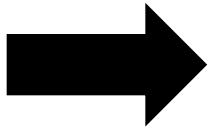
UNDERSTANDING GIT

Joe Gibson (SSCO/HFCS)

joseph.gibson@nasa.gov

Git…

- Is frustrating
- Is complicated
- Gets in your way
- Produces grey hair and stress



- Is intuitive
- Is easy
- Guides development
- Produces happy developers

Complaints/Concerns/Critiques

- **“It’s going to take me months to learn Git!”**
  - Yes.
- **“Git sucks, let’s move back to SVN/CVS/carrier-pigeon”**
  - Let’s not be rash!
- **“Git seems like a cult... Do we have to drink some sort of poisoned fruit punch?**
  - Nope, no punch.
- **“What is the difference between a fetch and a merge and a pull and what the hell does checkout actually do???”**
  - Glad you asked...

Understanding Git

- Requires:
  - ☑ Knowledge of the basic Git commands
  - ☑ Knowledge of the basic workflow
  - ☑ One-on-one training (please continue to ask us!)
  - ☐ ***Knowledge of how Git really works under the hood***
    - **i.e. What is Git and what is a Git repository?**

What is Git?

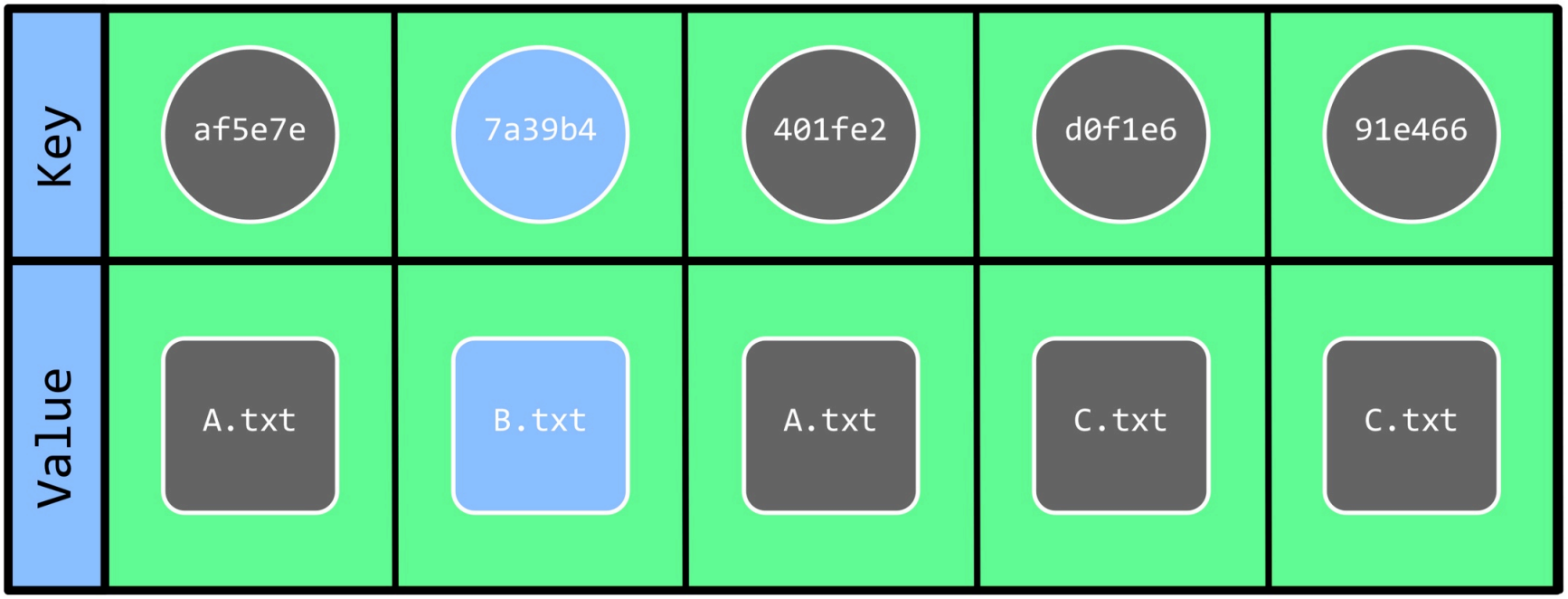
- Git is a **Content Addressable File System**

- The content (i.e. files) is stored as **(key, value)** pairs, where the key (a SHA-1 hash) is derived from the content itself

Write


A.txt
hash-object
af5e7e

Read


Key
af5e7e
7a39b4
401fe2
d0f1e6
91e466
Value
A.txt
B.txt
A.txt
C.txt
C.txt

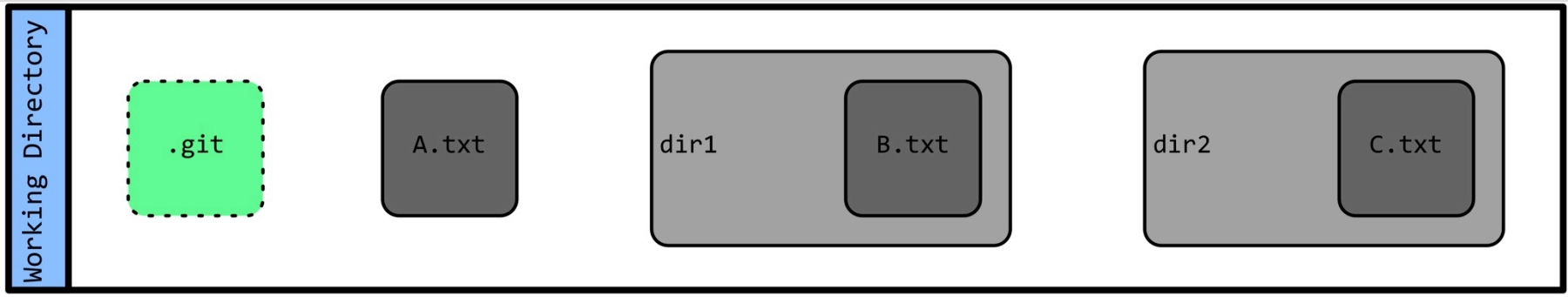
What is a Git repository?

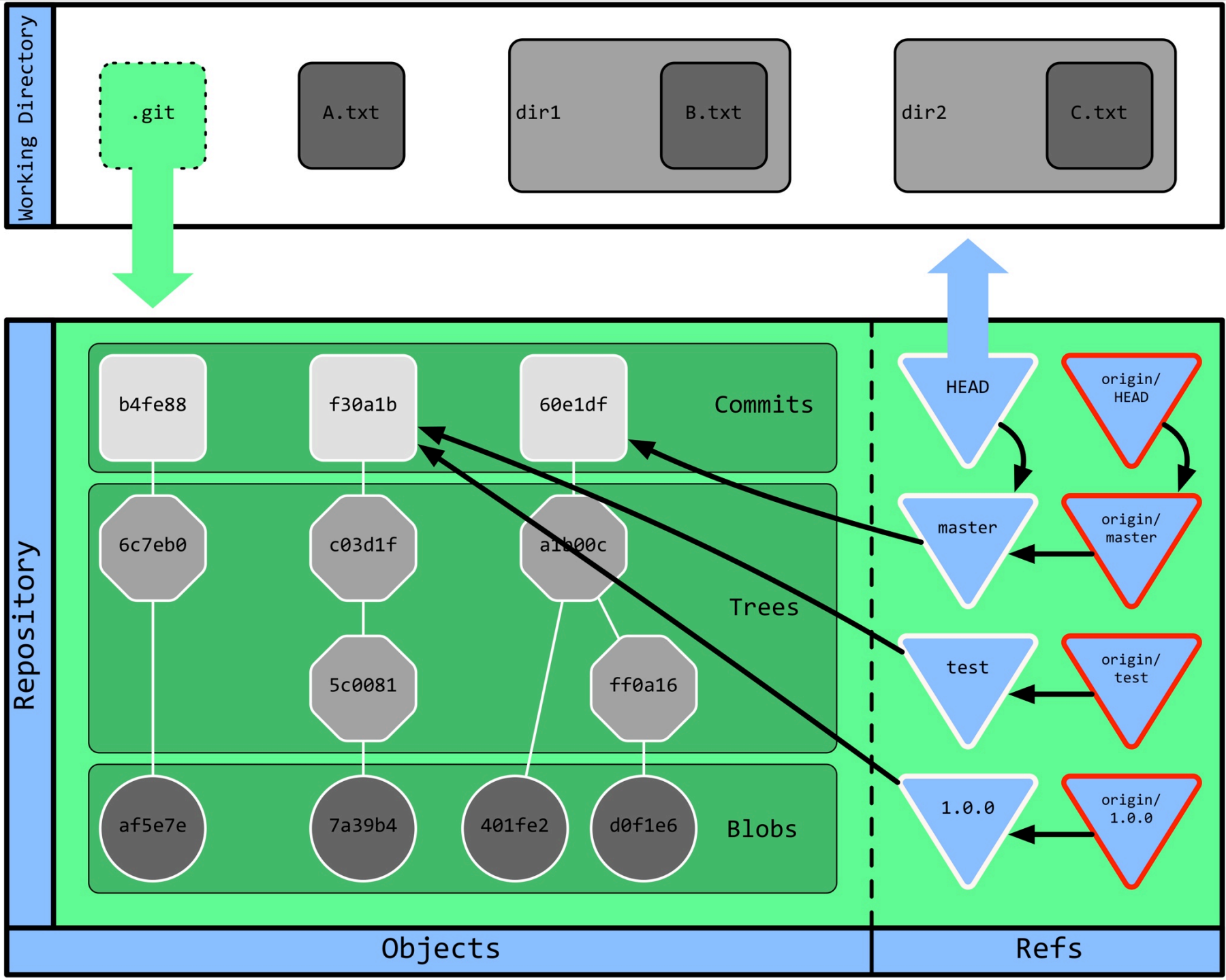
- Each directory we consider a repository has a **.git** folder

- The contents of the **.git** directory is really the **repository**, and the files at the top level constitute the **working directory/working tree**

What is a Git repository?

- The **.git** folder contains:
  - **References** (Refs)
  - **Objects**
    - **Commits**
    - **Trees**
    - **Blobs**


Working Directory
.git
A.txt
dir1
B.txt
dir2
C.txt


Working Directory
.git
A.txt
dir1
B.txt
dir2
C.txt
Repository
b4fe88
f30a1b
60e1df
Commits
6c7eb0
c03d1f
a1b00c
Trees
5c0081
ff0a16
af5e7e
7a39b4
401fe2
d0f1e6
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
origin/
test
1.0.0
origin/
1.0.0
Objects
Refs

```
$ tree -a
.
├── .git
│   ├── COMMIT_EDITMSG
│   ├── HEAD
│   ├── ORIG_HEAD
│   ├── config
│   ├── description
│   ├── hooks
│   ├── index
│   ├── info
│   │   └── exclude
│   ├── logs
│   │   ├── HEAD
│   │   └── refs
│   │       └── heads
│   │           ├── master
│   │           └── test
│   ├── objects
│   │   ├── 34
│   │   │   └── 5e6aef713208c8d50cdea23b85e6ad831f0449
│   │   ├── 3e
│   │   │   └── 6d9614ec3cfc76e440aca6dafa8e6e0c14f34a
│   │   ├── 43
│   │   │   └── 222cd2e96acac8c83f5162226721f3351c2208
│   │   ├── 46
│   │   │   └── 2221e11fd20cc6dc0203c38fe85b88047721d0
│   │   ├── info
│   │   └── pack
│   └── refs
│       ├── heads
│       │   ├── master
│       │   └── test
│       └── tags
│           └── 1.0.0
├── A.txt
├── dir1
│   └── B.txt
└── dir2
    └── C.txt
```

Refs

- **remotes**
- **branches** (i.e. ‘**heads**’ of branches)
- **tags**
- Either way... They’re just pointers!

```
$ tree .git/refs/
refs/
├── heads
│   ├── geons
│   └── master
├── remotes
│   └── origin
│       ├── geons
│       ├── HEAD
│       └── master
└── tags
```

Branches

- **Branches** point to the latest **commit** in a series of **commits**, i.e. ‘head’ of commit chain

```
$ cat .git/refs/heads/geons
c74945f3f0878aadb4a6f7126c9cd414fe9569b4

$ git cat-file -p c74945
tree c96564d63ad92d90d5225151e7b64a5994ba4fc9
parent cd4f876ba710b2fc48b414336cdd0d30781a3bee
author Joe Gibson <joseph.gibson@nasa.gov> 1469022963 -0400
committer Joe Gibson <joseph.gibson@nasa.gov> 1469022963 -0400

Remove GEONS CCSDS binary file and update CMakeList
```

HEAD

- **HEAD** is a special **ref**
  - Points to another **ref** (usually)
  - **Always** ends up pointing to the commit from which the **working directory** is populated

- **HEADs** point to **refs**
- **Refs** point to **commits**

HEADs and Refs

```
$ cat .git/HEAD
ref: refs/heads/master

$ cat .git/refs/heads/master
cd4f876ba710b2fc48b414336cdd0d30781a3bee

$ cat .git/refs/remotes/origin/HEAD
ref: refs/remotes/origin/master

$ cat .git/refs/remotes/origin/master
cd4f876ba710b2fc48b414336cdd0d30781a3bee
```

Objects

- **Commits**
- **Trees**
- **Blobs**

Commits

- **Tree SHA-1**
- **Parent Commit SHA-1**
- **Author/Committer Information**
- **Blank Line**
- **Commit Message**

```
$ git cat-file -p cd4f876
tree 723a574a17275f194faef6de2c8a2884fd320a37
parent f6e5d7fd3dcf3b1c483feb56ed00113ccf809827
author Joe Gibson <joseph.gibson@nasa.gov> 1469022841 -0400
committer Joe Gibson <joseph.gibson@nasa.gov> 1469022841 -0400

Roll back to using OS_Stat instead of CFS_IsValidFilename
```

Trees

- Basically a directory
- **Sub-tree SHA-1s**
- **Blob SHA-1s**

```
$ git cat-file -p 723a57
100644 blob 73322dc022ec7ecb80b254824756b0b8a94e4dca CMakeLists.txt
040000 tree f15e3959143fa3c7b416e0797ad3eb4328900d7e fsw
```

Blobs

- **Content!**

```
$ git cat-file -p 73322dc

cmake_minimum_required(VERSION 2.6.4)
project(FC CXX)

# Force all c files to be compiled with g++
file(GLOB_RECURSE C_FILES ${CMAKE_CURRENT_SOURCE_DIR} *.c)
set_source_files_properties(${C_FILES} PROPERTIES LANGUAGE CXX)

include_directories(fsw/src)
...
```

Blobs

- As a side note, the SHA-1 hash for the blob is created by concatenating and hashing the following string:
  - The word "**blob**"
  - A space
  - The size of the file in bytes
  - A null byte (**\0**)
  - The full contents of the file

- Note that the filename is **NOT** used in calculating the SHA-1...
  - This makes it easier to do things like renaming, moving, and reducing the total number of blobs

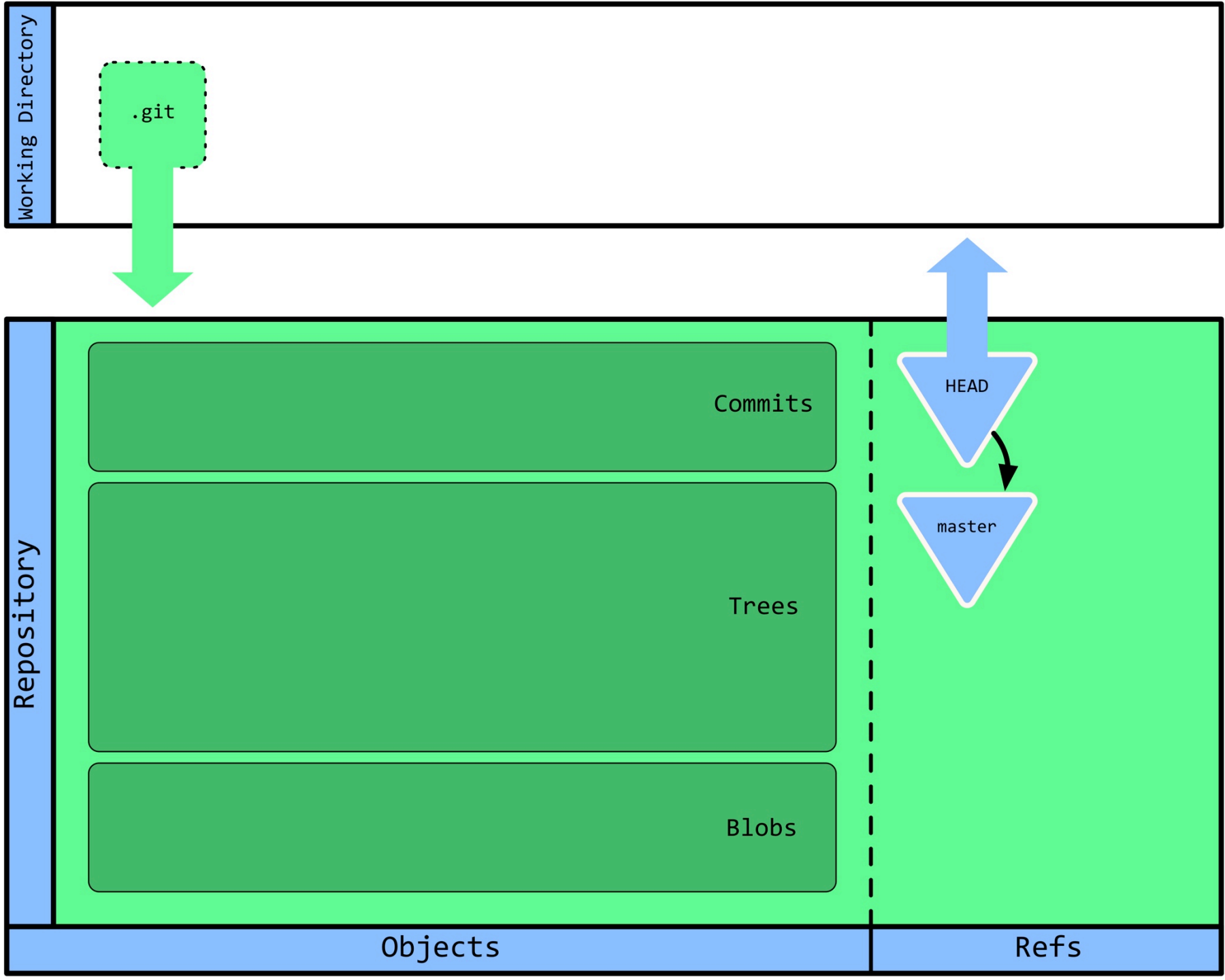
```
$ cat C.txt
Test
$ git hash-object C.txt
345e6aef713208c8d50cdea23b85e6ad831f0449
$ echo -e "blob $(wc -c C.txt | awk '{print $1}')\0$(cat C.txt)" | shasum
345e6aef713208c8d50cdea23b85e6ad831f0449
$ echo -e "blob 5\0Test" | shasum
345e6aef713208c8d50cdea23b85e6ad831f0449
```

Okay, so what?

