

Git 2

The inner workings of git



A short recap

- stage / index
- commit
- branch
- merge
- rebase
- push / fetch / pull



This time

- branch
- commit
- tree
- blob
- rebase

... the inside view



The `.git` folder

We'll dig through the `.git` folder and explore the bits and pieces which make up `git`.

*While it's **not** recommended to manually mess with the `.git` folder in production repositories, doing so has quite some educational value.*



What is a branch?

```
1 !cat .git/refs/heads/main
```

```
d2995d1549878920c372e4af6b1f75367b2095fd
```

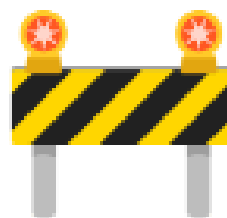


```
1 !cat .git/refs/heads/main
```

```
d2995d1549878920c372e4af6b1f75367b2095fd
```

- a branch is a **ref** (it's in the `refs` folder)
- a **ref** (so a branch) is **just a pointer to a hash**

let's dig deeper... 



(but first, a *little* detour)

Cryptographic hash

Algorithm producing fixed-size bytes h from any input m :

$$h = \text{hash}(m)$$

- deterministic
- fast
- pre-image resistant: $h \not\rightarrow m$
- collision resistant ¹: $\text{hash}(m_1) \neq \text{hash}(m_2)$

1. SHA1 is broken



SHA1

- git's current default ¹
- produces 160-bit (20 byte) hash value
- often represented in hexadecimal notation:
 - 2 digits [0-9a-f] for each byte
 - 40 digits

1. git [transitions away from SHA1](#)



Content Addressable Store

Items are addressed by **what** they are
instead of *where* they are.



CAS use hashes

```
1 from hashlib import sha1
2 class CAS:
3     def __init__(self):
4         self._store = {}
5     def put(self, content):
6         key = sha1(content).hexdigest()
7         self._store[key] = content
8         return key
9     def get(self, key):
10        return self._store[key]
11
12 store = CAS()
13 store.put(b"hello")
```

```
'aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d'
```

```
1 store.get('aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d')
```

```
b'hello'
```



objects are immutable

```
1 store.put(b"hello")
```

```
'aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d'
```

```
1 store.put(b"world")
```

```
'7c211433f02071597741e6ff5a8ea34789abbf43'
```

```
1 store.get('aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d')
```

```
b'hello'
```

An object either exists or not.
(but it's never "old")

... back to git 

commit

```
1 !cat .git/refs/heads/main
```

```
d2995d1549878920c372e4af6b1f75367b2095fd
```

```
1 !git cat-file -p d2995d1549878920c372e4af6b1f75367b2095fd
```

```
tree b219984ac18ed30a40059099f66bb9beba1ea085
```

```
parent 183a295f09e0a5daf4da9fba6c7cd016e7de4546
```

```
author Tobias Kölling <tobias.koelling@mpimet.mpg.de> 1712570344 +0200
```

```
committer Tobias Kölling <tobias.koelling@mpimet.mpg.de> 1712570344 +0200
```

```
add registration notice
```



