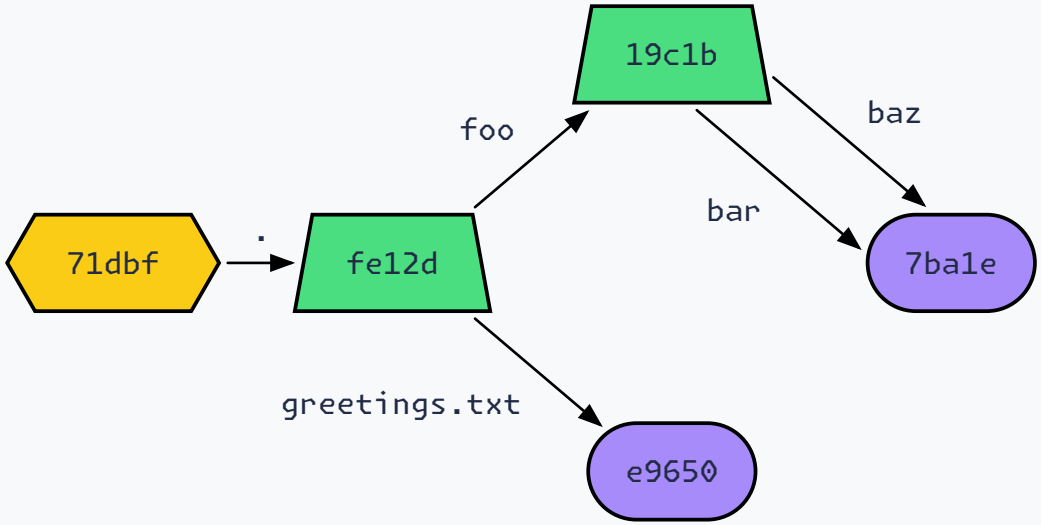
```
tree
```

```
git cat-file -p 19c1b
```

```
100644 blob 7ba1e7a055fba65387cbc061d45c9ad584796f4d bar
100644 blob 7ba1e7a055fba65387cbc061d45c9ad584796f4d baz
```

Depth !

```
.
├── foo
│   ├── bar
│   └── baz
└── greetings.txt
```



- A `git` tree is like a filesystem directory
- It can point to ~~files~~ blobs, other ~~directories~~ trees
- Or even whole `git` repositories, for what it's worth!

What's next?

