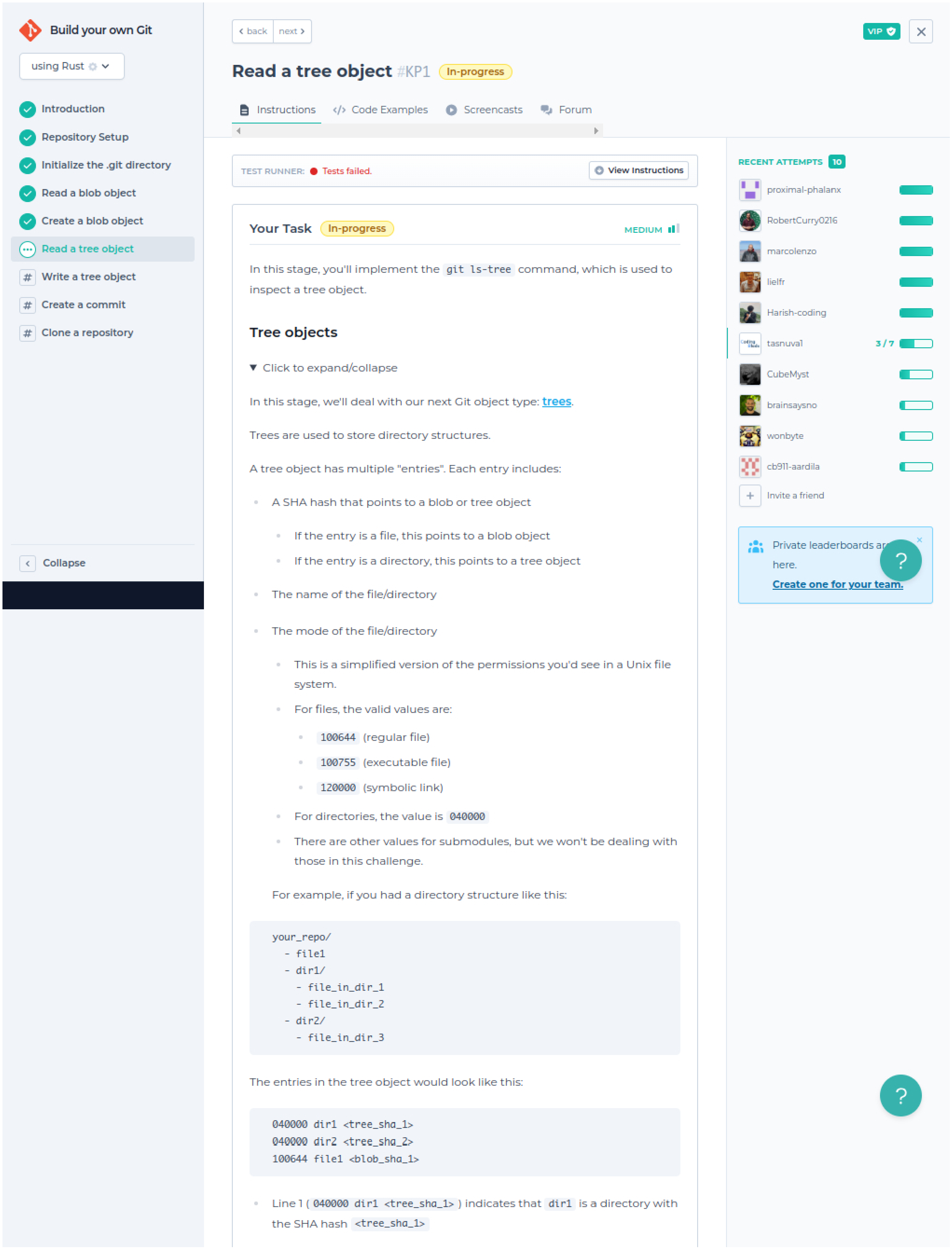

Build your own Git

using Rust

- ✓ Introduction
- ✓ Repository Setup
- ✓ Initialize the .git directory
- ✓ Read a blob object
- ✓ Create a blob object
- Read a tree object
- # Write a tree object
- # Create a commit
- # Clone a repository

Collapse

‹ back next ›

VIP

Read a tree object #KP1 In-progress

Instructions · Code Examples · Screencasts · Forum

TEST RUNNER: ● Tests failed. View Instructions

Your Task In-progress

MEDIUM

In this stage, you'll implement the `git ls-tree` command, which is used to inspect a tree object.

Tree objects

▼ Click to expand/collapse

In this stage, we'll deal with our next Git object type: trees.

Trees are used to store directory structures.

