GIT INTERNALS

PART 1: OBJECTS

OVERVIEW

- Motivation
- Objects
  - 3 Types
- Two new git commands
- DEMO

MOTIVATION

> *We found that understanding this information is **fundamentally important to appreciating how useful and powerful Git is**, others argue that it's confusing and unnecessarily complex for beginners. ... We leave it up to you to decide.*

— 10.1 Git Internals - Plumbing and Porcelain

MOTIVATING QUESTIONS

How does `git` *really* work?

How does `git` reconstruct a project with only a commit hash?

How do `git` repositories *not* take up a lot of disk space?

OBJECTS

WHAT IS GIT?

Two unconventional definitions:

1. a content-addressable filesystem
2. a simple key-value data store

WHAT ARE THE VALUES AND KEYS?

Values are **objects**.

Keys are **hashes**.

Objects are **immutable**.

`git` frequently provides information about objects!

CLONING

```
$ git clone git@github.com:gbroques/prop-types.git
Cloning into 'prop-types'...
remote: Enumerating objects: 6125, done.
remote: Counting objects: 100% (6125/6125), done.
remote: Compressing objects: 100% (3578/3578), done.
remote: Total 6125 (delta 2500), reused 5704 (delta 2238)
Receiving objects: 100% (6125/6125), 40.72 MiB | 592.00 KiB/s,
Resolving deltas: 100% (2500/2500), done.
Updating files: 100% (2951/2951), done.
```

PUSHING

```
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 1023 bytes | 511.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
To git@github.com:gbroques/prop-types.git
        8fd6b59..4e7ce21  master -> master
```

PULLING

```
$ git pull
remote: Enumerating objects: 112, done.
remote: Counting objects: 100% (101/101), done.
remote: Compressing objects: 100% (71/71), done.
remote: Total 71 (delta 52), reused 0 (delta 0)
Unpacking objects: 100% (71/71), 9.84 KiB | 13.00 KiB/s, done.
From git@github.com:gbroques/prop-types.git
        f1c0c2a3..3f4264d6  release/1.0             -> origin/
        * [new branch]        feature/adding-use-claims-map ->
        * [new branch]        feature/cleaning-up-app -> origi
        * [new branch]        feature/typography-font-size ->
        2f0c2b27..ef61c43d  feature/updating-url-in-commands -
        * [new branch]        release/1.1             -> origi
Updating f1c0c2a3..3f4264d6
```

3 OBJECT TYPES

1. 

— file

2. 

— directory

3. 

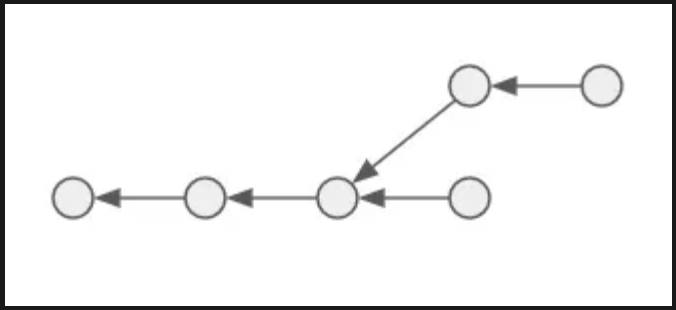
— pointer to a tree

OBJECT HASHING & STORAGE

Stored in the object database (`.git/objects/`)

Each object has a unique **40-char** SHA-1 hash

```
.git/objects
├── 0f/
│   └── 6a4e6354e536514412ba38ab50ad6264a4d323  # 38 char file
├── 2d/
│   ├── 0e74b07052fd2d44788bb3cced2923e56786bb
│   └── 2bd6b04c3e39b0c611ada8b0cbd12e64f46c11
├── dd/
│   └── f93dd01ed0c56b50f98ef7c4773a81a5a17a5b
```



Objects form a directed acyclic graph.