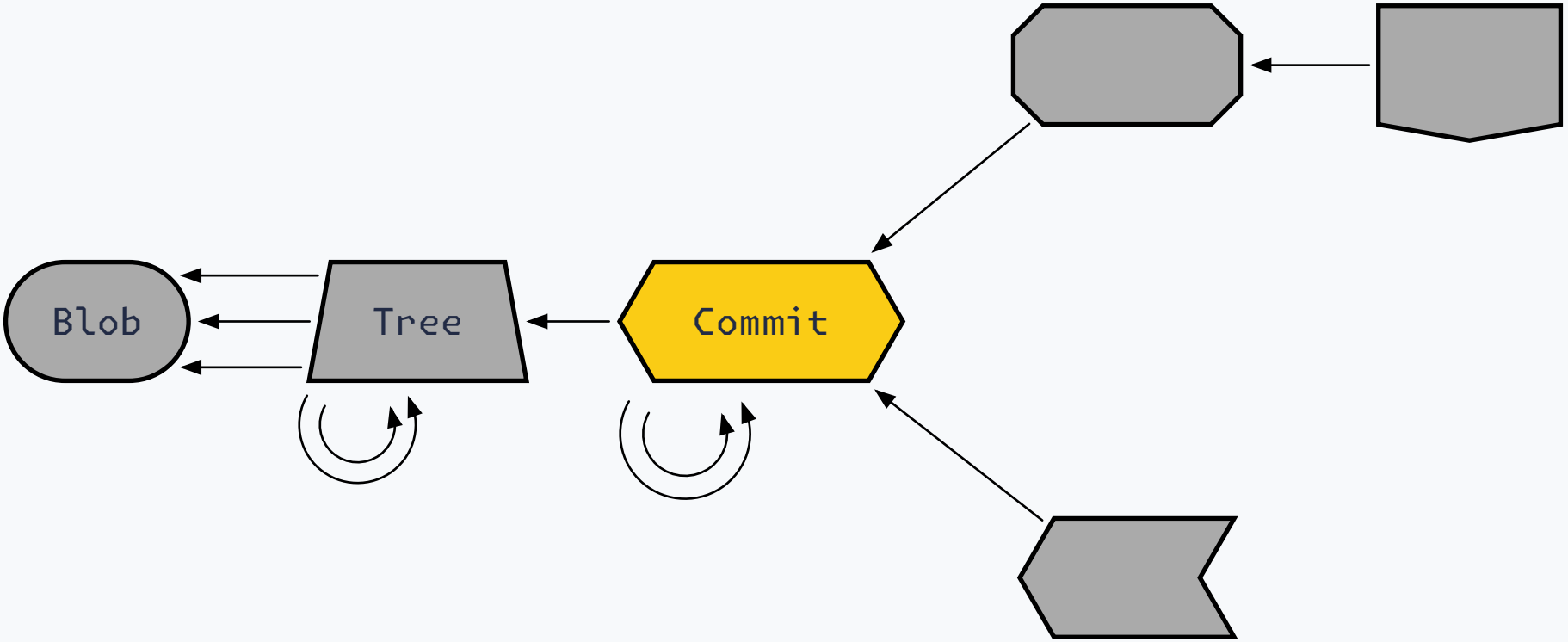
Did you spot the missing bits yet? 🤔

What's next?

We have paths, contents…

But *who* is to *blame*? And *when*?

How does `git` store its history?


Blob
Tree
Commit

Commit

```
git commit -m "feat: add foo directory"
```

```
[main 71dbf7e] feat: add foo directory
2 files changed, 2 insertions(+)
create mode 100644 foo/bar
create mode 100644 foo/baz
```

Who's your daddy?

```
git cat-file -p 71dbf
```

```
tree fe12d8007e7a6d5310abb583fd82fefde9d28861
parent f3c9648f6342b65f0e10972882fa722942bbcdfd
author Pablo COVES <pablo.coves@pm.me> 1763754961 +0100
committer Pablo COVES <pablo.coves@pm.me> 1763754961 +0100

feat: add foo directory
```

Who's your daddy?

```
git cat-file -p f3c96

tree 8d708e5316adbdc9e4e0f86c188f7f47e3ac6def
author Pablo COVES <pablo.coves@pm.me> 1763754412 +0100
committer Pablo COVES <pablo.coves@pm.me> 1763754412 +0100

Initial commit
```

Being a single parent is hard

- The last commit has a single *parent*
- The initial one has no *parent* at all!

Let's make some friends

```
git switch -c dev
Switched to a new branch 'dev'
```

Get to know eachother

```
echo "World" >> greetings.txt
```

```
git commit -am "fix(greatings): great the world"
```

```
[dev 4dc6343] fix(greatings): great the world
1 file changed, 1 insertion(+)
```

And share a ~~toothbrush~~ history

```
git switch main
```

```
Switched to branch 'main'
```

```
git merge --no-ff dev
```

```
Merge made by the 'ort' strategy.
greetings.txt | 1 +
1 file changed, 1 insertion(+)
```

And share a ~~toothbrush~~ history

```
git log --graph --abbrev-commit --decorate --oneline
```

```
*   52580cf (HEAD -> main) Merge branch 'dev'
|\
| * 4dc6343 (dev) fix(greatings): great the world
|/
* 71dbf7e feat: add foo directory
* f3c9648 Initial commit
```

And share a ~~toothbrush~~ history

```
git cat-file -p 52580
```

```
tree ad86bdedd95bcc3eb58c3246014927c95c4dc42c
parent 71dbf7e44b95e9419a0f040da129ed21f428deaf
parent 4dc63435734a09801af8ee36a692a253cded700b
author Pablo COVES <pablo.coves@pm.me> 1763759002 +0100
committer Pablo COVES <pablo.coves@pm.me> 1763759002 +0100

Merge branch 'dev'
```