tree

```
tree b219984ac18ed30a40059099f66bb9beba1ea085
parent 183a295f09e0a5daf4da9fba6c7cd016e7de4546
author Tobias Kölling <tobias.koelling@mpimet.mpg.de> 1712570344 +0200
...
```

```
1 !git cat-file -p b219984ac18ed30a40059099f66bb9beba1ea085
```

```
100644 blob 8dbee0df3b3e0acfcf8ee05614131a8566cb4e21      .gitignore
100644 blob 763d3d4997864c873835ebbef2e000d5c73a4b07      .gitlab-ci.yml
100644 blob 01fefe8e7a9a2b2d5651f88cbd3d62e4c386d81b      .pre-commit-config.yml
100644 blob 343e822cbec4008e50a8be7685e26dda3adcd3e9      LICENSE
100644 blob 8f47e134c259766ae06e653e0a98f7dadcb56ed      README.md
100644 blob 97d3cadd2ddd6a3c74c41cdb17b0342d04fa69f      _quarto.yml
100644 blob 1aecac4d48ad857aeb3badc95b00f562b77b4890      environment.yml
100644 blob cf8294e57cd9666b6a27aa7864c05c204c380402      index.qmd
040000 tree 225106f9d0d05586504d22c93a4416bd915c240a      lectures
100755 blob 0c853ac829af90b6278d3889b3a5b32f989d85b2      make_presentations
100644 blob cb323a6fb614d0866d3543227cf8d465a09866c4      module_description.qmd
100644 blob 4ca0a7a09ac148b6e7874358435ee544d298cfb9      requirements.txt
040000 tree f4787e8e0de9ddb2fd1ede0be8ea5b89688871c7      scripts
040000 tree c2e5f232443da3d38c7d6835b7e06d8efce34466      static
100644 blob b3b03f24940acdb62b0e6f56b47e99403d61cbbc      styles.css
```



blob

```
...
100644 blob 1aecac4d48ad857aeb3badc95b00f562b77b4890    environment.yaml
100644 blob cf8294e57cd9666b6a27aa7864c05c204c380402    index.qmd
040000 tree 225106f9d0d05586504d22c93a4416bd915c240a    lectures
...
```

```
1 !git cat-file -p cf8294e57cd9666b6a27aa7864c05c204c380402
```

```
---
title: "Generic Software Skills"
---
```

```
::: {.callout-tip title="Course registration"}
Please [register online for the course](https://survey.academiccloud.de/index.php/765845).
:::
```

The course will take place during Summer Semester 2024, **Tuesdays** at **13:00** to **15:00** in **room 1536a** in Bundesstraße 55.

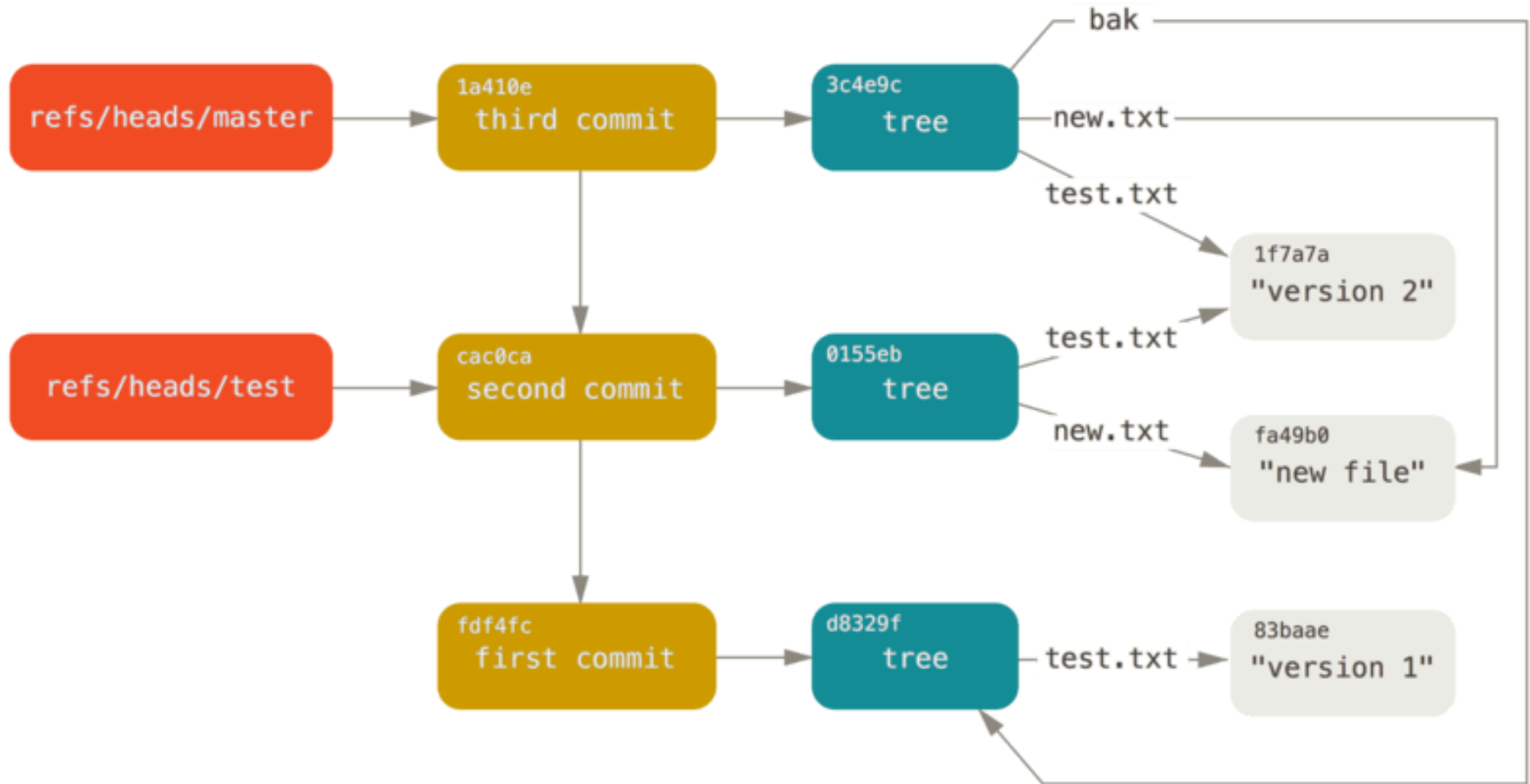
If you have a portable computer at hand, please bring it to the lecture.
If not, don't worry, the lecture will be in a computer room.