A tree object has multiple "entries". Each entry includes:

- A SHA hash that points to a blob or tree object
  - If the entry is a file, this points to a blob object
  - If the entry is a directory, this points to a tree object
- The name of the file/directory
- The mode of the file/directory
  - This is a simplified version of the permissions you'd see in a Unix file system.
  - For files, the valid values are:
    - `100644` (regular file)
    - `100755` (executable file)
    - `120000` (symbolic link)
  - For directories, the value is `040000`
  - There are other values for submodules, but we won't be dealing with those in this challenge.

For example, if you had a directory structure like this:

```
your_repo/
  - file1
  - dir1/
    - file_in_dir_1
    - file_in_dir_2
  - dir2/
    - file_in_dir_3
```

The entries in the tree object would look like this:

```
040000 dir1 <tree_sha_1>
040000 dir2 <tree_sha_2>
100644 file1 <blob_sha_1>
```

- Line 1 (`040000 dir1 <tree_sha_1>`) indicates that `dir1` is a directory with the SHA hash `<tree_sha_1>`

RECENT ATTEMPTS 10

proximal-phalanx	
RobertCurry0216	
marcolenzo	
lielfr	
Harish-coding	
tasnuval	3 / 7
CubeMyst	
brainsaysno	
wonbyte	
cb911-aardila	

Invite a friend

Private leaderboards ar… here.
Create one for your team.

- Line 2 (`040000 dir2 <tree_sha_2>`) indicates that `dir2` is a directory with the SHA hash `<tree_sha_2>`
- Line 3 (`100644 file1 <blob_sha_1>`) indicates that `file1` is a regular file with the SHA hash `<blob_sha_1>`

  `dir1` and `dir2` would be tree objects themselves, and their entries would contain the files/directories inside them.

The `ls-tree` command

▼ Click to expand/collapse

The `git ls-tree` command is used to inspect a tree object.

For a directory structure like this:

```
your_repo/
  - file1
  - dir1/
    - file_in_dir_1
    - file_in_dir_2
  - dir2/
    - file_in_dir_3
```

The output of `git ls-tree` would look like this:

```
$ git ls-tree <tree_sha>
040000 tree <tree_sha_1>    dir1
040000 tree <tree_sha_2>    dir2
100644 blob <blob_sha_1>    file1
```

Note that the output is alphabetically sorted, this is how Git stores entries in the tree object internally.

In this stage you'll implement the `git ls-tree` command with the `--name-only` flag. Here's how the output looks with the `--name-only` flag:

```
$ git ls-tree --name-only <tree_sha>
dir1
dir2
file1
```

The tester uses `--name-only` since this output format is easier to test against.

We recommend implementing the full `ls-tree` output too since that'll require that you parse all data in the tree object, not just filenames.

Tree Object Storage

▼ Click to expand/collapse

Just like blobs, tree objects are stored in the `.git/objects` directory. If the hash of a tree object is `e88f7a929cd70b0274c4ea33b209c97fa845fdbc`, the path to the object would be `./git/objects/e8/8f7a929cd70b0274c4ea33b209c97fa845fdbc`.

The format of a tree object file looks like this (after Zlib decompression):

```
tree <size>\0
<mode> <name>\0<20_byte_sha>
<mode> <name>\0<20_byte_sha>
```

(The above code block is formatted with newlines for readability, but the actual file doesn't contain newlines)

- The file starts with `tree <size>\0`. This is the "object header", similar to what we saw with blob objects.

- After the header, there are multiple entries. Each entry is of the form `<mode> <name>\0<sha>`.
  - `<mode>` is the mode of the file/directory (check the previous section for valid values)
  - `<name>` is the name of the file/directory
  - `\0` is a null byte
  - `<20_byte_sha>` is the 20-byte SHA-1 hash of the blob/tree (this is **not** in hexadecimal format)

You can read more about the internal format of a tree object here.

Tests

The tester will use your program to initialize a new repository:

```
$ mkdir test_dir && cd test_dir
$ /path/to/your_git.sh init
```

It'll then write a tree object to the `.git/objects` directory.

It'll then run your program like this:

```
$ /path/to/your_git.sh ls-tree --name-only <tree_sha>
```

It'll verify that the output of your program matches the contents of the tree object.

For a directory structure like this:

```
your_repo/
  - file1
  - dir1/
    - file_in_dir_1
    - file_in_dir_2
  - dir2/
    - file_in_dir_3
```

The output expected is:

```
dir1
dir2
file1
```

Notes

- In a tree object file, the SHA hashes are not in hexadecimal format. They're just raw bytes (20 bytes long).
- In a tree object file, entries are sorted by their name. The output of `ls-tree` matches this order.

View Code Examples | View Screencasts | Test Cases

Collapse ↑

Hints

Filter by Rust

Write | Preview

Found an interesting resource? Share it with the community.

Markdown supported.

Comment

rohitpaulk staff • 1 year ago

Ben Hoyt's pygit article has a section on writing tree objects:
https://benhoyt.com/writings/pygit/#committing

9 reply

?

● Tests failed. Show logs