You gotta commit

- A `git` commit brings the whole story together
- It allows you to use `git` as a time-machine
- And a commit can have plenty of parents! 🐙

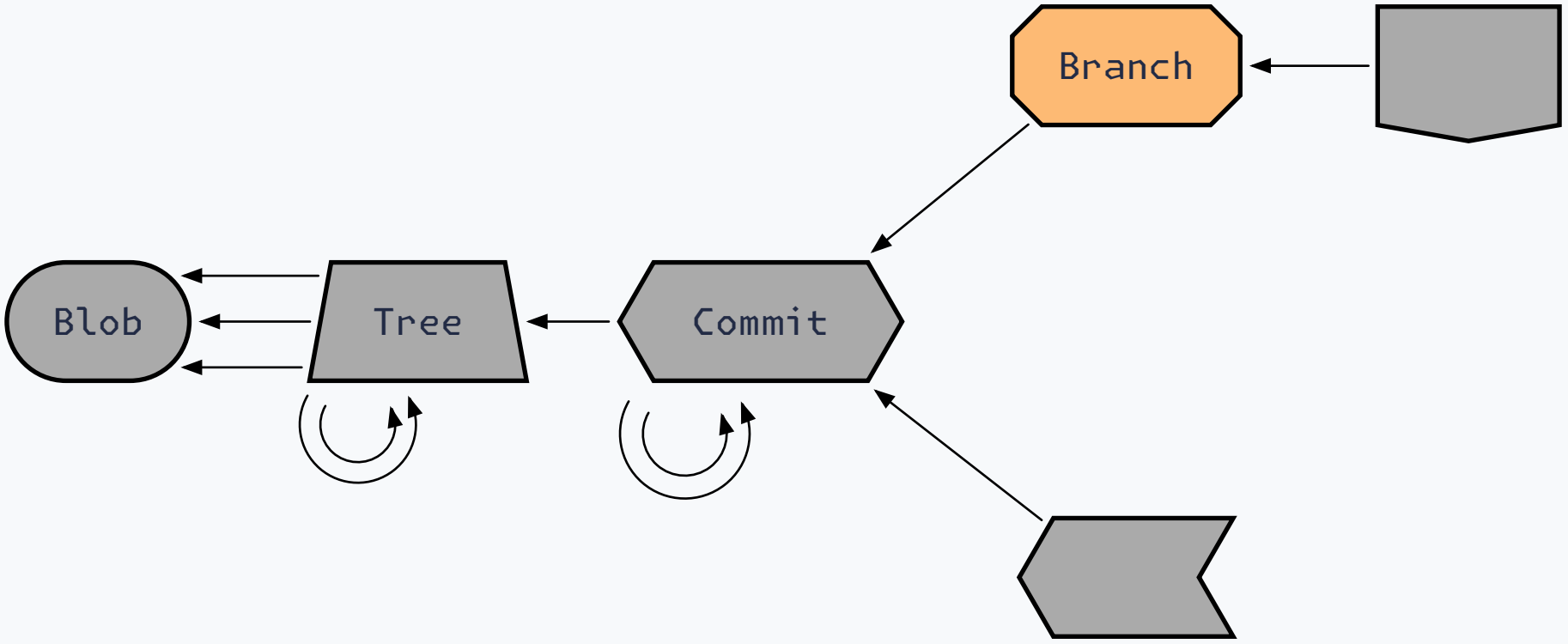
What's next?

Did you spot the missing bits yet? 🤔

What's next?

Just kidding, we could have stopped right there

Unless…


Branch
Blob
Tree
Commit

Long time no see, old Branch

```
git log --graph --abbrev-commit --decorate --oneline
```

```
*   52580cf (HEAD -> main) Merge branch 'dev'
|\
| * 4dc6343 (dev) fix(greatings): great the world
|/
* 71dbf7e feat: add foo directory
* f3c9648 Initial commit
```

You know the drill

```
git cat-file -t main
```

```
commit
```

```
git cat-file -t dev
```

```
commit
```