Lectures

Date	Title	Lecturers
---	---	---
2024-04-02	The command line	Lukas, Flo



git objects



git objects (from the [git book](#))



git objects

`blob`, `tree`, `commit` (and `tag`) are git objects
internally, **every** object is prefixed by a header:

`<type> <size>\0`

```
1 def make_object_header(otype: str, contents: bytes) -> bytes:  
2     return f"{otype} {len(contents)}".encode("ascii") + b"\0"
```



build a blob

```
1 def make_blob(contents):  
2     return make_object_header("blob", contents) + contents  
3  
4 file_contents = open("hello.txt", "rb").read()  
5 blob = make_blob(file_contents)  
6 print(sha1(blob).hexdigest())
```

```
b6fc4c620b67d95f953a5c1c1230aaab5db5a1b0
```

```
1 !git hash-object hello.txt
```

```
b6fc4c620b67d95f953a5c1c1230aaab5db5a1b0
```



Hands-on: object store

- create a new git repo (`git init <some folder>`)
- take a look into the `.git/objects` directory
- add a blob (file) to the git object store
(`git hash-object -w` or `git add` can help)
- check how `.git/objects` changed



Content Addressable Store in git

- git stores objects in the `.git/objects` directory
- objects are compressed using deflate / zlib
- two digits are used as folder name

`hello` would be stored in

`.git/objects/b6/fc4c620b67d95f953a5c1c1230aaab5db5a1b0`

```
1 print(sha1(make_blob(b"hello")).hexdigest())
```

```
b6fc4c620b67d95f953a5c1c1230aaab5db5a1b0
```