TWO NEW GIT COMMANDS

1. `git hash-object`
2. `git cat-file`

`git` **`hash-object`**

generate the **hash** of an **object** from its *content*

Reference

HASH-OBJECT EXAMPLE

```
$ echo "text" | git hash-object --stdin
8e27be7d6154a1f68ea9160ef0e18691d20560
```

```
git cat-file -p <hash>
```

pretty print an object

Reference

```
git cat-file -t <hash>
```

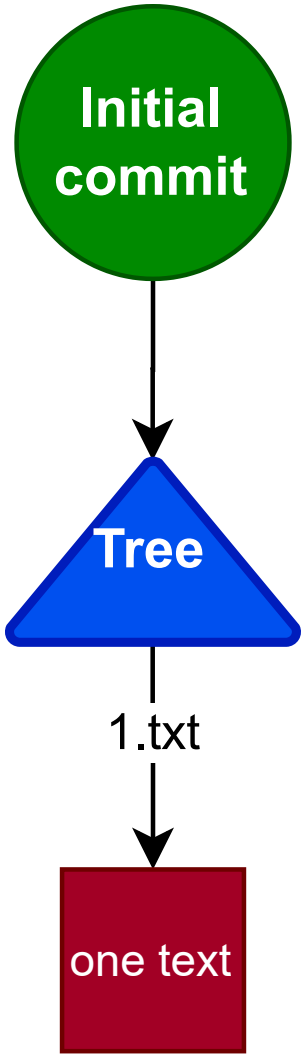
output the **type** of an object

Reference

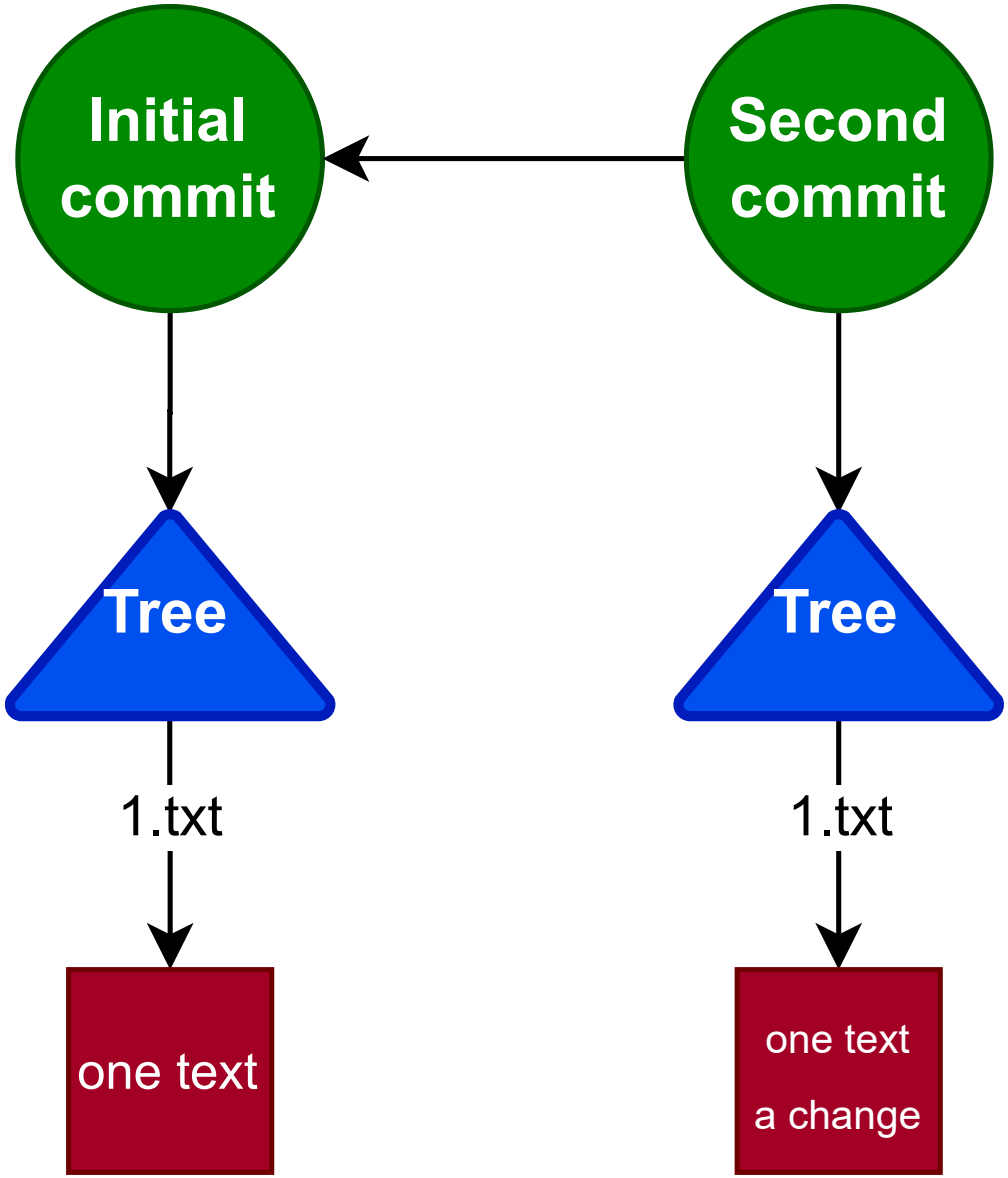

blob
?
tree
?
commit
?

DEMO!

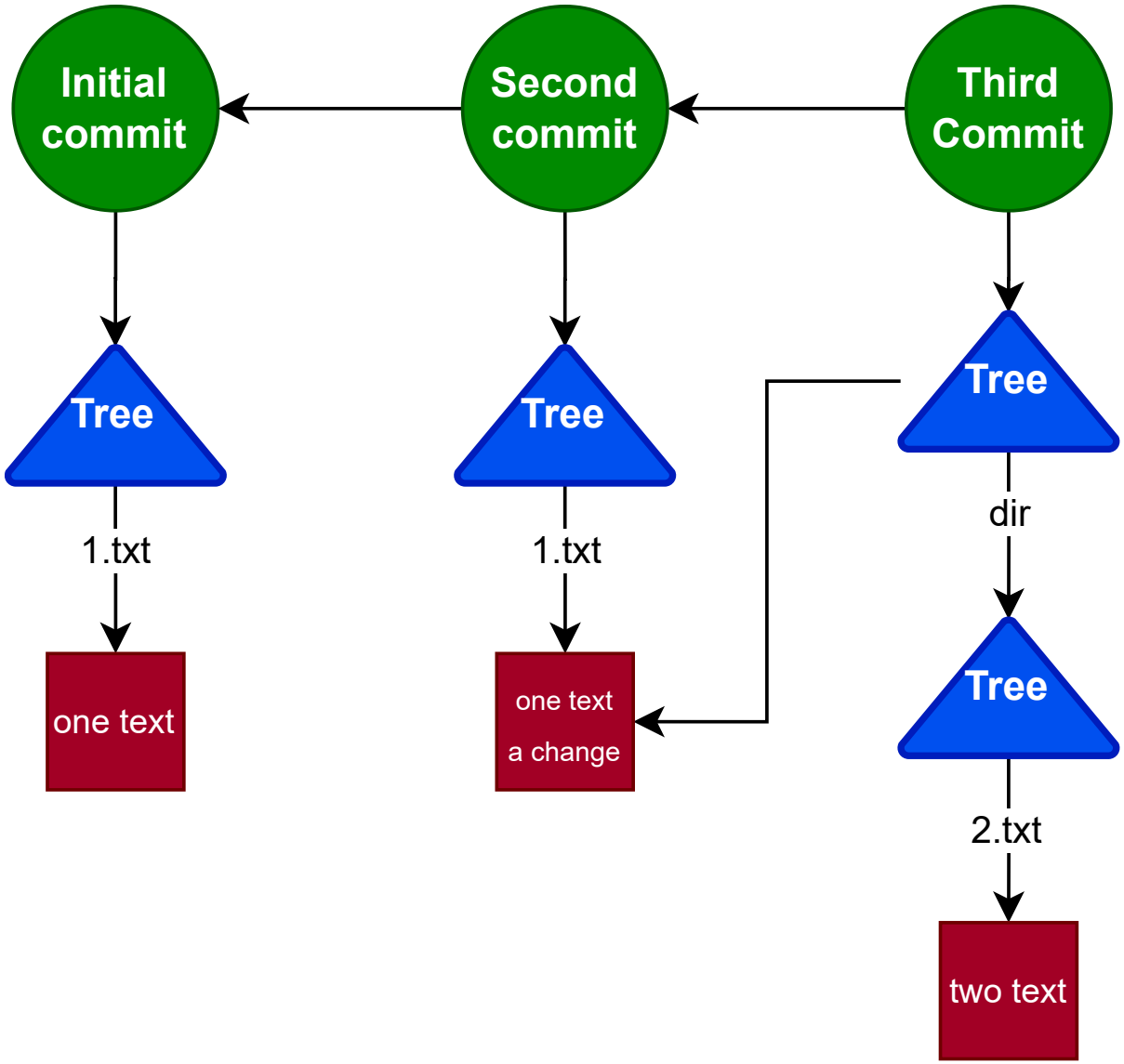
```
$ echo "one text" > 1.txt
$ git add .
$ git commit -m "Initial commit"
```