You know the drill

```
git cat-file -p main
```

```
tree ad86bdedd95bcc3eb58c3246014927c95c4dc42c
parent 71dbf7e44b95e9419a0f040da129ed21f428deaf
parent 4dc63435734a09801af8ee36a692a253cded700b
author Pablo COVES <pablo.coves@pm.me> 1763759002 +0100
committer Pablo COVES <pablo.coves@pm.me> 1763759002 +0100

Merge branch 'dev'
```

You know the drill

```
git cat-file -p dev
```

```
tree ad86bdedd95bcc3eb58c3246014927c95c4dc42c
parent 71dbf7e44b95e9419a0f040da129ed21f428deaf
author Pablo COVES <pablo.coves@pm.me> 1763758924 +0100
committer Pablo COVES <pablo.coves@pm.me> 1763758924 +0100

fix(greatings): great the world
```

Git branches are not objects!

A *branch* just points at a *commit*

```
cat .git/refs/heads/{main,dev}
```

```
52580cf830b7dff558781a09ba0ffed96454c946
4dc63435734a09801af8ee36a692a253cded700b
```

!! See?

They're not even in the `.git/objects` directory!

- A `git` branch simply references a commit
- When you create a commit, `git` updates the branch
- They’re light, they don’t duplicate content!

What's next?

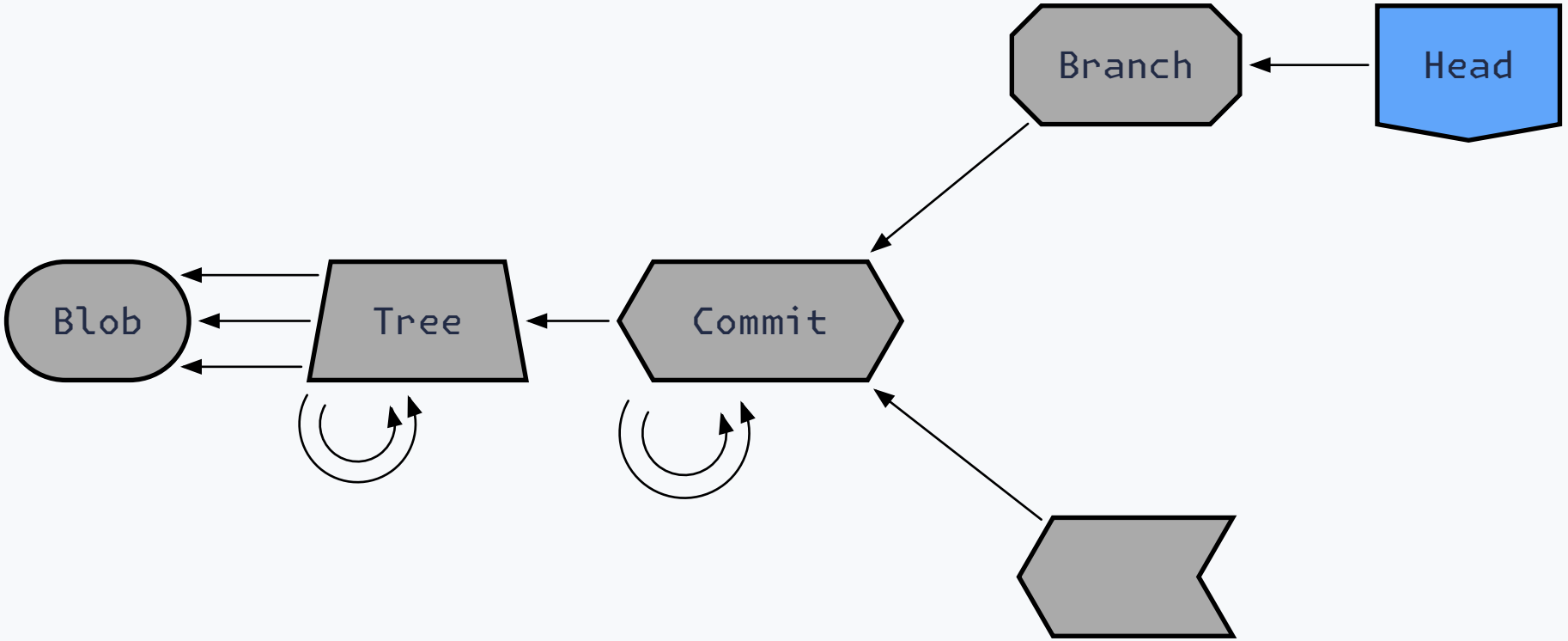
Did you spot the missing bits yet? 🤔

What's next?