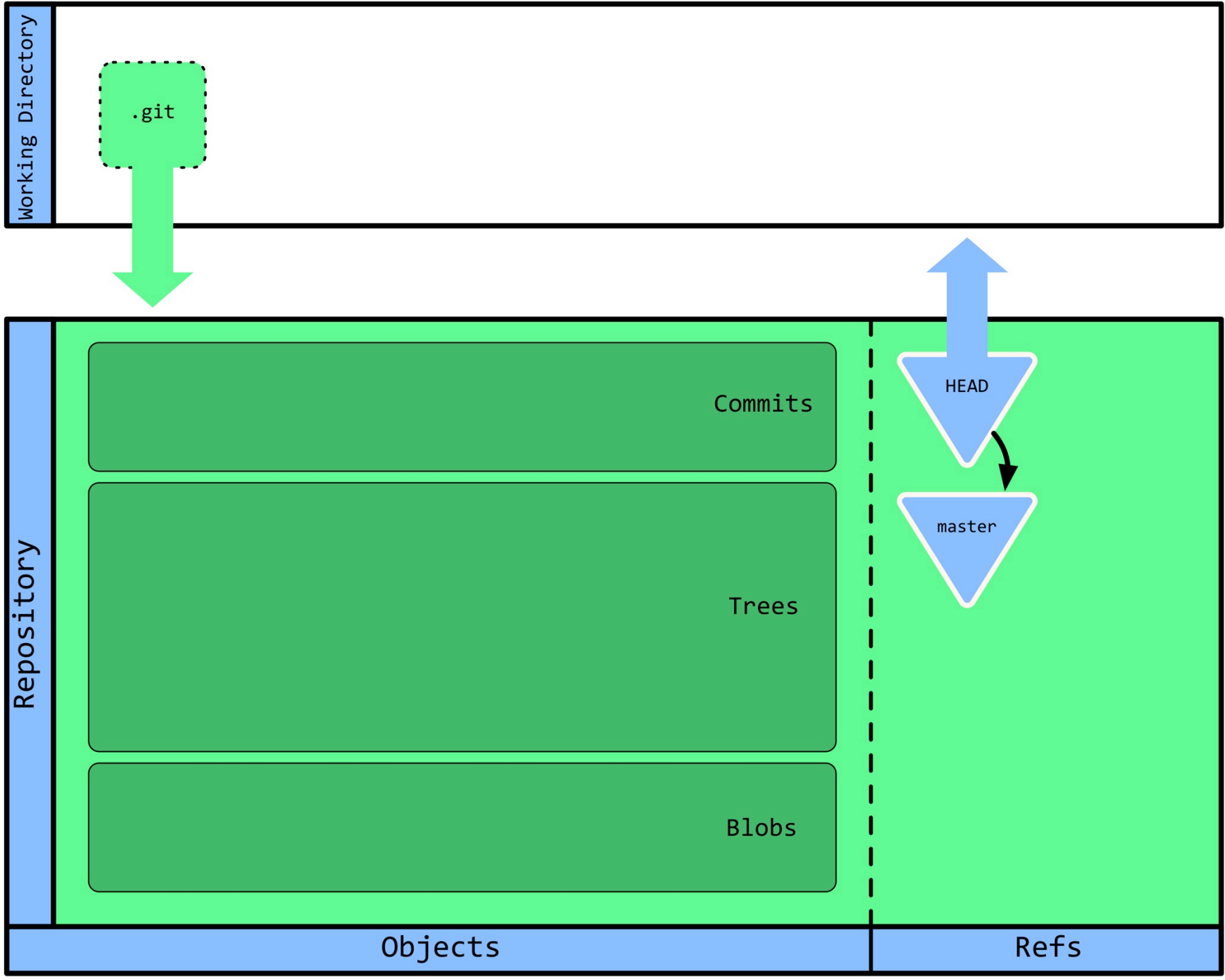
A Detailed Walkthrough…

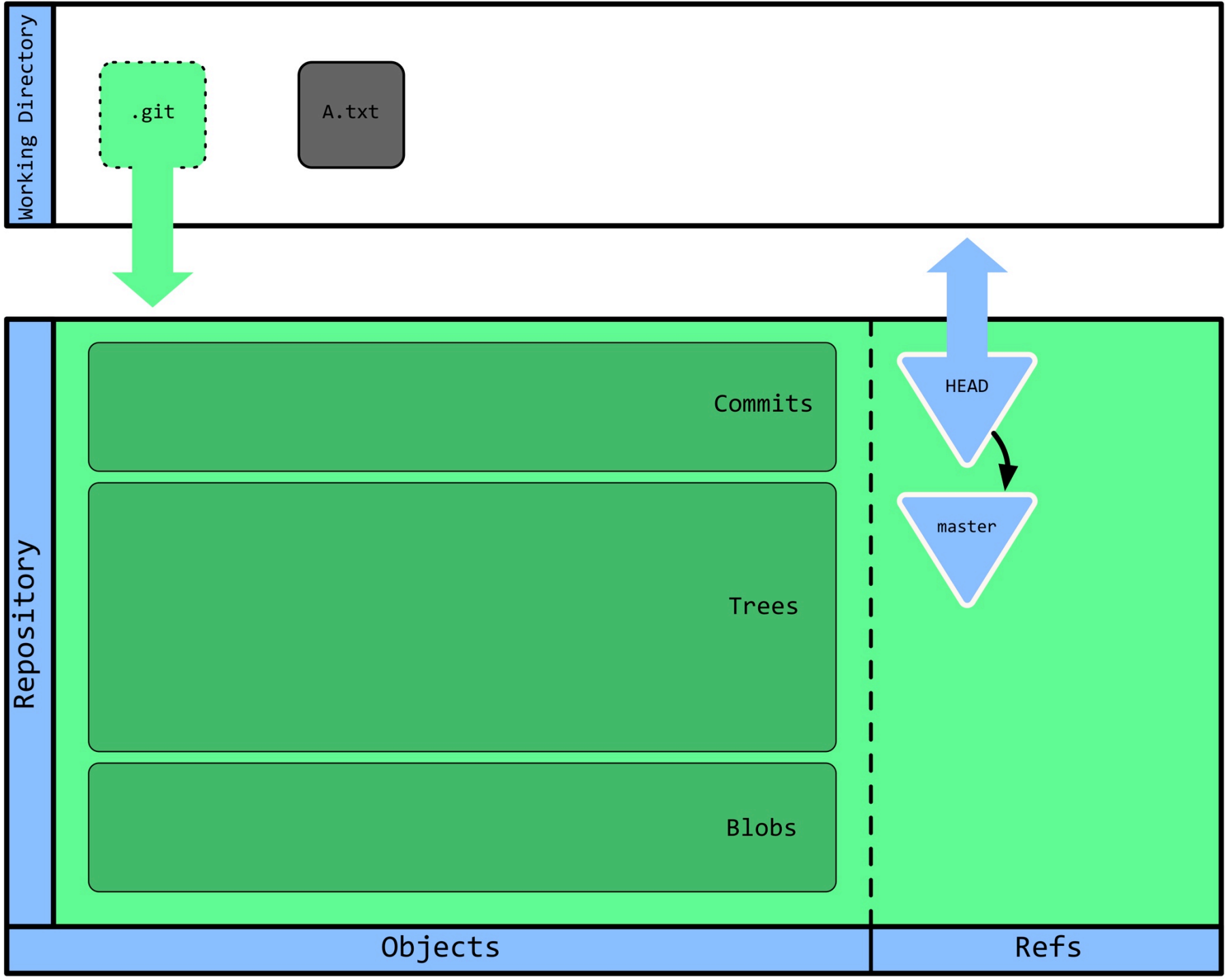
- Let's take a look at the following Git operations in gory detail:
  - **git add** a new file
  - **git commit** the new file
  - **git remote add** a remote called **origin**
  - **git push** to the remote
  - **git branch** to a new branch called **test**
  - **git checkout** the new branch
  - **git add** a new directory and new file
  - **git commit** the new directory and file
  - **git merge** into **master**
  - **git tag** the commit
  - **git push** the changes to the remote
  - **git fetch** for new changes
  - **git merge** with the remote branch to update ours
  - **git checkout** a previous commit

We'll start with an empty repo

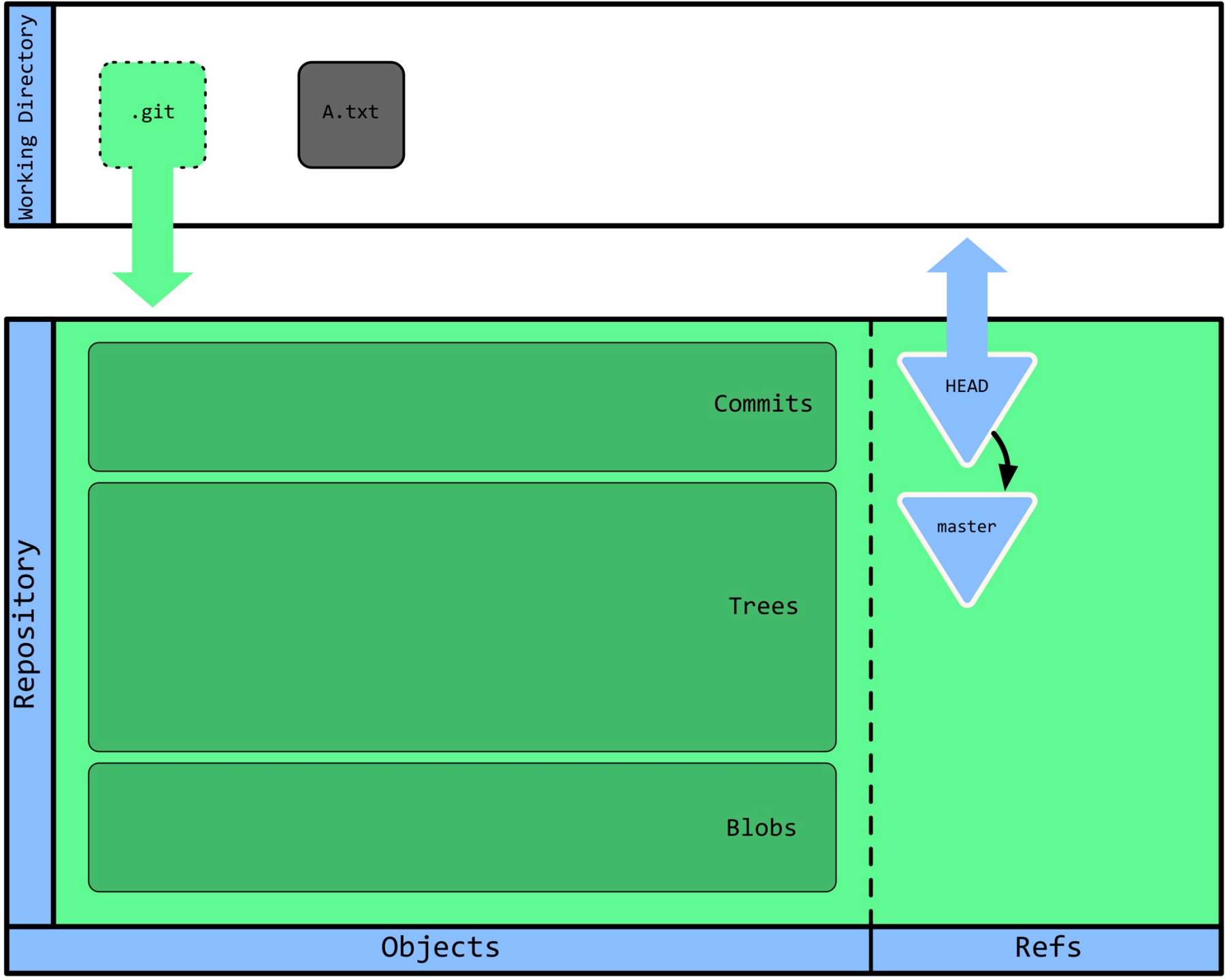

Working Directory
.git
Repository
Commits
Trees
Blobs
HEAD
master
Objects
Refs

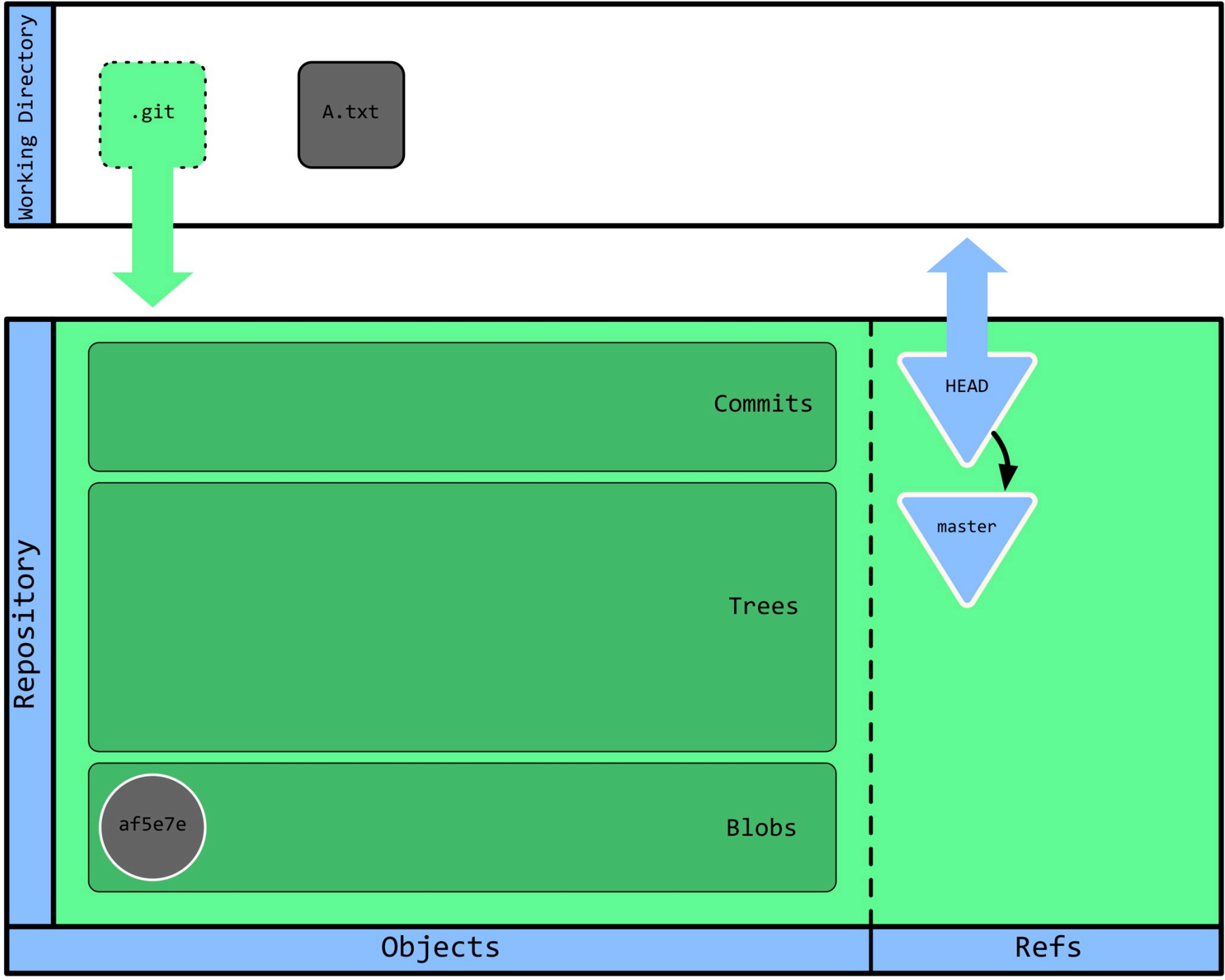
We create a file: `A.txt`


Working Directory
.git
Repository
Commits
Trees
Blobs
HEAD
master
Objects
Refs

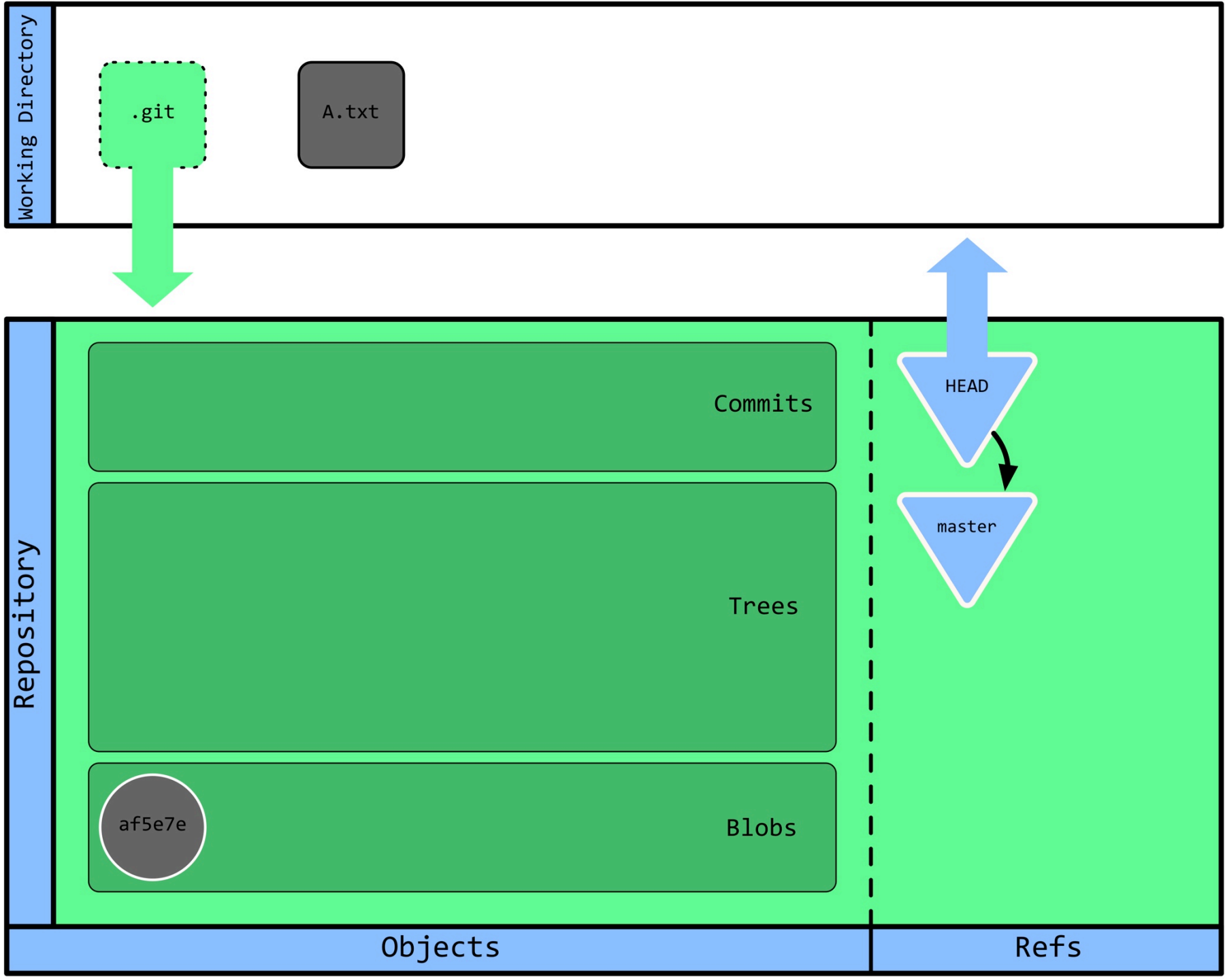

Working Directory
.git
A.txt
Repository
Commits
Trees
Blobs
HEAD
master
Objects
Refs