Content Addressable Store in git

```
1 import zlib
2 from pathlib import Path
3 class GitCAS:
4     def __init__(self, gitdir):
5         self._gitdir = Path(gitdir)
6     def put(self, content):
7         ...
8     def get(self, key):
9         path = self._gitdir / "objects" / key[:2] / key[2:]
10        return zlib.decompress(path.read_bytes())
11
12 git = GitCAS(".git")
```

```
1 git.get("b6fc4c620b67d95f953a5c1c1230aaab5db5a1b0")
```

```
b'blob 5\x00hello'
```

be aware of **packfiles**



Hands-on: tree & commit

- create a commit containing your blob in the test repo
- check how `.git/objects` changed
- have a look at the tree and commit objects
- compare the raw object contents to `git cat-file -p`
- to get the raw object, use e.g. GitCAS or

```
1 cat .git/objects/<ha/sh> | pigz --uncompress
```



tree

- Sequence of `<mode> ␣ <name> \0 <hash>`
- Sorted by `name`
- `mode`: “100644” = file, “40000” = directory
- `hash` is stored in binary

```
1 git.get("b219984ac18ed30a40059099f66bb9beba1ea085")
```

```
b'tree 603\x00100644
.gitignore\x00\x8d\xbe\xe0\xdf;>\n\xcf\xcf\x8e\xe0V\x14\x13\x1a\x85f\xcbN!100644 .gitlab-
ci.yml\x00v==I\x97\x86L\x8785\xeb\xbe\xf2\xe0\x00\xd5\xc7:K\x07100644 .pre-commit-
config.yml\x00\x01\xfe\xfe\x8ez\x9a+-VQ\xf8\x8c\xbd=b\xe4\xc3\x86\xd8\x1b100644
LICENSE\x004>\x82, \xbe\xc4\x00\x8eP\xa8\xbev\x85\xe2m\xda:\xdc\xd3\xe9100644
README.md\x00\x8fG\xe14\xc2Yvj\xe0ne>\n\x98\xf7\xda\xdc\x0bV\xed100644
_quarto.yml\x00\x97\xd3\xca\xdd-\xdd\xf6\xa3\xc7LA\xcd\xb1{\x03B\xd0\xa6\x9f100644
environment.yml\x00\x1a\xec\xacMH\xad\x85z\xeb;\xad\xc9[\x00\xf5b\xb7{H\x90100644
index.qmd\x00\xcf\x82\x94\xe5|\xd9fkj\ '\xaaxd\xc0\ L8\x04\x0240000
lectures\x00"Q\x06\xf9\xd0\xd0U\x86PM"\xc9:D\x16\xbd\x91\\$\n100755
make_presentations\x00\x0c\x85:\xc8)\xaf\x90\xb6\ '\x8d8\x89\xb3\xa5\xb3/\x98\x9d\x85\xb2100644
module_description.qmd\x00\xcb2:o\xb6\x14\xd0\x86m5C"|\xf8\xd4e\xa0\x98f\xc4100644
requirements.txt\x00L\xa0\xa7\xa0\x9a\xc1H\xb6\xe7\x87CXC^\xe5D\xd2\x98\xcf\xb940000
scripts\x00\xf4x~\x8e\r\xe9\xdd\xb2\xfd\x1e\xde\x0b\xe8\xea[\x89h\x88q\xc740000
```



static\00\c2\e5\ef22D=\a3\xd3\x8c}h5\xb7\em\x8e\fc\3Df100644 styles.css\00\b3\b0?
\$\x94\n\xcd\xb6+\x0eoV\b4~\x99@=a\cb\bc '



commit

```
tree <tree_hash>
parent <parent_hash>
author Author Name <author.name@email.example> <timestamp> <timezone>
committer Committer Name <committer.name@email.example> <timestamp> <timezone>

<commit message>
```

- `parent` can occur 0 or more times
- `timestamp` is UNIX time (seconds since 1970-01-01)
- hashes are stored in hexadecimal

```
1 print(git.get("d2995d1549878920c372e4af6b1f75367b2095fd").decode("utf-8"))
```