Okay, this is getting old

Bear with me

We're almost done using our *HEAD*


Branch
Head
Blob
Tree
Commit

Where's your *H*ead at?

```
git log --graph --abbrev-commit --decorate --oneline
```

```
*   52580cf (HEAD -> main) Merge branch 'dev'
|\
| * 4dc6343 (dev) fix(greatings): great the world
|/
* 71dbf7e feat: add foo directory
* f3c9648 Initial commit
```

Why change a winning game?

```
git cat-file -t HEAD
```

```
commit
```

Why change a winning game?

```
git cat-file -p HEAD
```

```
tree ad86bdedd95bcc3eb58c3246014927c95c4dc42c
parent 71dbf7e44b95e9419a0f040da129ed21f428deaf
parent 4dc63435734a09801af8ee36a692a253cded700b
author Pablo COVES <pablo.coves@pm.me> 1763759002 +0100
committer Pablo COVES <pablo.coves@pm.me> 1763759002 +0100

Merge branch 'dev'
```

It’s just the last commit, isn’t it?

With extra steps, yes

```
cat .git/HEAD
ref: refs/heads/main

cat .git/refs/heads/main
52580cf830b7dff558781a09ba0ffed96454c946
```

What's so special about *HEAD*?

```
git reflog
```

```
52580cf (HEAD -> main) HEAD@{0}: merge dev [..]
71dbf7e HEAD@{1}: checkout: moving from dev to main
4dc6343 (dev) HEAD@{2}: commit: fix(greatings): great the world
71dbf7e HEAD@{3}: checkout: moving from main to dev
71dbf7e HEAD@{4}: commit: feat: add foo directory
f3c9648 HEAD@{5}: commit (initial): Initial commit
```

It's a breadcrumb trail

```
git switch 71dbf7e --detach

HEAD is now at 71dbf7e feat: add foo directory
```

It's a breadcrumb trail

```
git reflog
```

```
71dbf7e (HEAD) HEAD@{0}: checkout: moving from main to 71dbf7e
52580cf (main) HEAD@{1}: merge dev: [..]
71dbf7e (HEAD) HEAD@{2}: checkout: moving from dev to main
4dc6343 (dev) HEAD@{3}: commit: fix(greatings): great the world
71dbf7e (HEAD) HEAD@{4}: checkout: moving from main to dev
71dbf7e (HEAD) HEAD@{5}: commit: feat: add foo directory
f3c9648 HEAD@{6}: commit (initial): Initial commit
```

It's a breadcrumb trail

```
git log --graph --abbrev-commit --decorate --oneline

* 71dbf7e (HEAD) feat: add foo directory
* f3c9648 Initial commit
```

The detached head state 👑

It’s when `HEAD` does not point to a branch

Do yourself a favor, use `switch` and forget about `checkout`
It’s the best way not to get lost in the `commits` history

What's next?