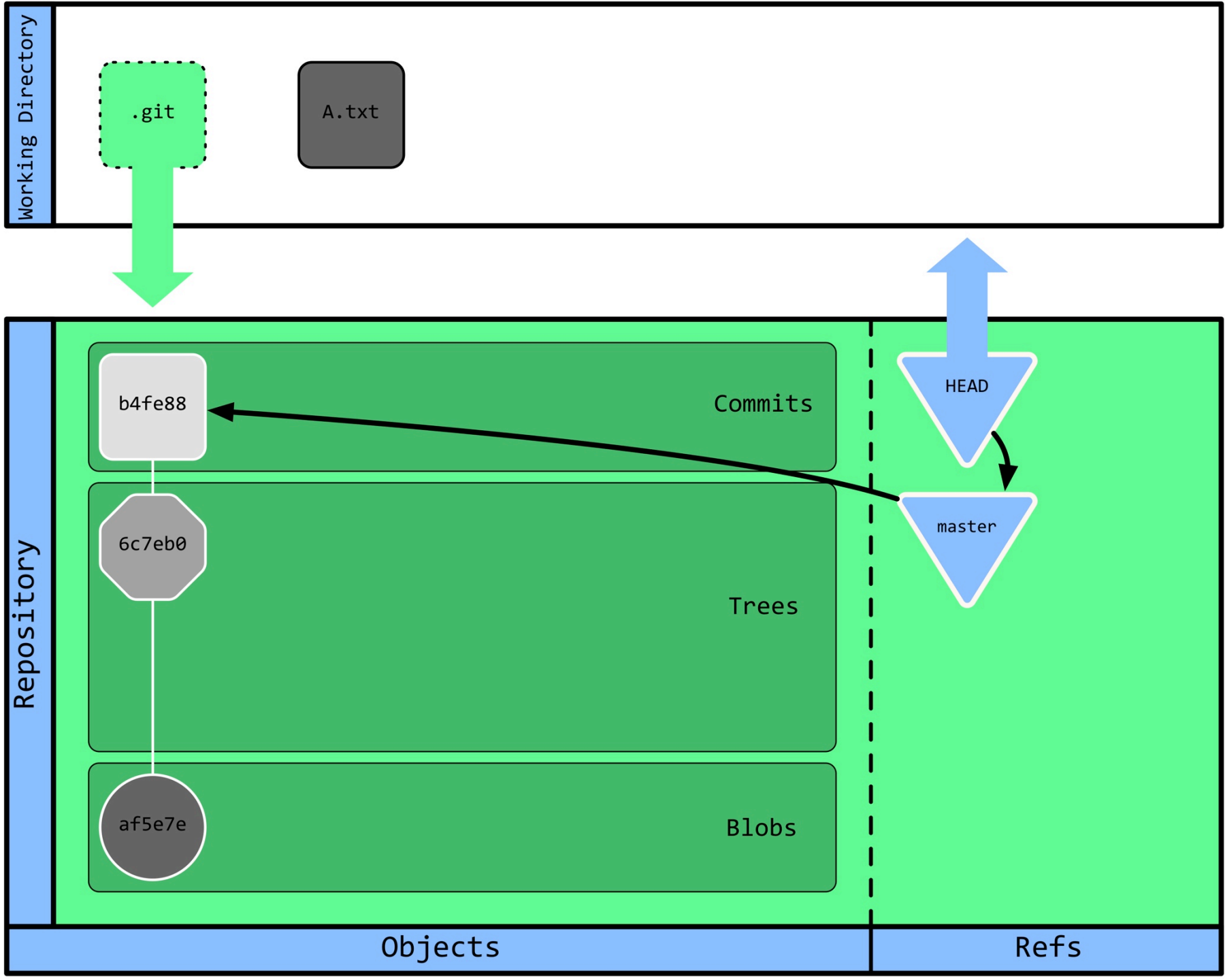
```
git add A.txt
```


Working Directory
.git
A.txt
Repository
Commits
Trees
Blobs
HEAD
master
Objects
Refs


Working Directory
.git
A.txt
Repository
Commits
Trees
af5e7e
Blobs
HEAD
master
Objects
Refs

```
git commit -m “...”
```

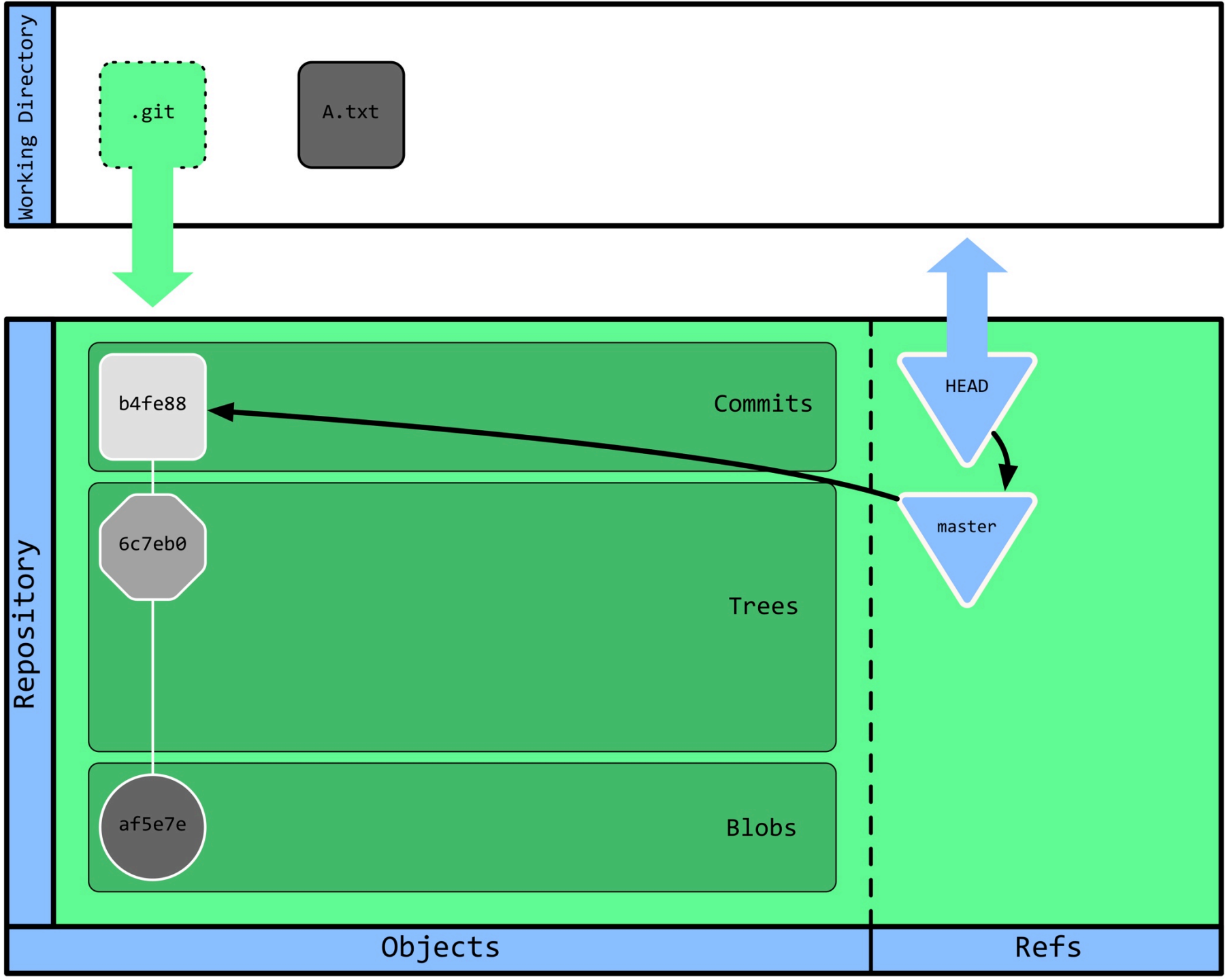

Working Directory
.git
A.txt
Repository
Commits
Trees
af5e7e
Blobs
HEAD
master
Objects
Refs

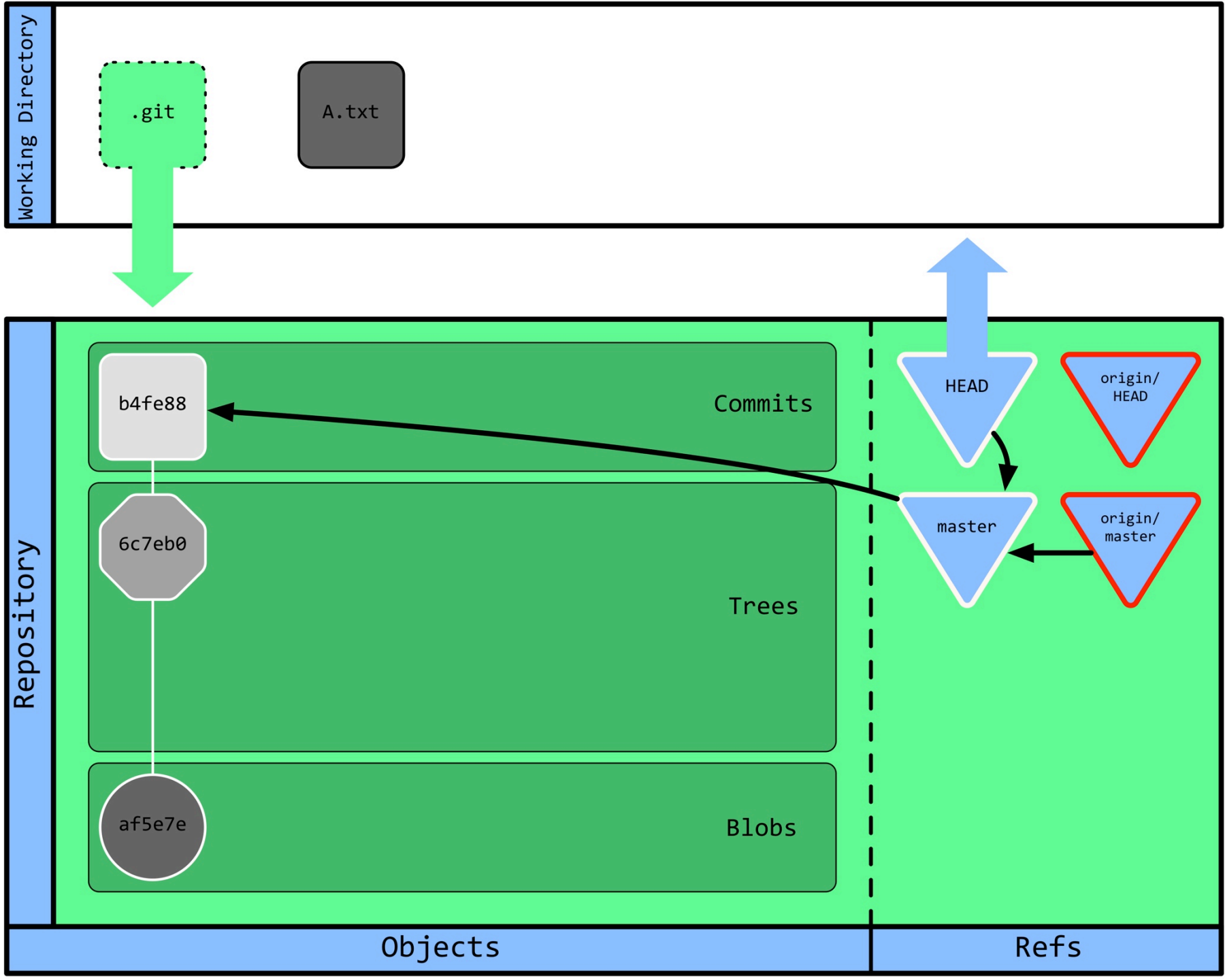

Working Directory
.git
A.txt
Repository
b4fe88
Commits
6c7eb0
Trees
af5e7e
Blobs
HEAD
master
Objects
Refs