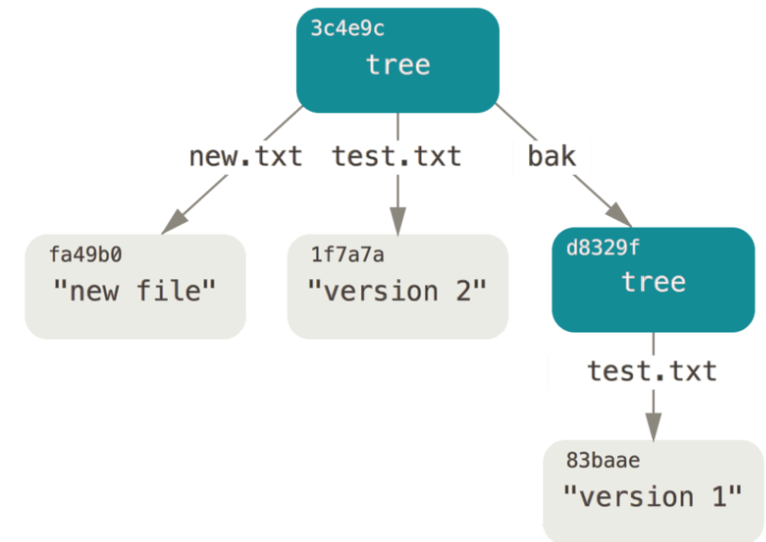
```
commit 268tree b219984ac18ed30a40059099f66bb9beba1ea085
parent 183a295f09e0a5daf4da9fba6c7cd016e7de4546
author Tobias Kölling <tobias.koelling@mpimet.mpg.de> 1712570344 +0200
committer Tobias Kölling <tobias.koelling@mpimet.mpg.de> 1712570344 +0200
```

```
add registration notice
```



Merkle Tree

- blobs, trees and commits form a [Merkle Tree](#).
- the hash of a tree identifies everything below



(from the [git book](#))

Operations



`diff` (the program)

- The `diff` program is part of UNIX since 1974
- computes line-by-line differences of text files
- generate “diff” or “patch” files
- diffs can be applied to other files using `patch`



Hands-on: diff

In the code subdirectory you find 3 files.

1. produce a patch for the `fibonacci.py` file with the changes in `fibonacci_fixed.py` using the `diff` command-line tool.
2. inspect the the patch
3. apply the patch on `golden_ratio.py` using the `patch` command-line tool.



git diff

- extends `diff` to work on entire trees
- the Merkle Tree property helps
- `git diff` files have additional version information



A diff looks like

```
1 git diff
1 diff --git a/hello_world.txt b/hello_world.txt
2 index e13e32c..cd08755 100644
3 --- a/hello_world.txt
4 +++ b/hello_world.txt
5 @@ -1 +1 @@
6 -Hello wordl!
7 +Hello world!
```



cherry-pick

```
1 git cherry-pick <commit>
```

- take the diff of one commit to its parent
- apply it to the current **HEAD** (like **patch** would do)
- keep commit metadata (author, time, message, etc...)

rebase

```
1 git rebase <branch>
```

1. extract all commits in the current branch that are not in `<branch>`
2. makes `<branch>` the **HEAD** of the current branch
3. cherry-pick all the commits extracted in 1.

`git rebase -i` (interactive)

shows all planned pick commands for modification



Take home messages

- branches are cheap
- git is different but not crazy
- good data structures can do a lot



Shotgun buffet



index

- the staging area
- a virtual worktree
- a file
 - containing references to blobs
 - which should end up in a tree for a commit



packfiles

- We only covered *loose objects* in the git object store
- There are also *packfiles*, which combine multiple objects in one file to improve performance
- Packfiles are created by `push`, `fetch`, `gc` and some other commands
- They don't change object store semantics

[a nice technical overview](#)



tag

- just another **ref**, stored in `.git/refs/tags`
- lightweight tag: ref to a `commit` object
- annotated tag: ref to a `tag` object with more information
 - the `tag` object then points to a `commit`



... and what's HEAD?

```
1 !cat .git/HEAD
```

```
ref: refs/heads/main
```

a pointer to a **ref** (the current one)

(except if HEAD is detached, then it points directly to a commit)

references

- [git repository layout](#)
- [git internals: objects](#)