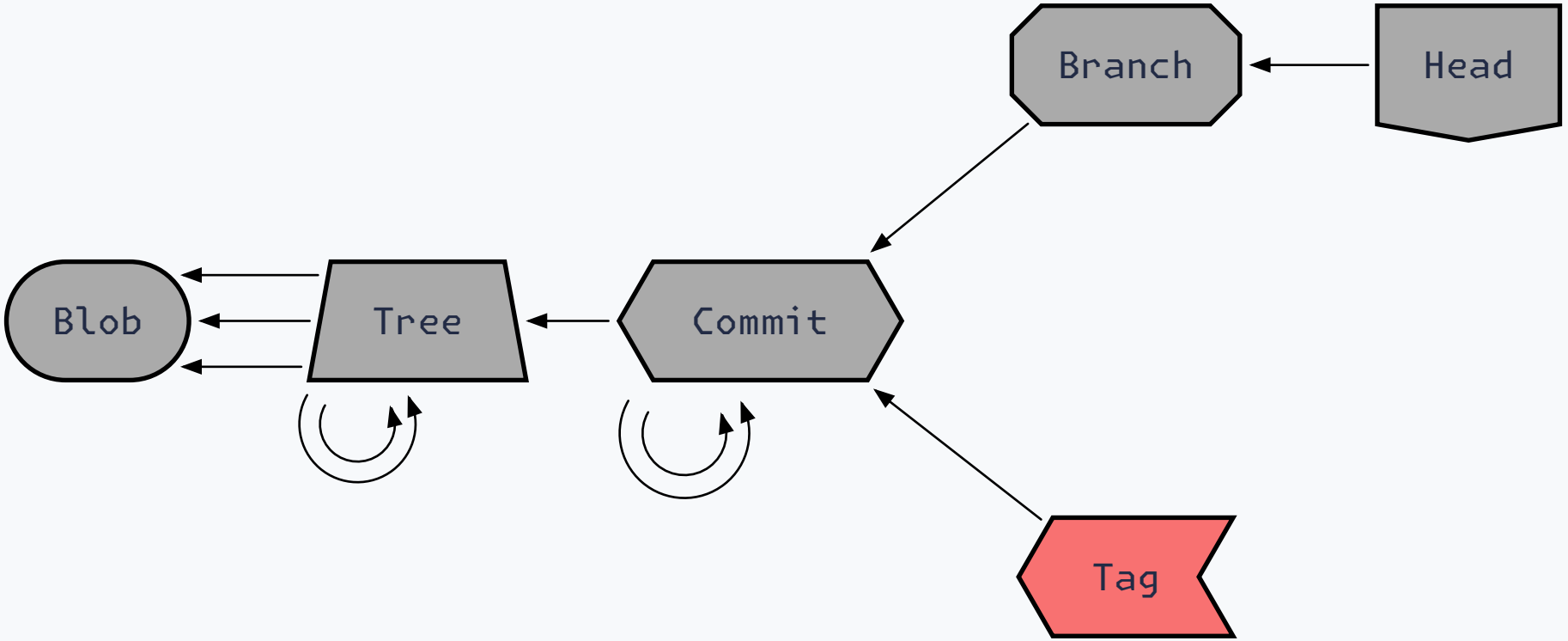
Did you spot the missing bits yet? 🤔

What's next?

Last time, pinky swear!

There's yet another way to point at a *commit*

And it's time to play a game of *tag*!


Branch
Head
Blob
Tree
Commit
Tag

Tag, you're it

```
git switch main
git log --graph --abbrev-commit --decorate --oneline
```

```
*   52580cf (HEAD -> main) Merge branch 'dev'
|\
| * 4dc6343 (dev) fix(greatings): great the world
|/
* 71dbf7e feat: add foo directory
* f3c9648 Initial commit
```

Git allows one to *tag* a *commit*

```
git tag v0.1.0
git log --graph --abbrev-commit --decorate --oneline
```

```
*   52580cf (HEAD -> main, tag: v0.1.0) Merge branch 'dev'
|\
| * 4dc6343 (dev) fix(greatings): great the world
|/
* 71dbf7e feat: add foo directory
* f3c9648 Initial commit
```

Here is our last git *object*

```
git cat-file -t v0.1.0
```

```
commit
```

Wait what?

```
cat .git/refs/tags/v0.1.0
```

```
52580cf830b7dff558781a09ba0ffed96454c946
```

Not even in the `.git/objects` directory!