Let's add a remote: `origin`

```
git remote add origin
```

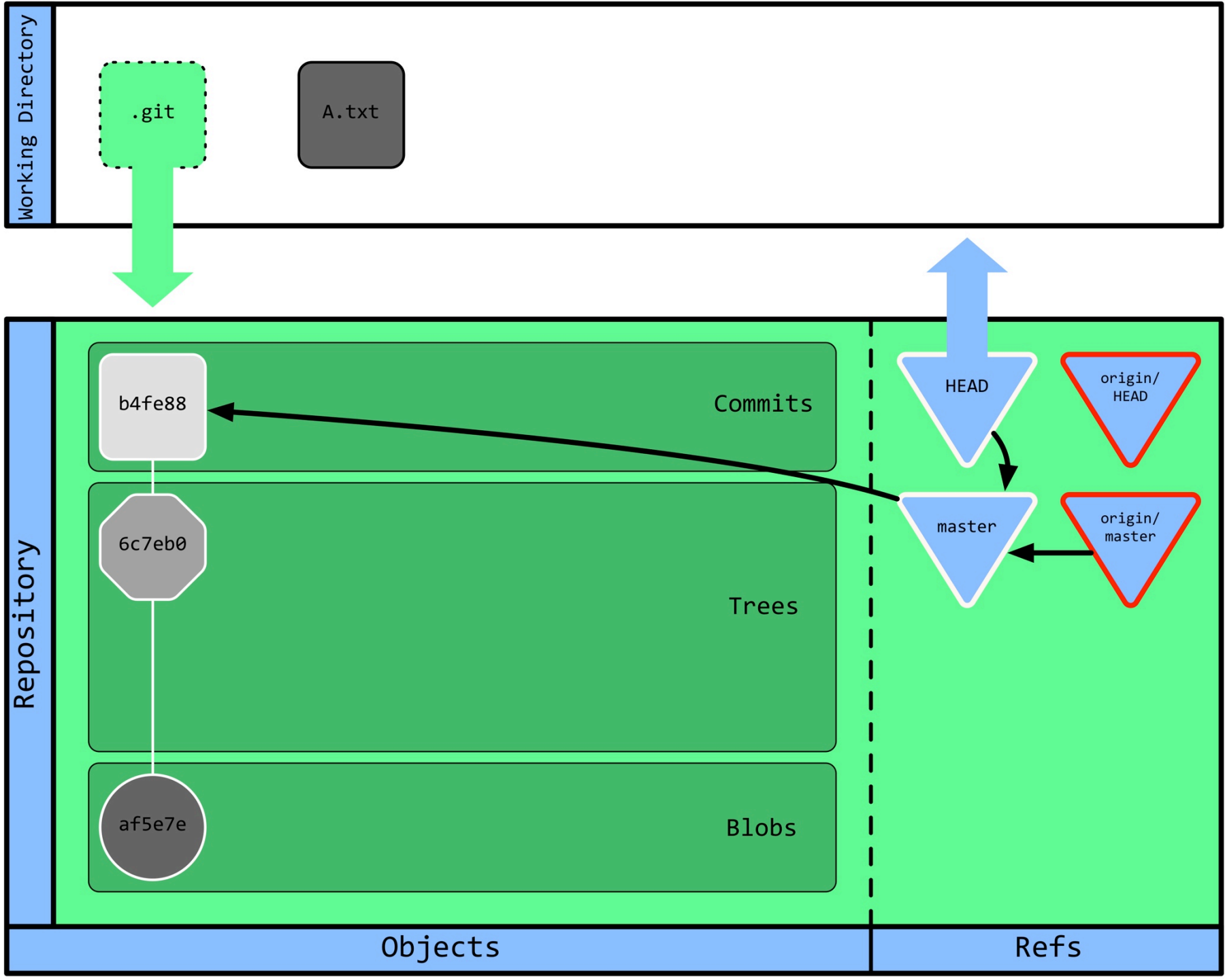
```
git push origin master
```

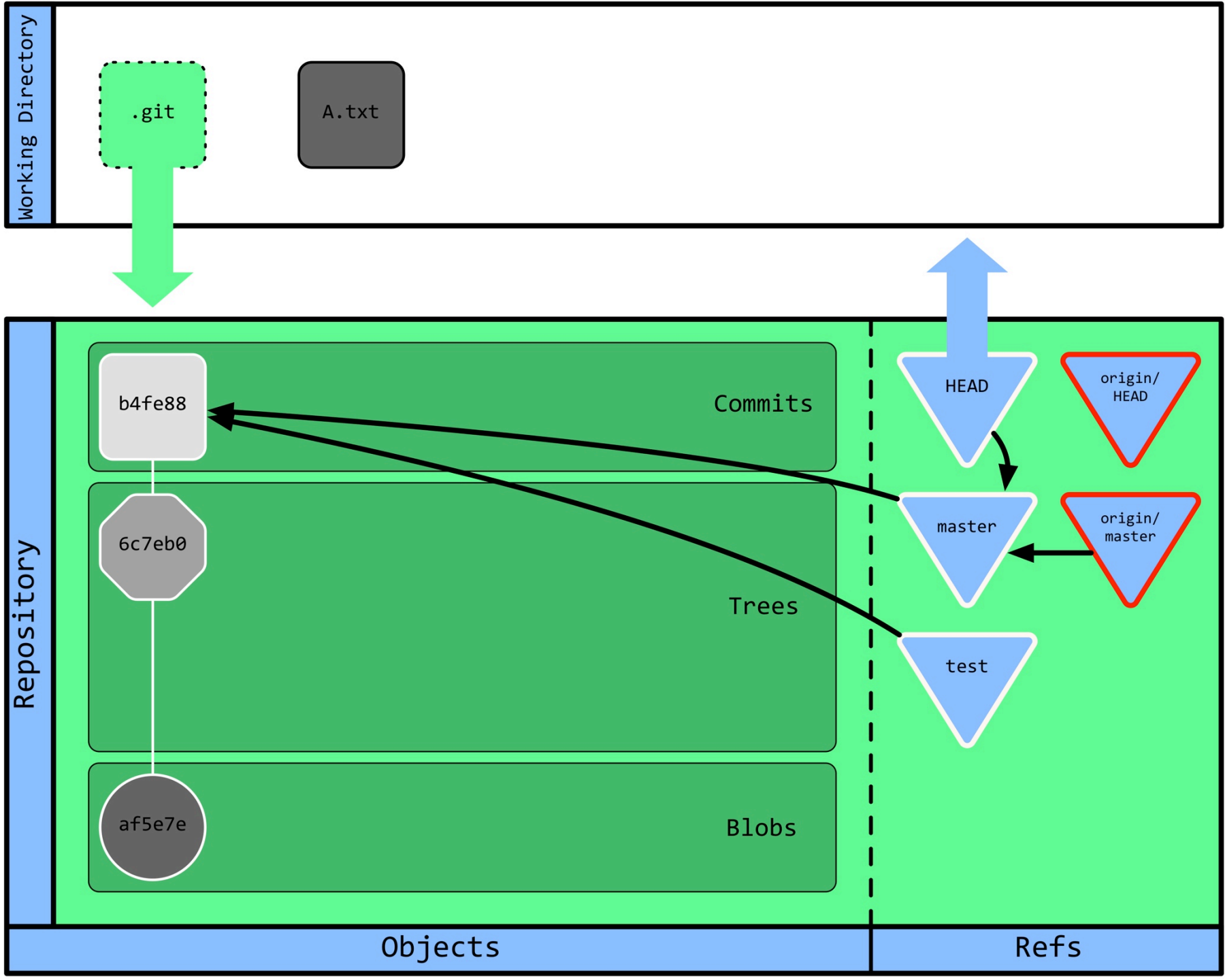

Working Directory
.git
A.txt
Repository
b4fe88
Commits
6c7eb0
Trees
af5e7e
Blobs
HEAD
master
Objects
Refs


Working Directory
.git
A.txt
Repository
b4fe88
Commits
6c7eb0
Trees
af5e7e
Blobs
HEAD
origin/
HEAD
master
origin/
master
Objects
Refs

Let's add a branch: `test`

```
git branch test
```

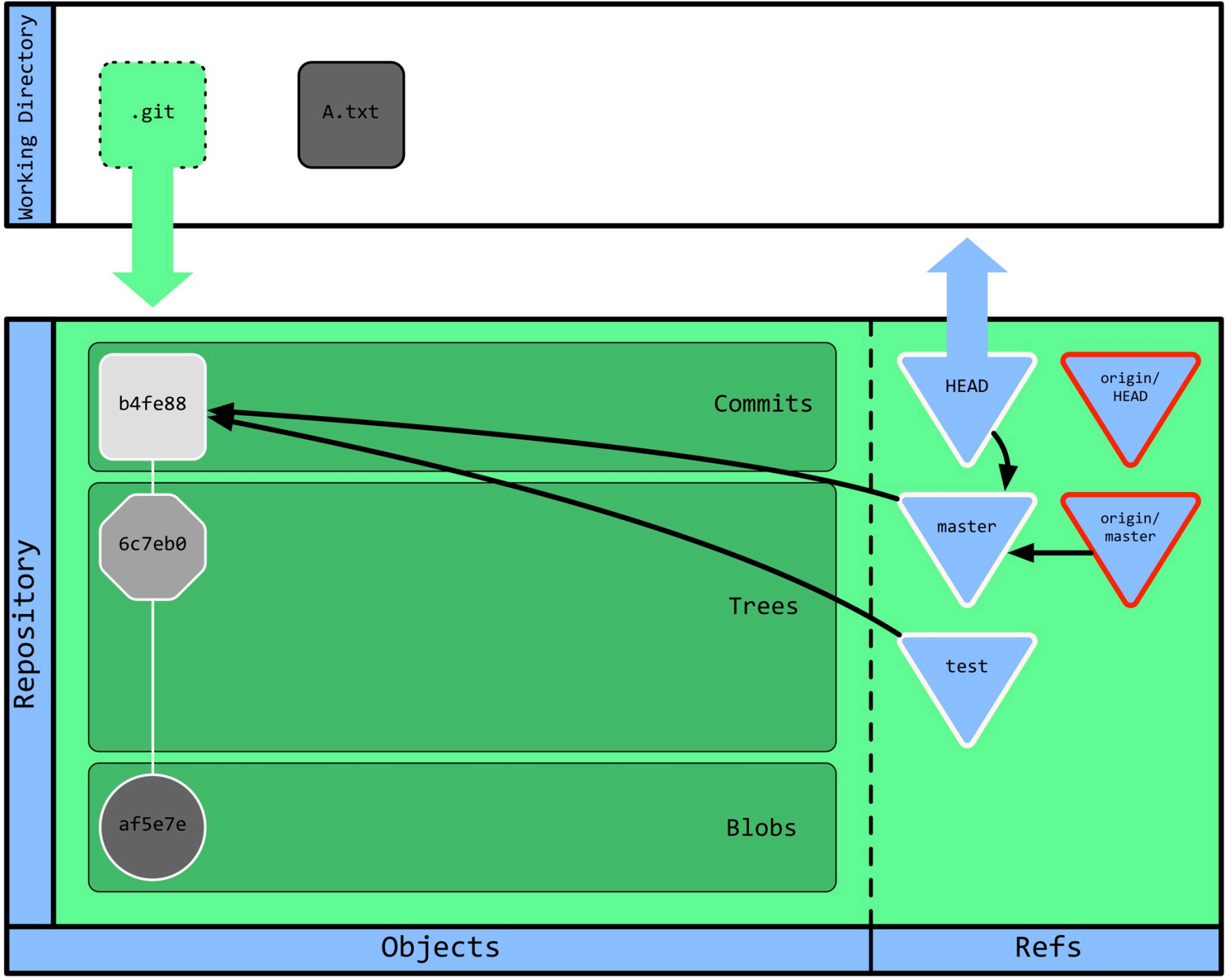

Working Directory
.git
A.txt
Repository
b4fe88
Commits
6c7eb0
Trees
af5e7e
Blobs
HEAD
origin/
HEAD
master
origin/
master
Objects
Refs

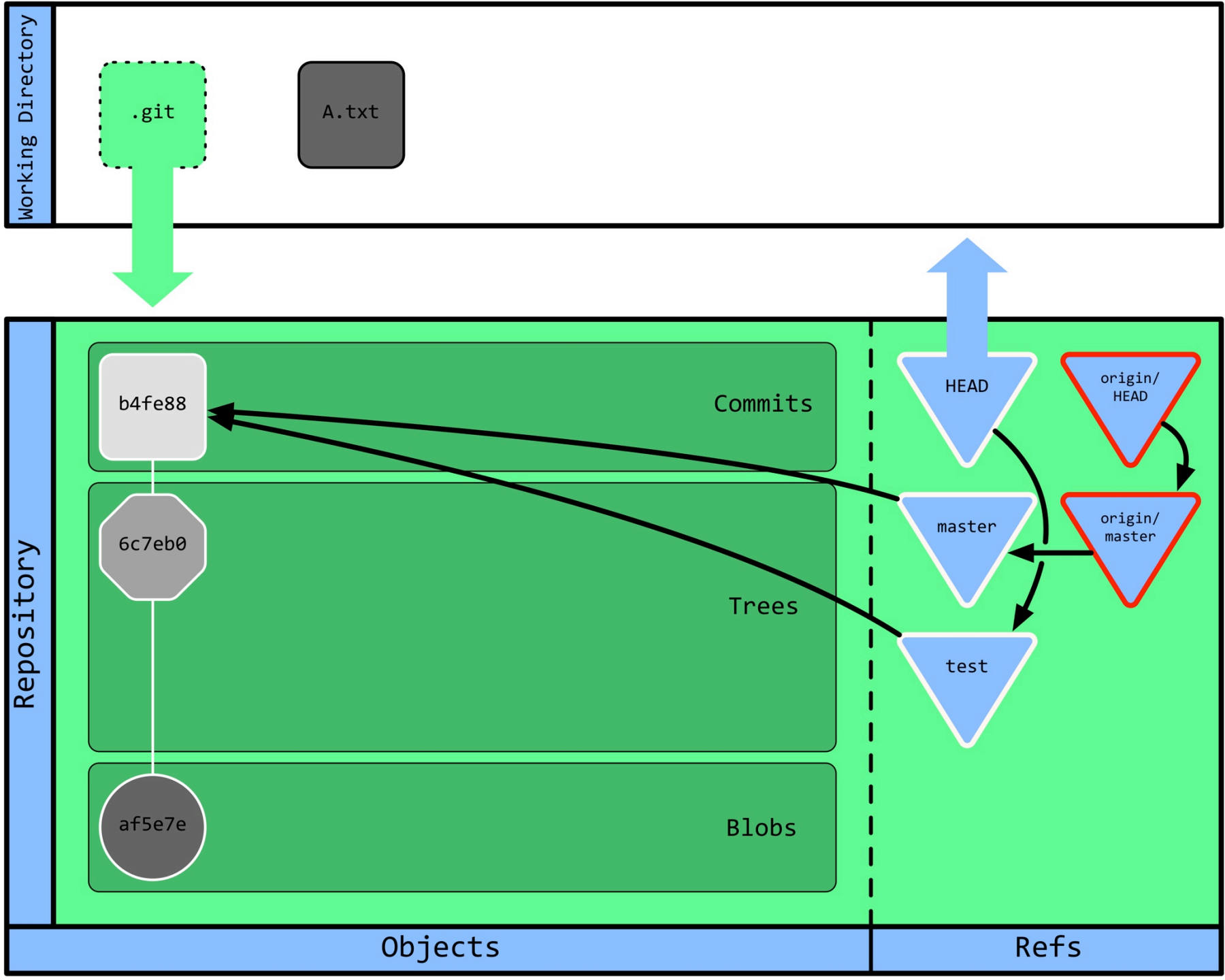

Working Directory
.git
A.txt
Repository
b4fe88
Commits
6c7eb0
Trees
af5e7e
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs

That's seriously what a branch is in Git

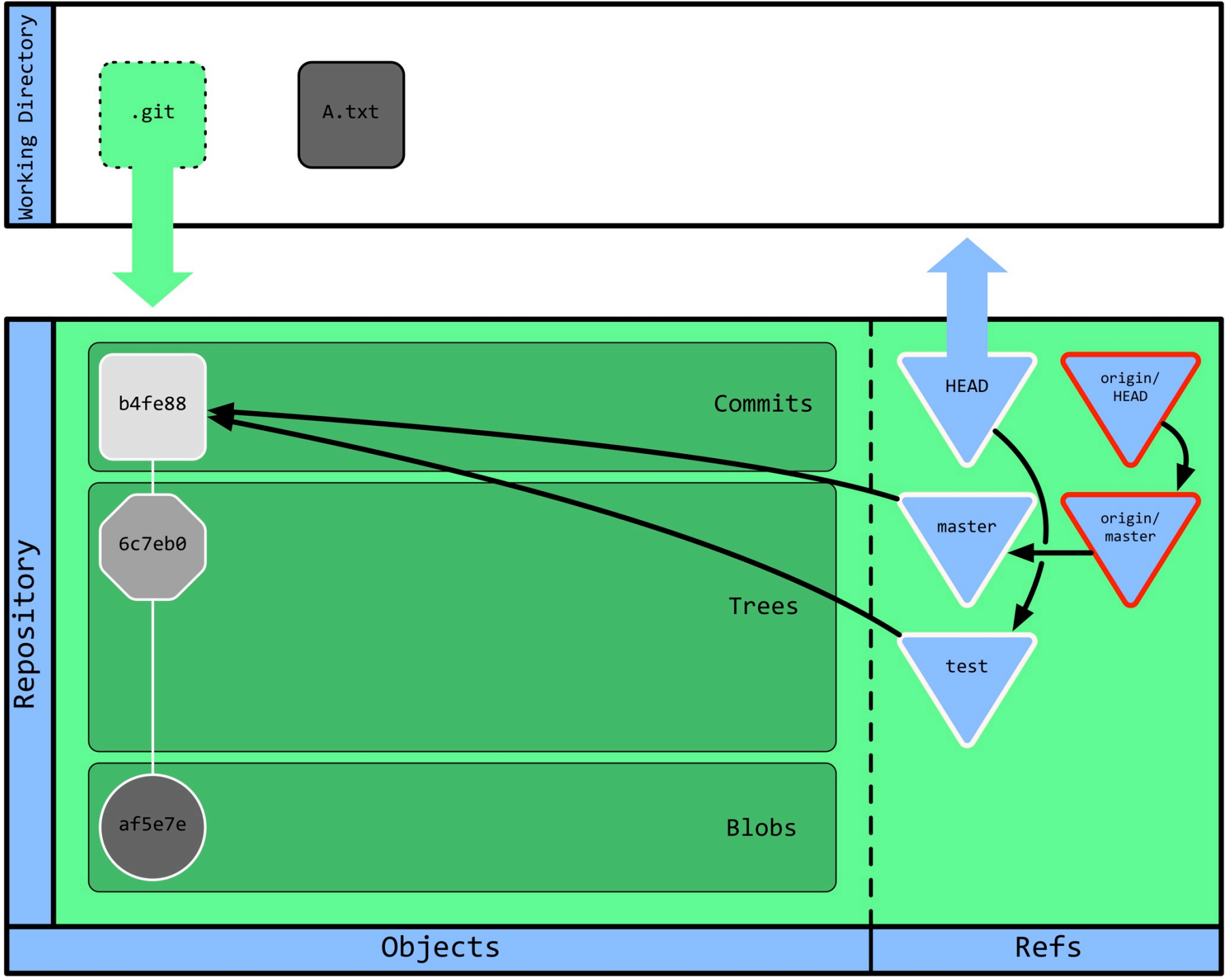
We can checkout the `test` branch

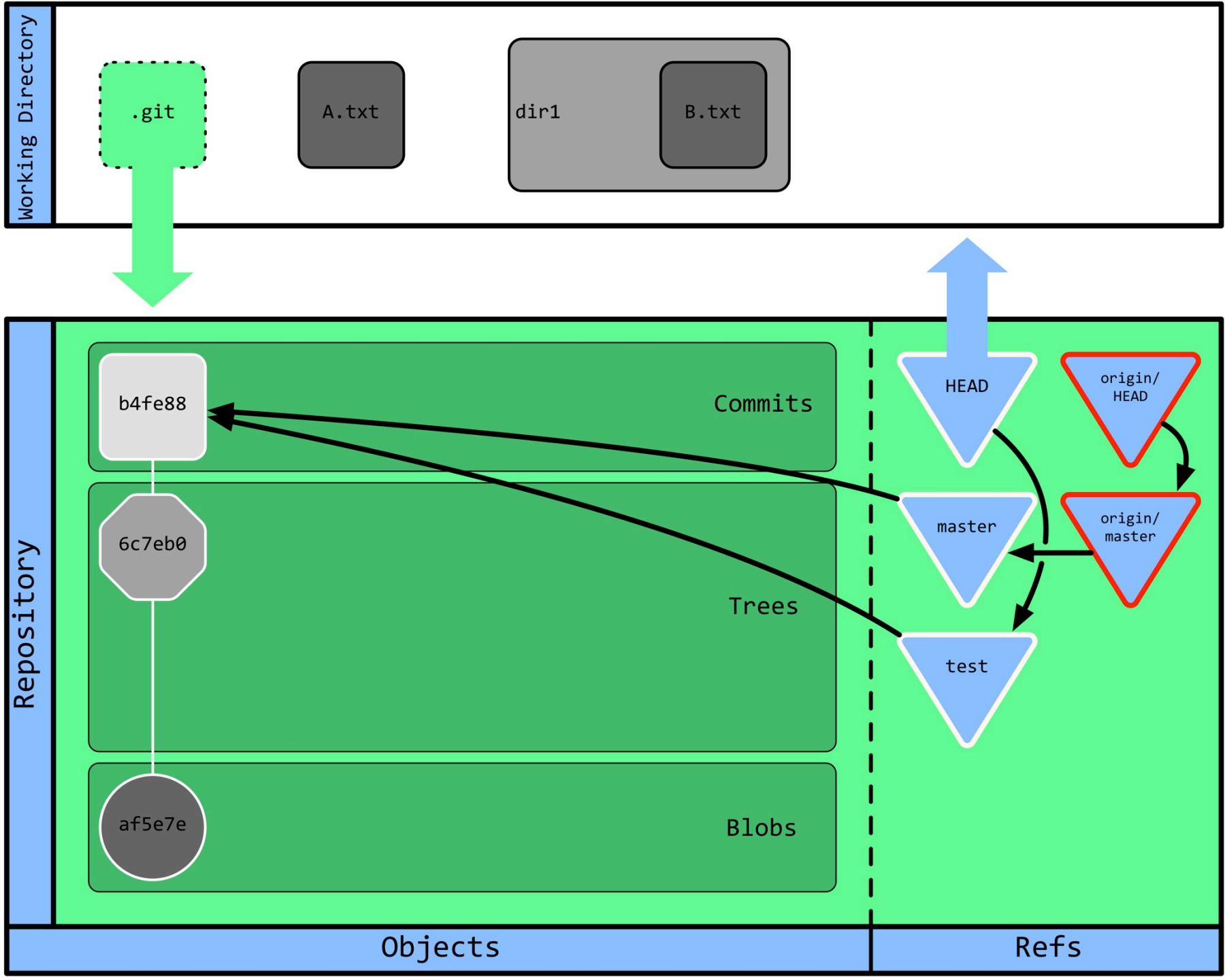
```
git checkout test
```


Working Directory
.git
A.txt
Repository
b4fe88
Commits
6c7eb0
Trees
af5e7e
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs

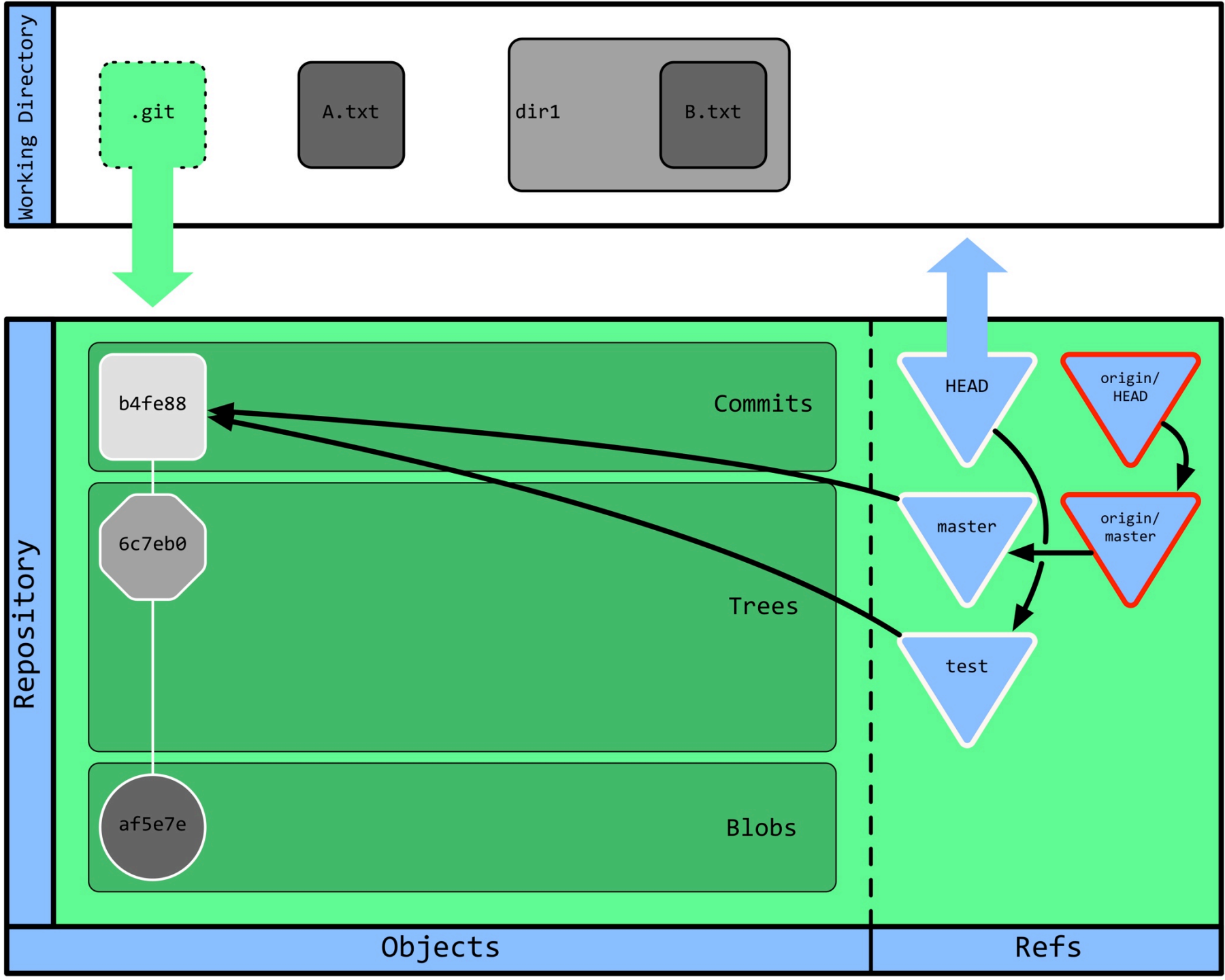

Working Directory
.git
A.txt
Repository
b4fe88
Commits
6c7eb0
Trees
af5e7e
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs

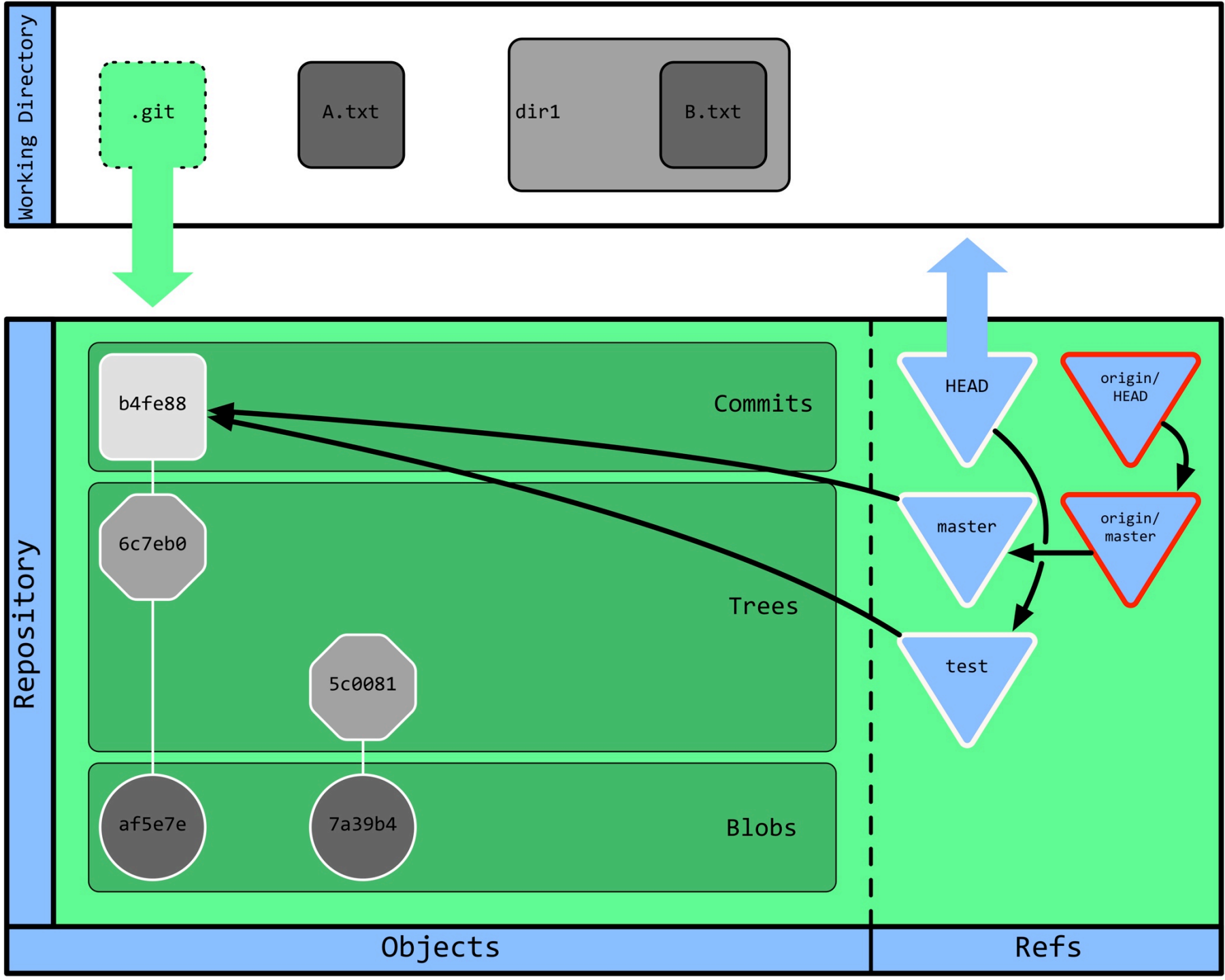
Let's create a directory: `dir1`
and a file: `B.txt`


Working Directory
.git
A.txt
Repository
b4fe88
Commits
6c7eb0
Trees
af5e7e
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs

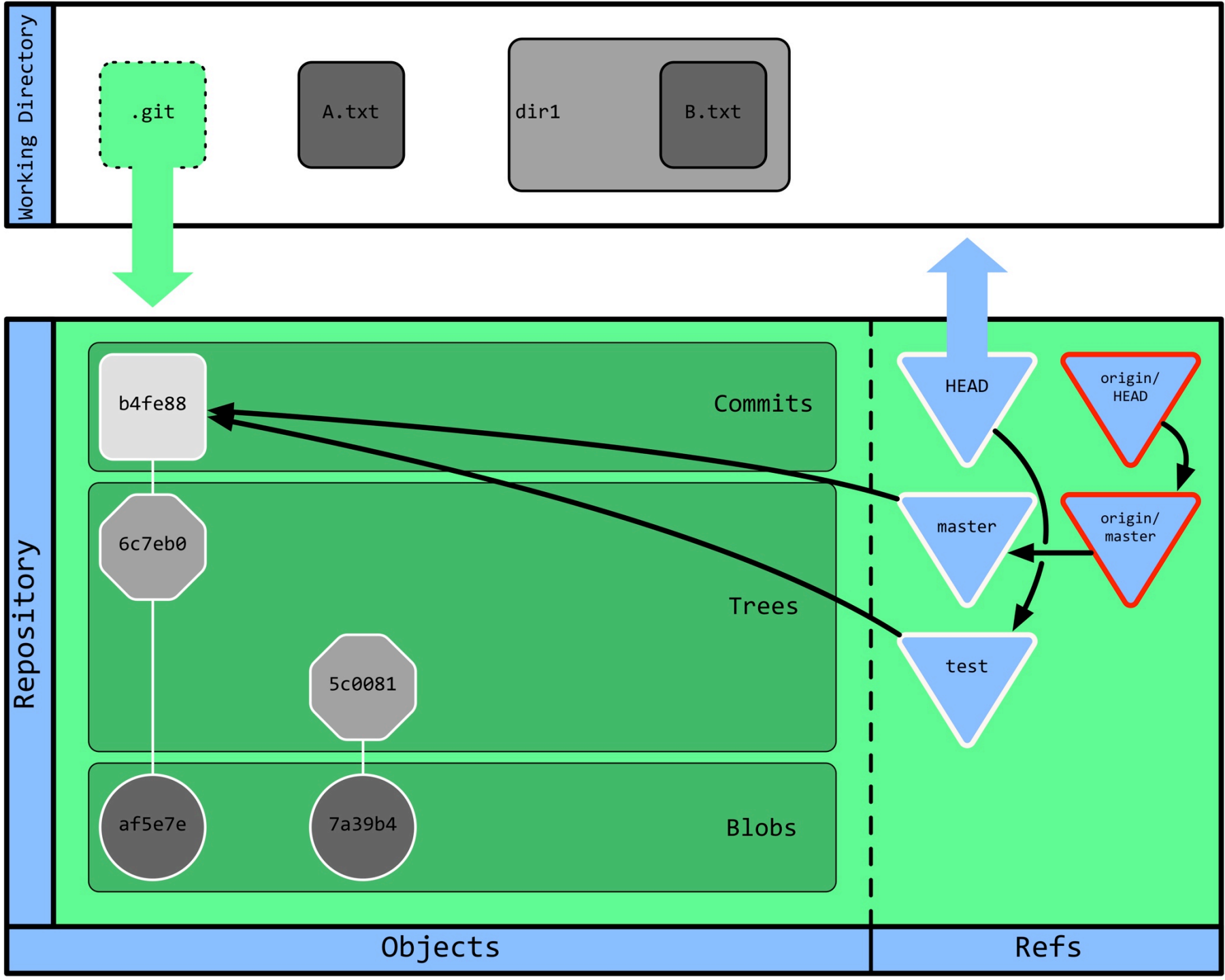

Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
Commits
6c7eb0
Trees
af5e7e
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs

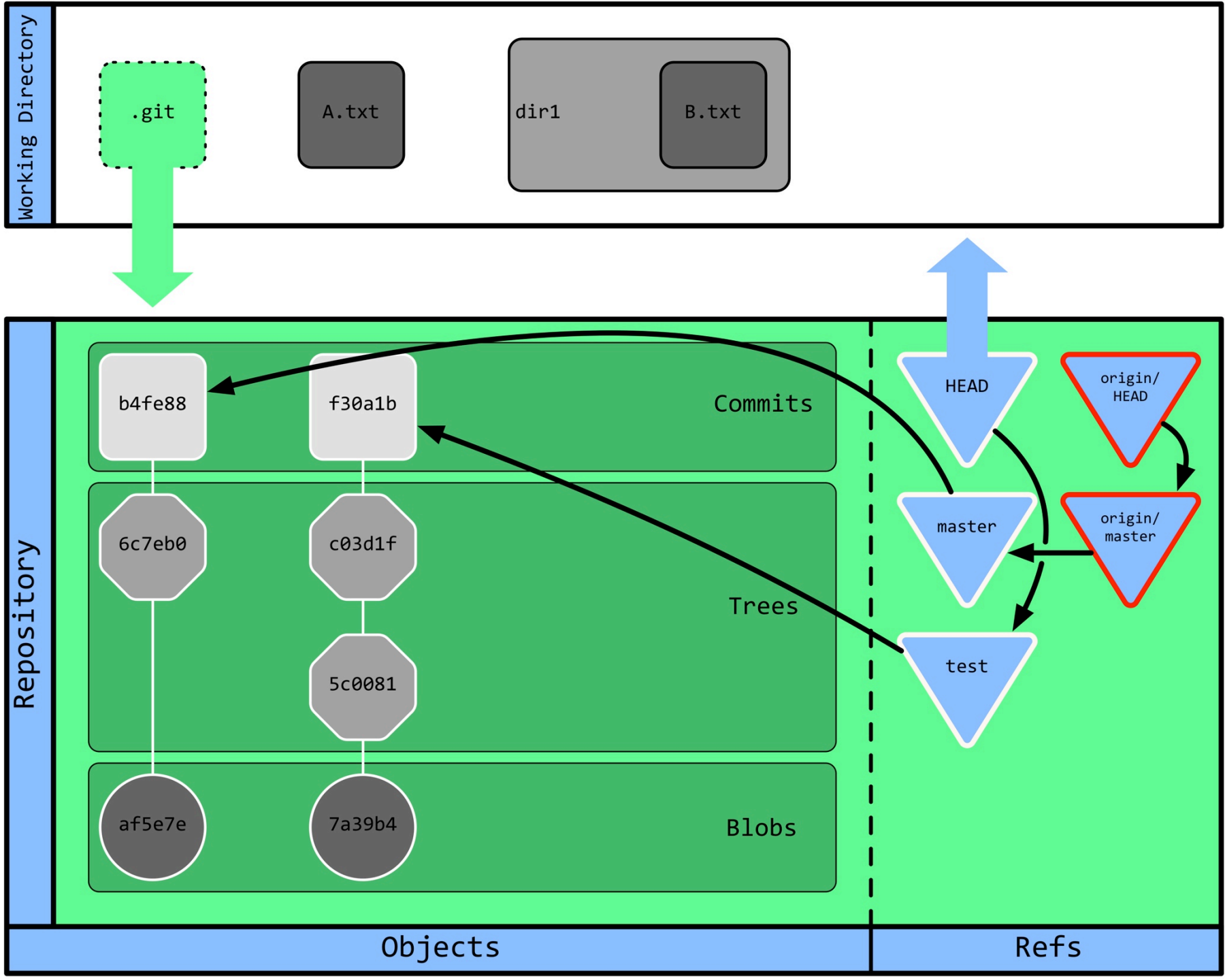
```
git add dir1/B.txt
```


Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
Commits
6c7eb0
Trees
af5e7e
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs

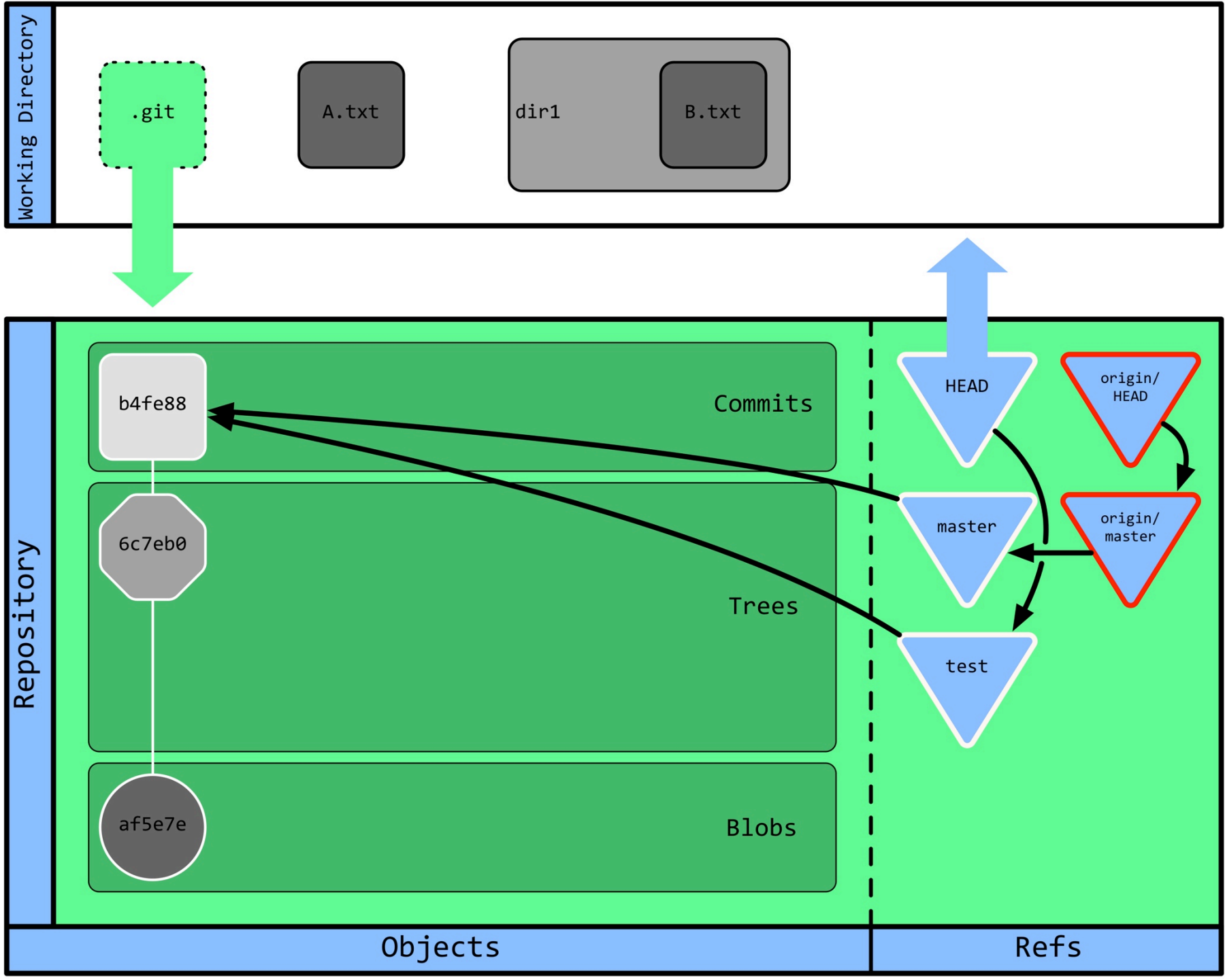

Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
Commits
6c7eb0
Trees
5c0081
af5e7e
7a39b4
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs

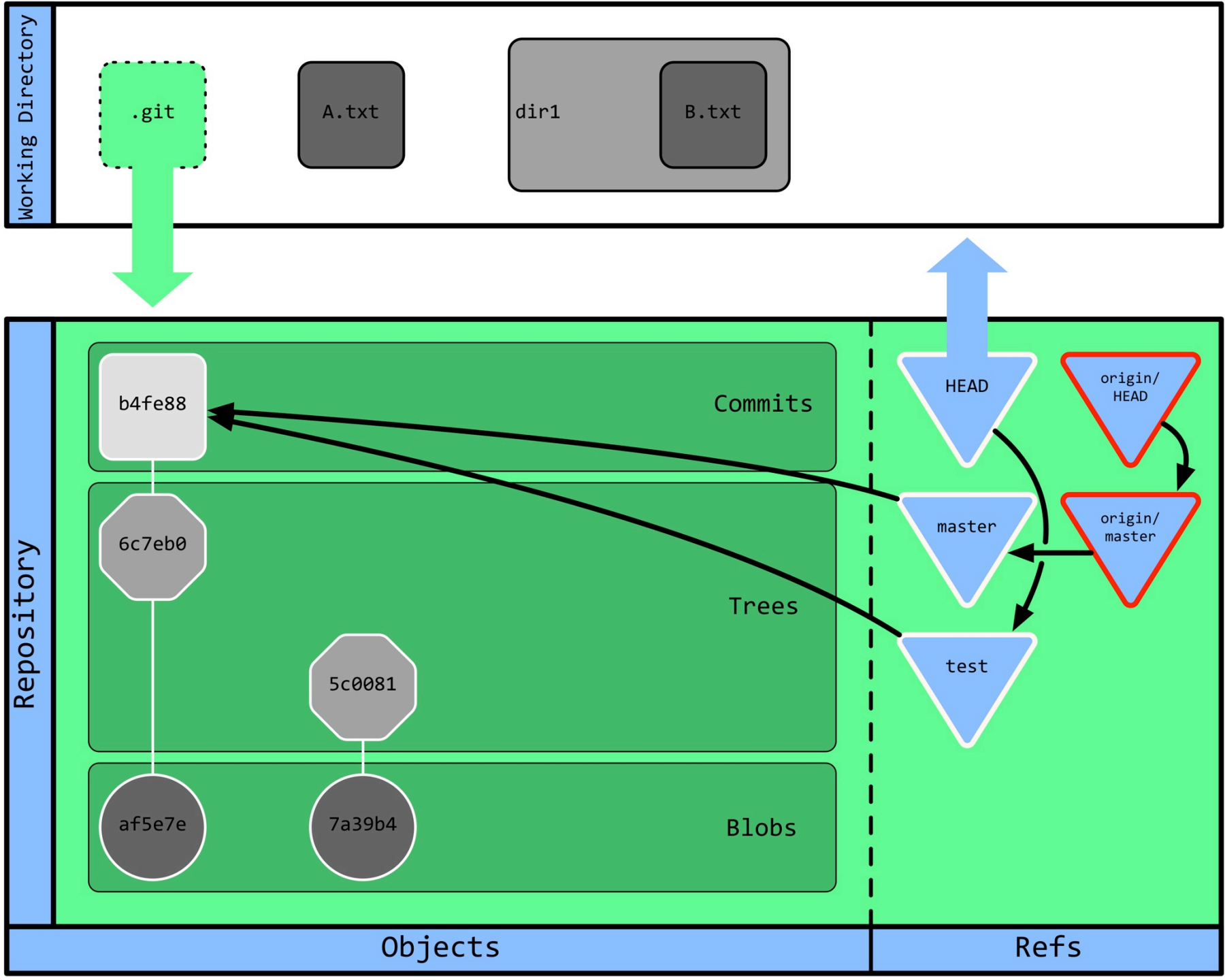
```
git commit -m “...”
```


Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
Commits
6c7eb0
Trees
5c0081
af5e7e
7a39b4
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs


Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
f30a1b
Commits
6c7eb0
c03d1f
Trees
5c0081
af5e7e
7a39b4
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs

Wait…What's happening between the `add` and `commit`?


Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
Commits
6c7eb0
Trees
af5e7e
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs


Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
Commits
6c7eb0
Trees
5c0081
af5e7e
7a39b4
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs

The Staging Area/Index

- As soon as a blob or tree is created by calling **add**, it is added to the **index**
- It’s a literal database...

```
$ git add dir1/B.txt
$ git ls-files --stage
100644 e69de29bb2d1d6434b8b29ae775ad8c2e48c5391  0   A.txt
100644 4c5477a837a981cb182f4bad7c4425ec379a9cac  0   dir1/B.txt
$ git commit -m "Add file B”
```

The Staging Area/Index

- Calling **git commit** takes the blobs and trees in the index, adds a top-level tree object and then wraps them all up in a commit object

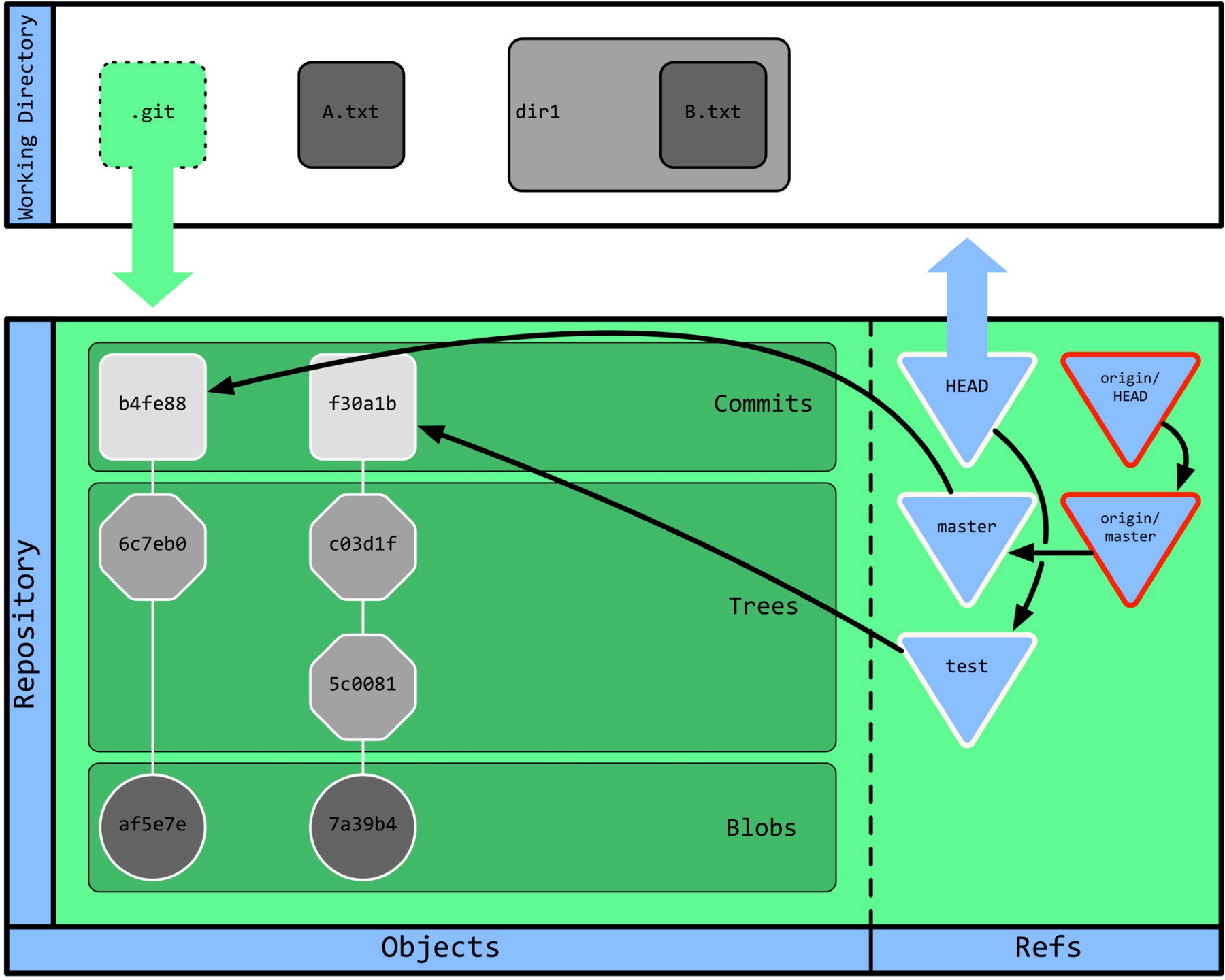
Branches, again

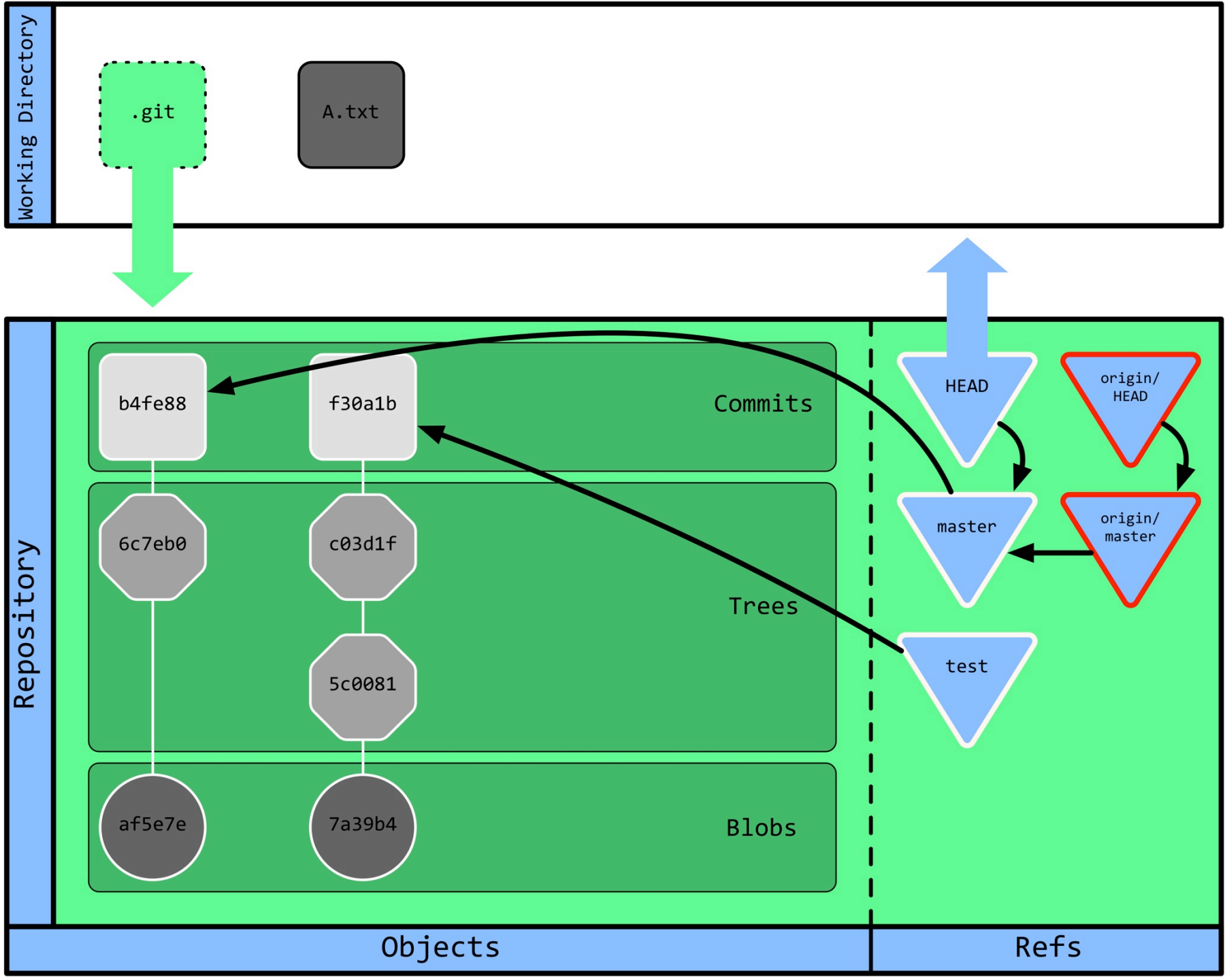
- Also notice that the **test** branch pointer automatically moved after the commit...

```
$ git ls
* 3e6d961 - (HEAD -> test) Add file B (6 minutes ago) <Joe Gibson>
* 901afad - (master) Add file A (8 minutes ago) <Joe Gibson>

$ git cat-file -p 3e6d961
tree 43222cd2e96acac8c83f5162226721f3351c2208
parent 901afadf676ff8790af9b520708be50d0c10f645
author Joe Gibson <joseph.gibson@nasa.gov> 1469156527 -0400
committer Joe Gibson <joseph.gibson@nasa.gov> 1469156527 -0400

Add file B
```

Let's try merging `test` into `master`

```
git checkout master
```

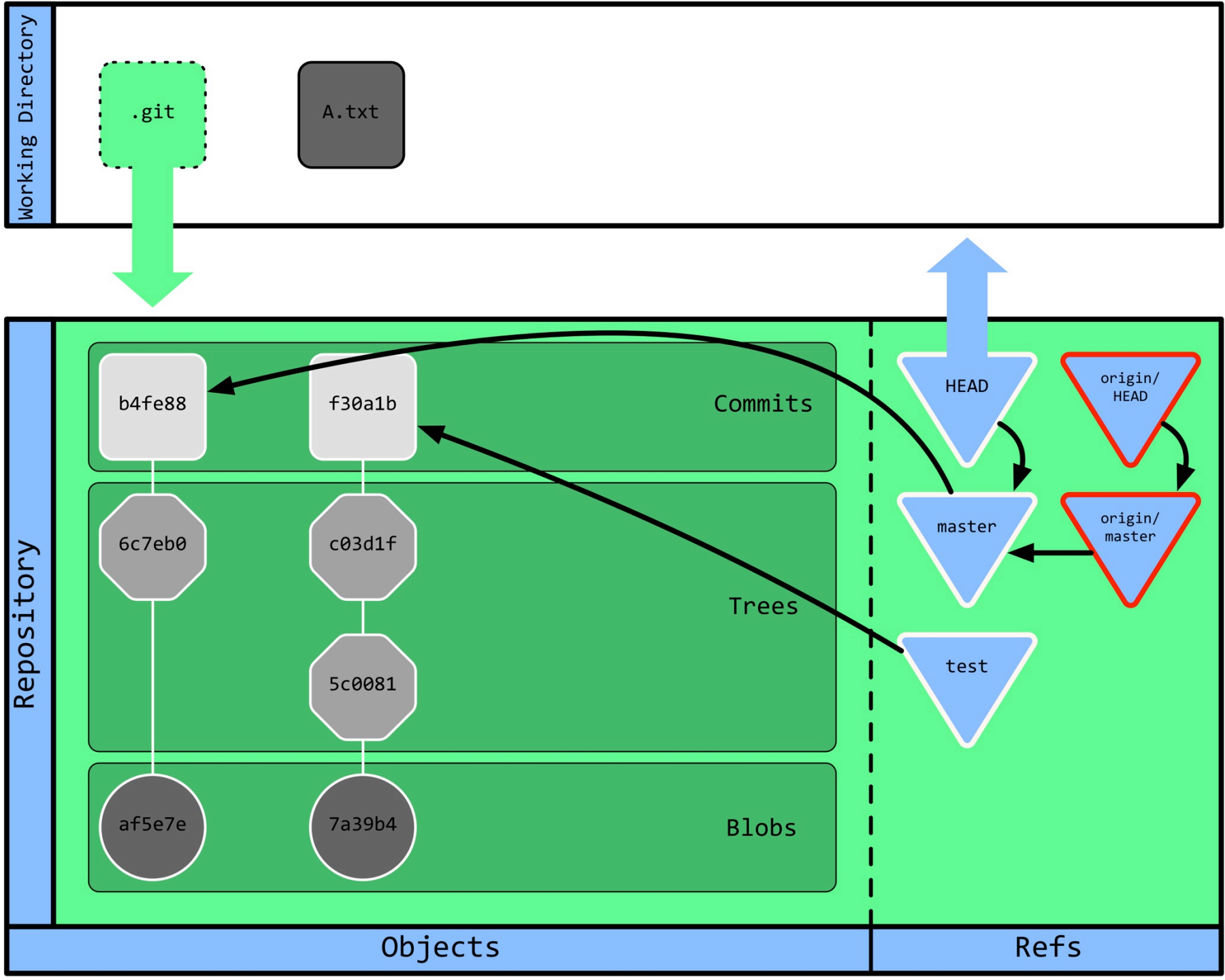

Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
f30a1b
Commits
6c7eb0
c03d1f
Trees
5c0081
af5e7e
7a39b4
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs

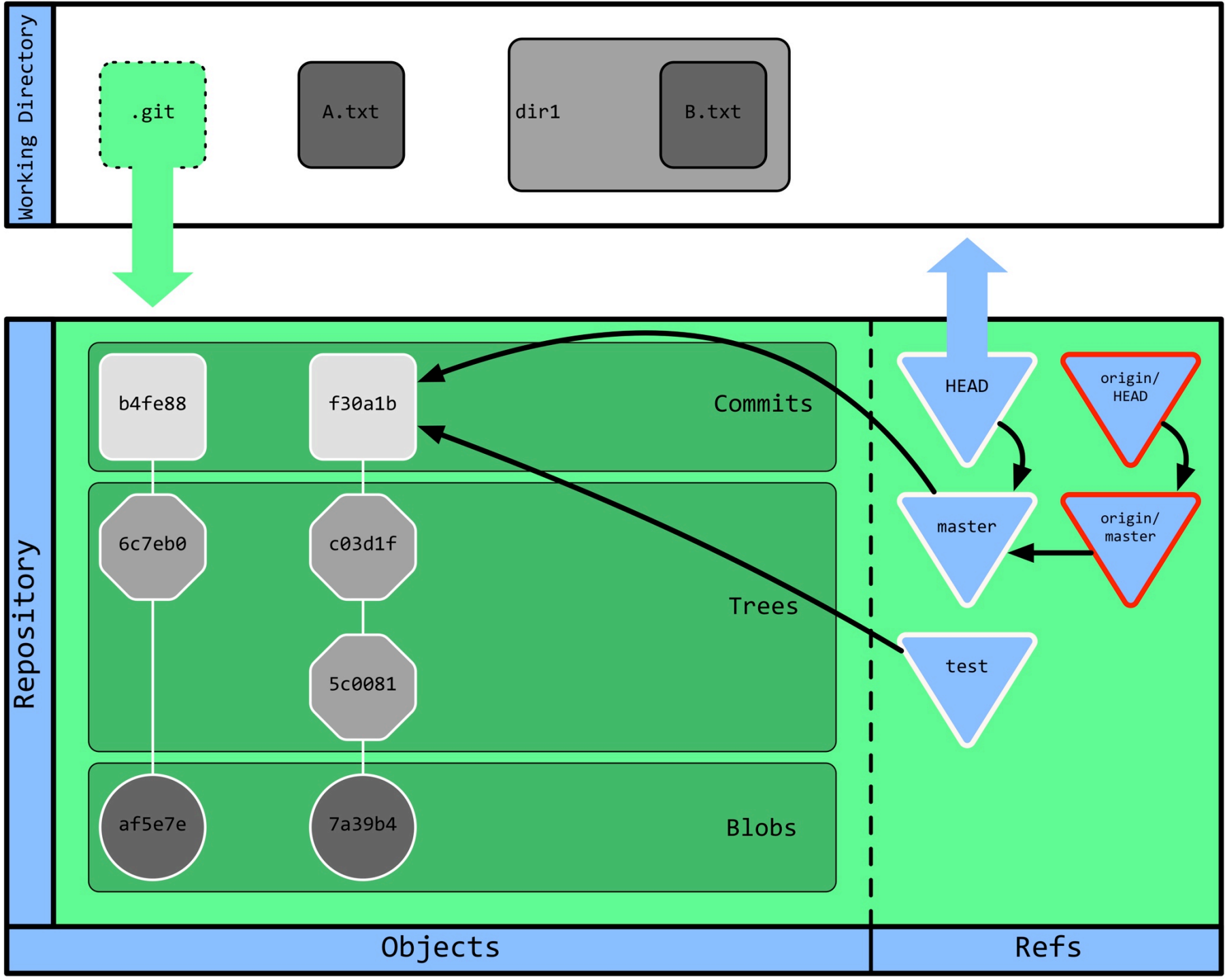

Working Directory
.git
A.txt
Repository
b4fe88
f30a1b
Commits
6c7eb0
c03d1f
Trees
5c0081
af5e7e
7a39b4
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs

Notice how `A.txt` is the only file now…

Hint: HEAD points to master, and whatever HEAD points to ends up in your working directory.

```
git merge test
```


Working Directory
.git
A.txt
Repository
b4fe88
f30a1b
Commits
6c7eb0
c03d1f
Trees
5c0081
af5e7e
7a39b4
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs


Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
f30a1b
Commits
6c7eb0
c03d1f
Trees
5c0081
af5e7e
7a39b4
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs

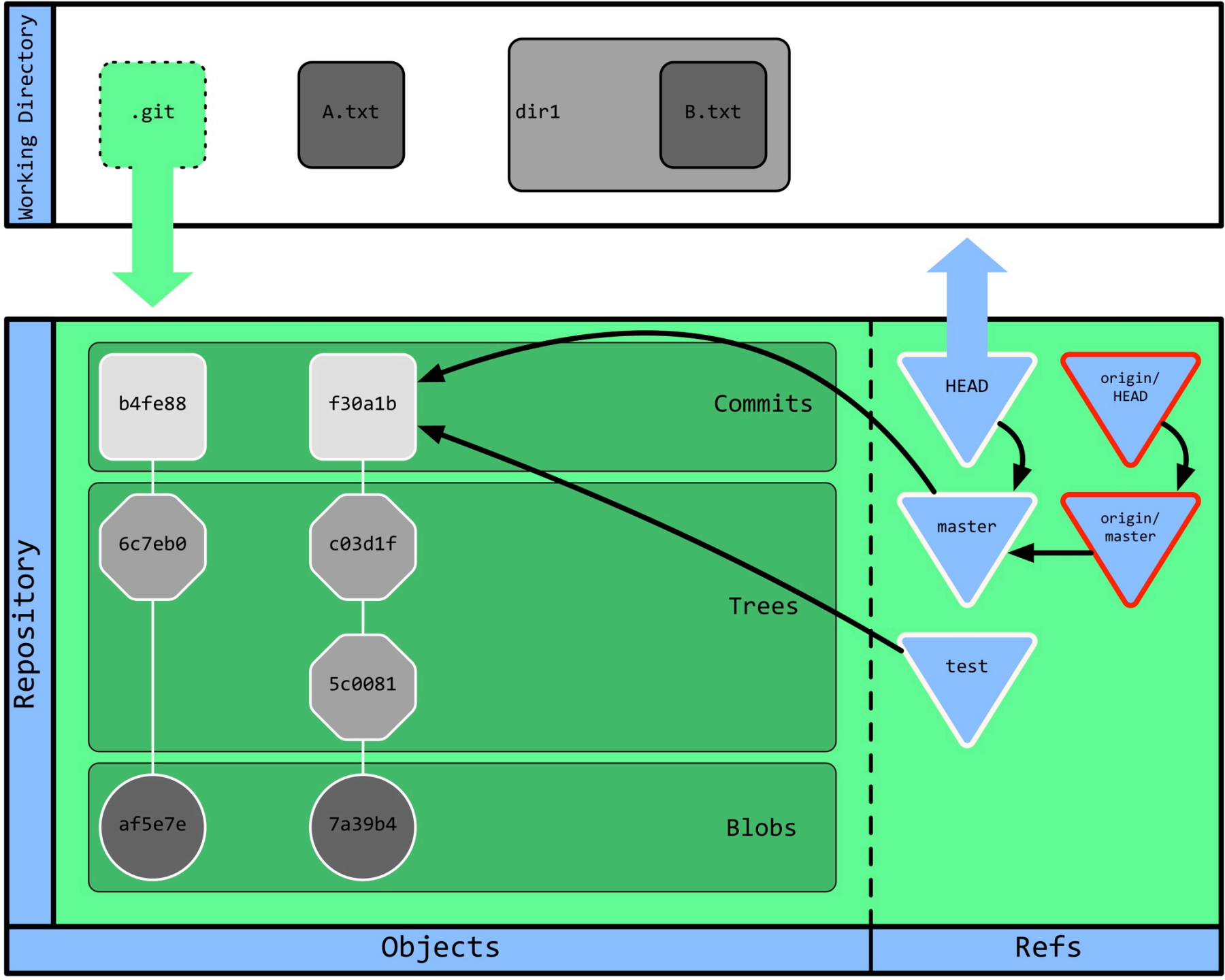
Easy, right?

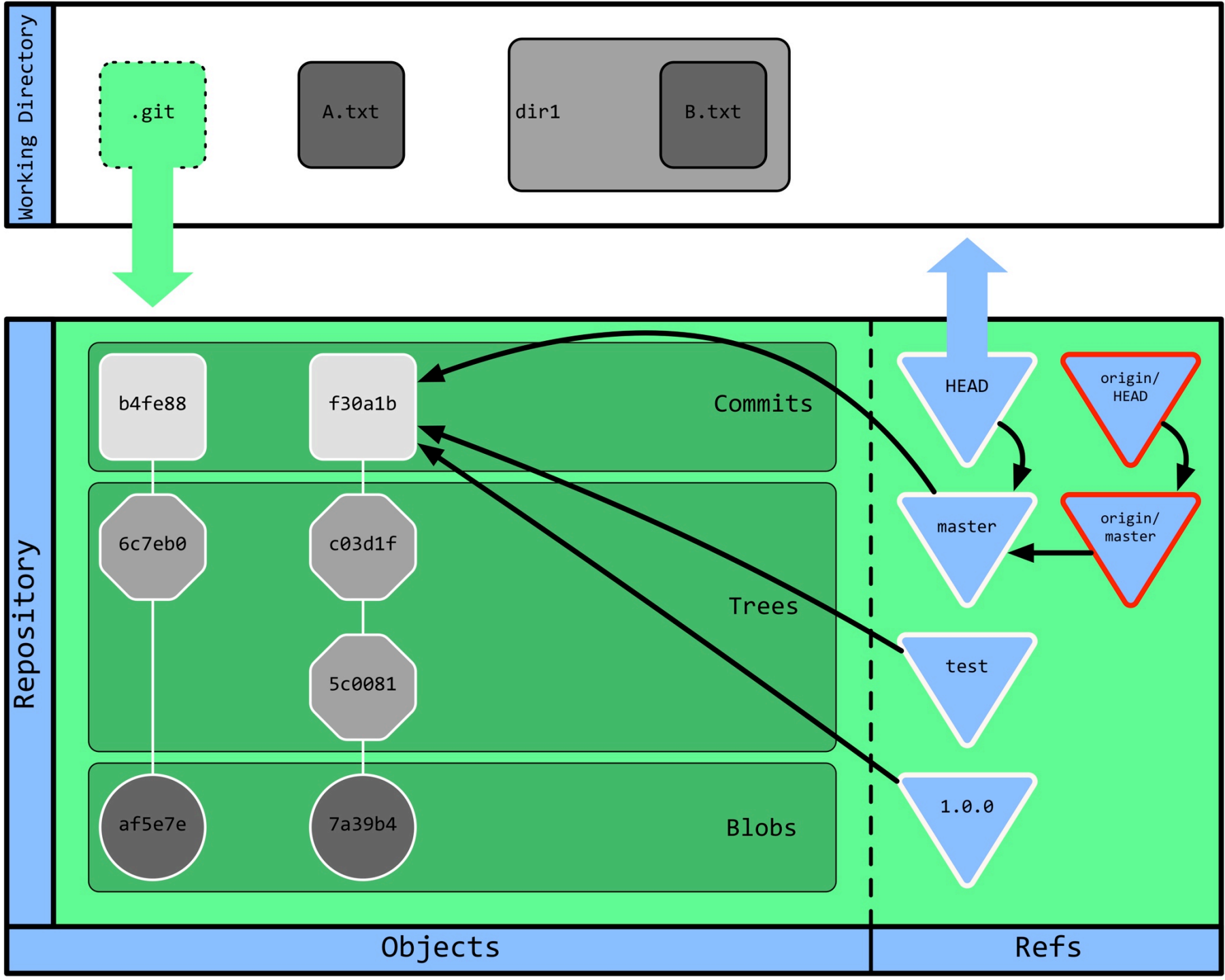
That was called a “fast-forward” merge.

Other merges can require an additional commit, but we’ll stick with the simple case.

We're happy with the state of the repo, so let's tag it as `1.0.0`

```
git tag 1.0.0
```


Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
f30a1b
Commits
6c7eb0
c03d1f
Trees
5c0081
af5e7e
7a39b4
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
Objects
Refs

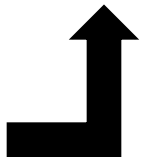

Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
f30a1b
Commits
6c7eb0
c03d1f
Trees
5c0081
af5e7e
7a39b4
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
1.0.0
Objects
Refs

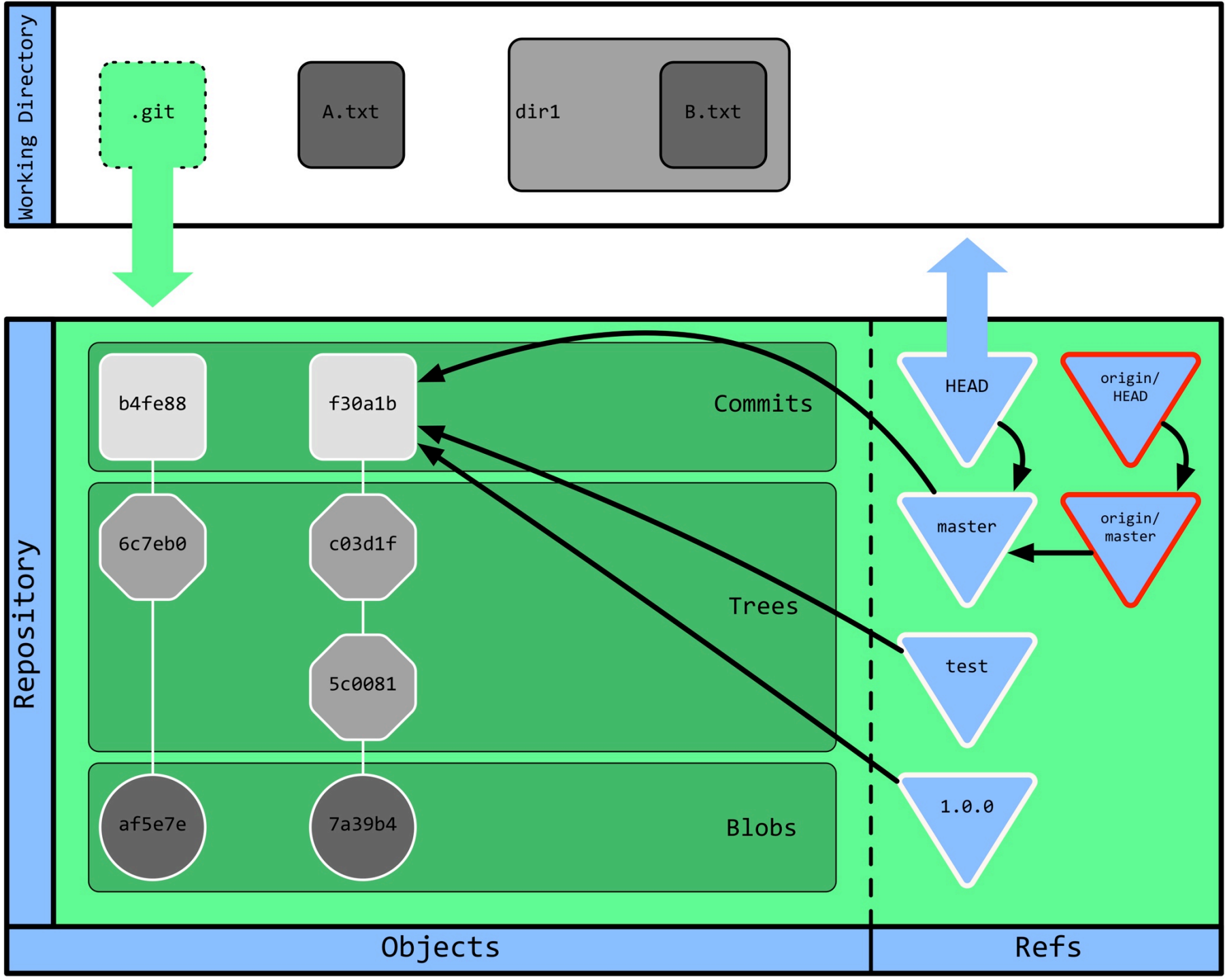
Yes, it's really that simple.