Initial commit
Tree
1.txt
one text

```
$ echo "a change" >> 1.txt
$ git add .
$ git commit -m "Second commit"
```


Initial commit
Second commit
Tree
Tree
1.txt
1.txt
one text
one text
a change

```
$ mkdir dir
$ echo "two text" > dir/2.txt
$ git add .
$ git commit -m "Third commit"
```


Initial commit
Second commit
Third Commit
Tree
Tree
Tree
1.txt
1.txt
dir
one text
one text
a change
Tree
2.txt
two text

CURRENT STATE

```
demo/
├── 1.txt # "one text\na change"
└── dir/
    └── 2.txt # "two text"
```

If I made a **fourth commit**:

- adding a file — `3.txt`
- with the content "`one text`"

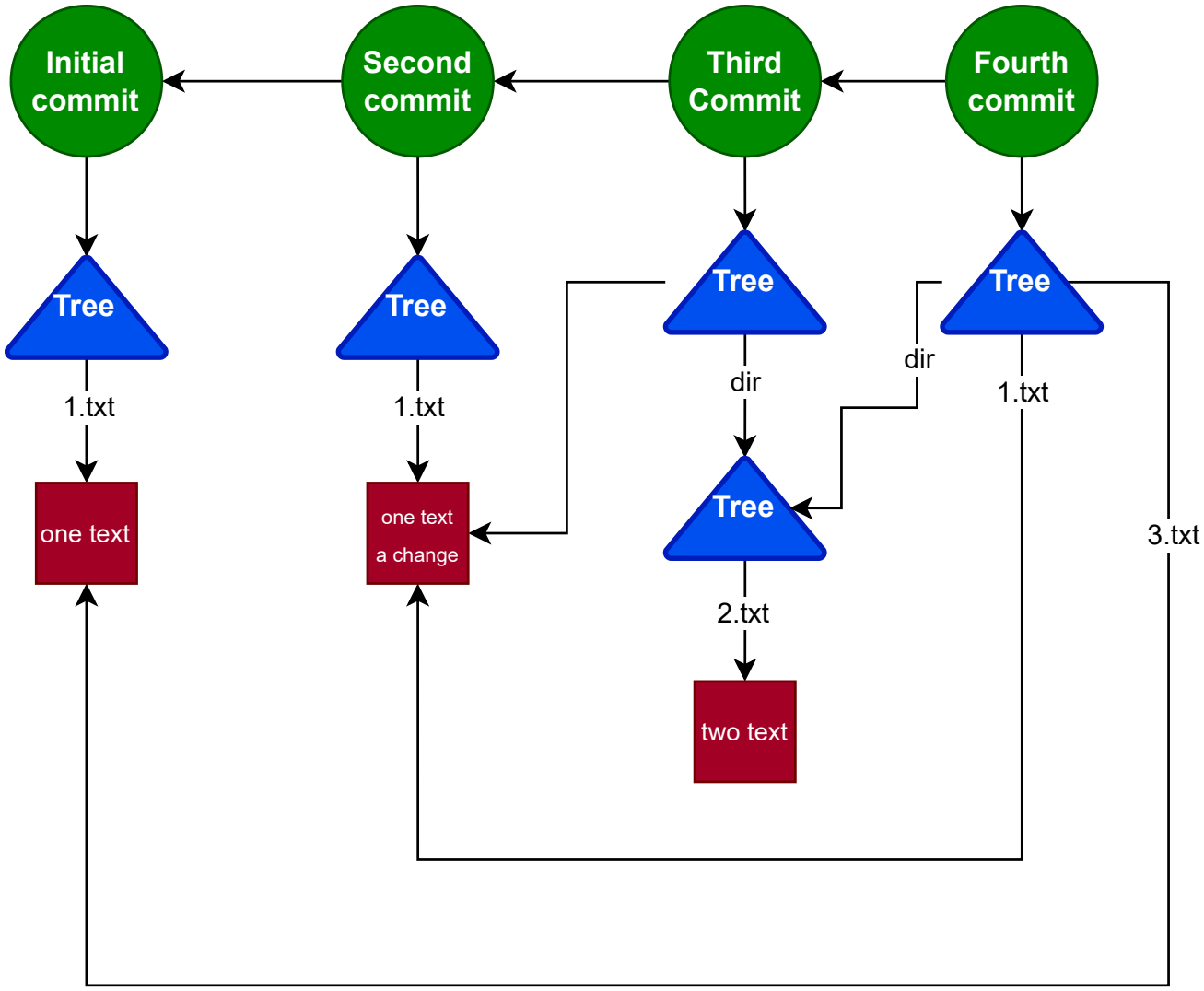
then **how many new objects would be created?**

```
demo/
├── 1.txt # "one text\na change"
├── 3.txt # "one text"
└── dir/
    └── 2.txt # "two text"
```

2


commit
tree

```
$ echo "one text" > 3.txt
$ git add .
$ git commit -m "Fourth commit"
```


Initial commit
Second commit
Third Commit
Fourth commit
Tree
Tree
Tree
Tree
1.txt
1.txt
dir
dir
1.txt
one text
one text
a change
Tree
3.txt
2.txt
two text

CONCLUSION

Git is a content-addressable filesystem, and a *simple* key-value store, built on:

- 

Blobs — like files
- 

Trees — like directories
- 

Commits — *pointers to trees* (with meta-data)

and `git` relates these **objects** together in a **graph structure** on disk.

ADDITIONAL RESOURCES

- https://git-scm.com/book/en/v2/Git-Internals-Plumbing-and-Porcelain
- GIT Internals (Part 1: Architecture and Objects)
- YouTube: Git Internals - How Git Works - Fear Not The SHA!

BONUS TOPICS

FOURTH TYPE OF GIT OBJECT

lightweight versus annotated tags

Reference

EMPTY COMMITS

```
git commit --allow-empty -m "Empty commit"
```

Reference

Node.js implementation of `git cat-file` and `git hash-object`:

https://github.com/gbroques/zlib-git-nodejs

PART 2: HEAD, BRANCHES, AND TAGS