Let me try again

```
git tag --delete v0.1.0
Deleted tag 'v0.1.0' (was 52580cf)

git tag --annotate v0.1.0 --message "An object tag"

git cat-file -t v0.1.0
tag
```

That's much better!

```
git cat-file -p v0.1.0
```

```
object 52580cf830b7dff558781a09ba0ffed96454c946
type commit
tag v0.1.0
tagger Pablo COVES <pablo.coves@pm.me> 1763840721 +0100

An object tag
```

There are two kinds of tags 🏷️

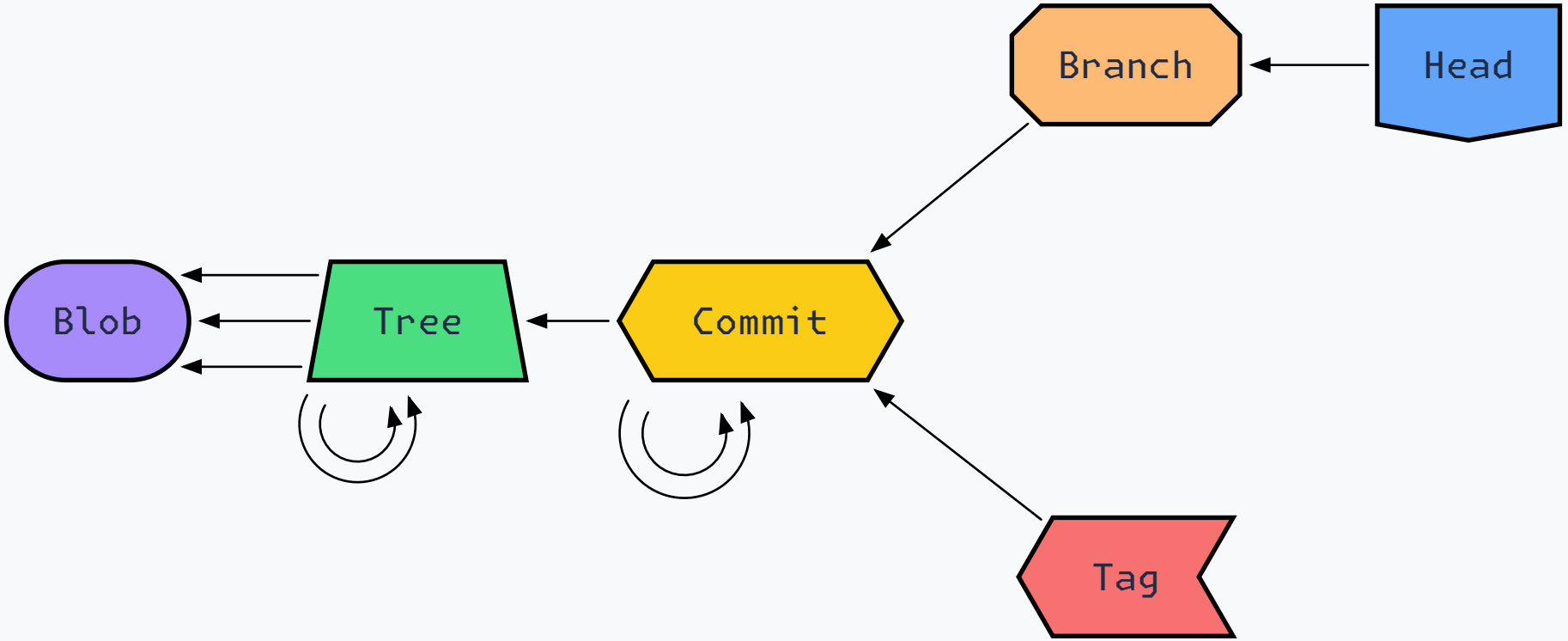
- Light tags which are mere pointers to commits
- Annotated tags which are actual `git` objects

What's next?

Did you spot the missing bits yet? 🤔

What's next?

Just kidding, we're good


Branch
Head
Blob
Tree
Commit
Tag

We *pushed* it that far

```
git log --graph --abbrev-commit --decorate --oneline
```

```
*   52580cf (HEAD -> main, tag: v0.1.0) Merge branch
'dev'
|\
| * 4dc6343 (dev) fix(greatings): great the world
|/
* 71dbf7e feat: add foo directory
* f3c9648 Initial commit
```

That’s all folks!,

- Blobs store files content and save space
- Trees add depth to the flat `.git/objects` directory
- Commits establish order and assign blame
- Tags give human-readable sense to SHA-1 hashes
- Branches are really just pointers
- And HEAD shows where you’ve been

Questions?

https://gitlab.com/pcoves/gits_internals