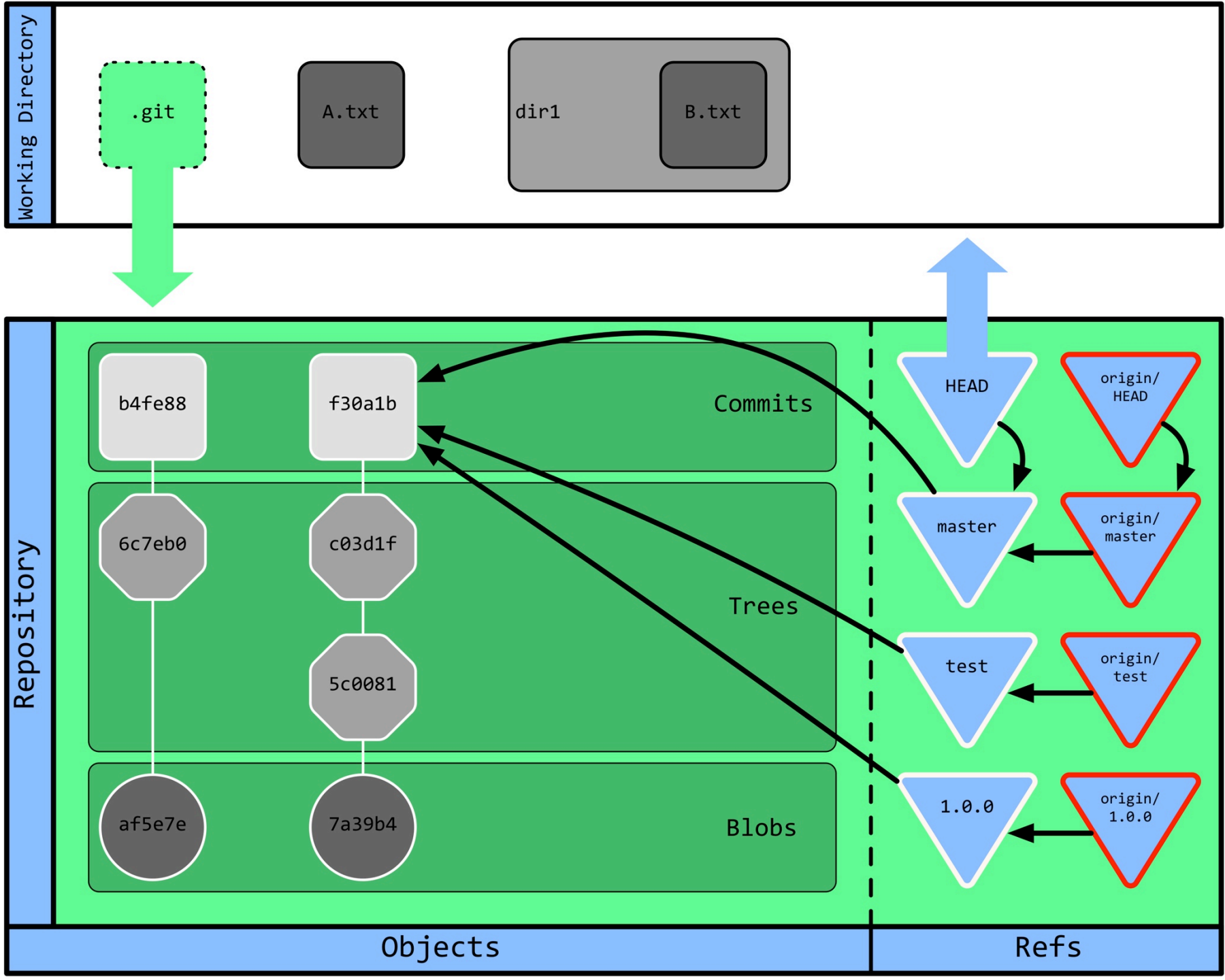
We'll be a team player and push our changes to the remote

```
git push origin master 1.0.0
```

Explicitly specify tag,
or use the **--tags** option

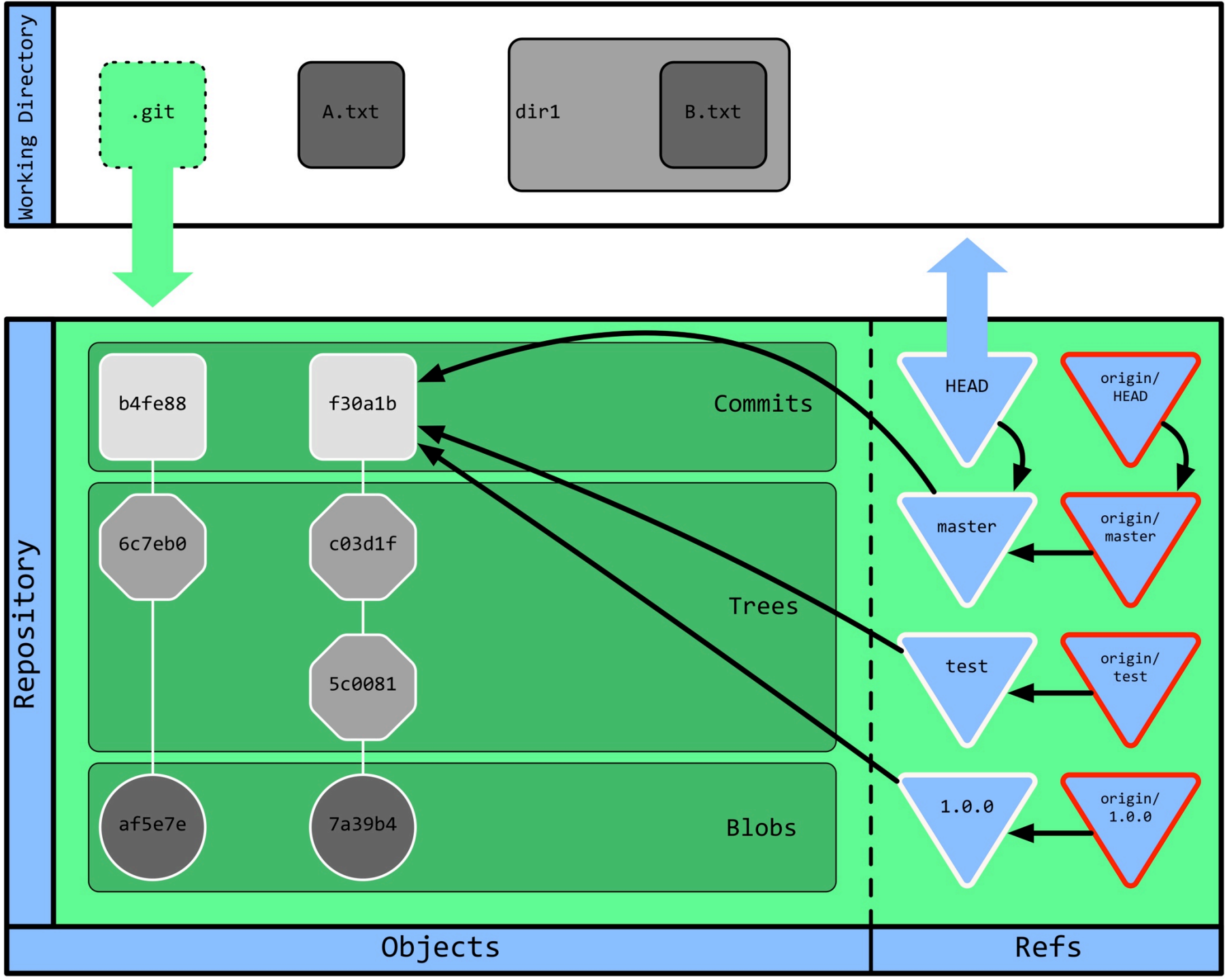


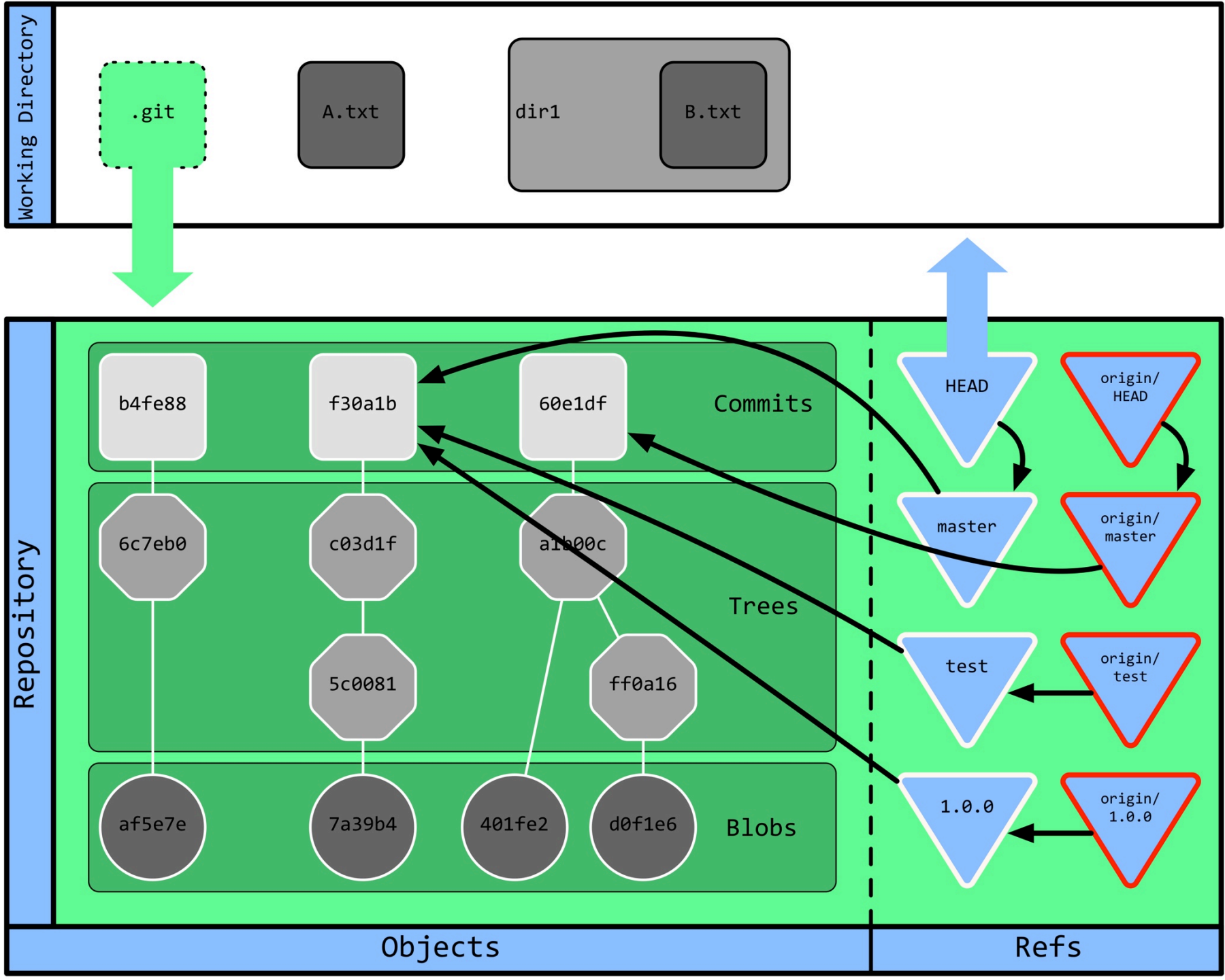

Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
f30a1b
Commits
6c7eb0
c03d1f
Trees
5c0081
af5e7e
7a39b4
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
1.0.0
Objects
Refs


Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
f30a1b
Commits
6c7eb0
c03d1f
Trees
5c0081
af5e7e
7a39b4
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
origin/
test
1.0.0
origin/
1.0.0
Objects
Refs

After some time, we can `fetch` from the remote to grab any changes others have made.

`git fetch`

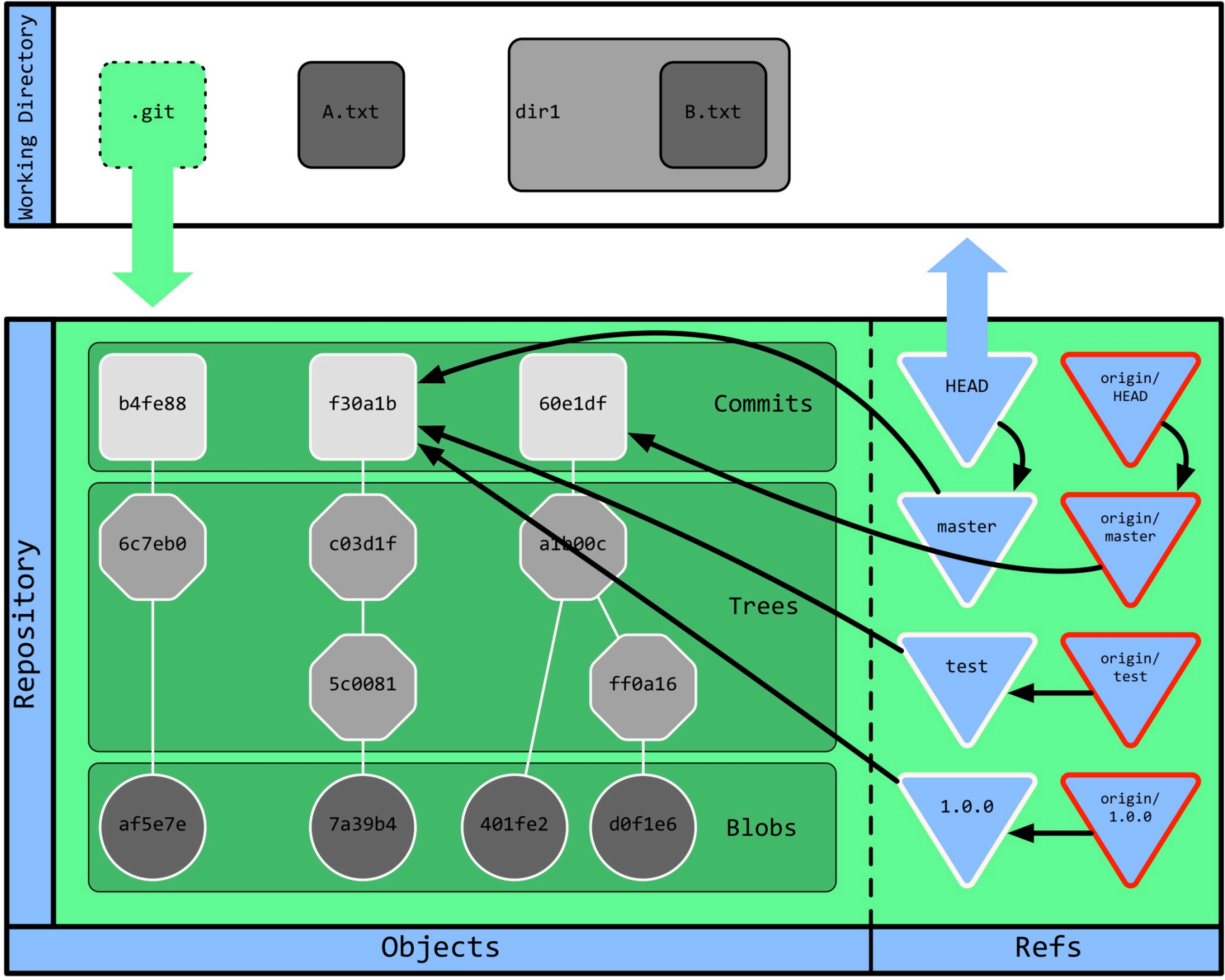

Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
f30a1b
Commits
6c7eb0
c03d1f
Trees
5c0081
af5e7e
7a39b4
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
origin/
test
1.0.0
origin/
1.0.0
Objects
Refs

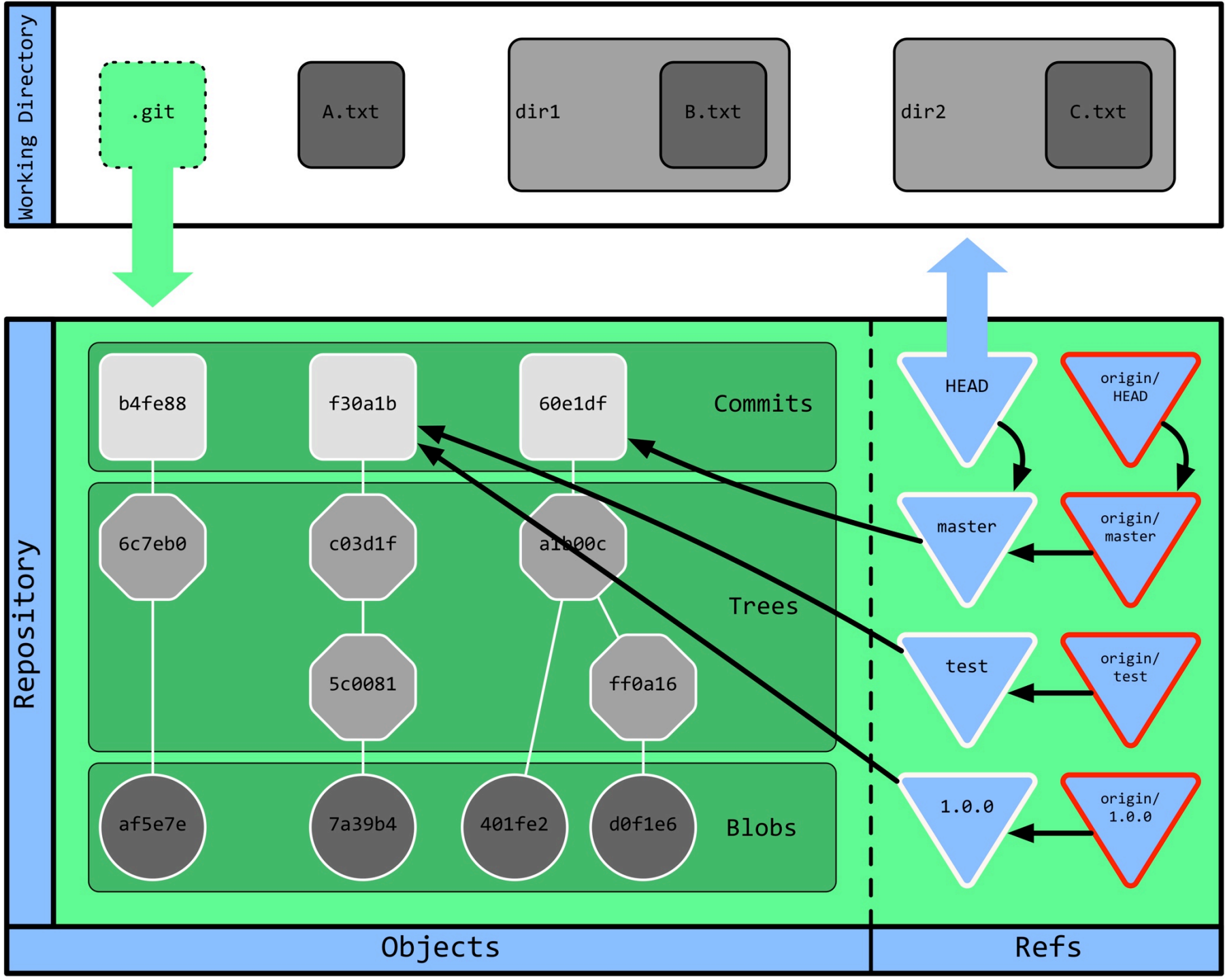

Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
f30a1b
60e1df
Commits
6c7eb0
c03d1f
a1b00c
Trees
5c0081
ff0a16
af5e7e
7a39b4
401fe2
d0f1e6
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
origin/
test
1.0.0
origin/
1.0.0
Objects
Refs

It looks like someone made a commit, adding `dir2` and `C.txt`, and modified the contents of `A.txt`

To actually update our files, we need to merge `origin/master` into our local `master`

```
git merge origin/master
```


Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
f30a1b
60e1df
Commits
6c7eb0
c03d1f
a1b00c
Trees
5c0081
ff0a16
af5e7e
7a39b4
401fe2
d0f1e6
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
origin/
test
1.0.0
origin/
1.0.0
Objects
Refs

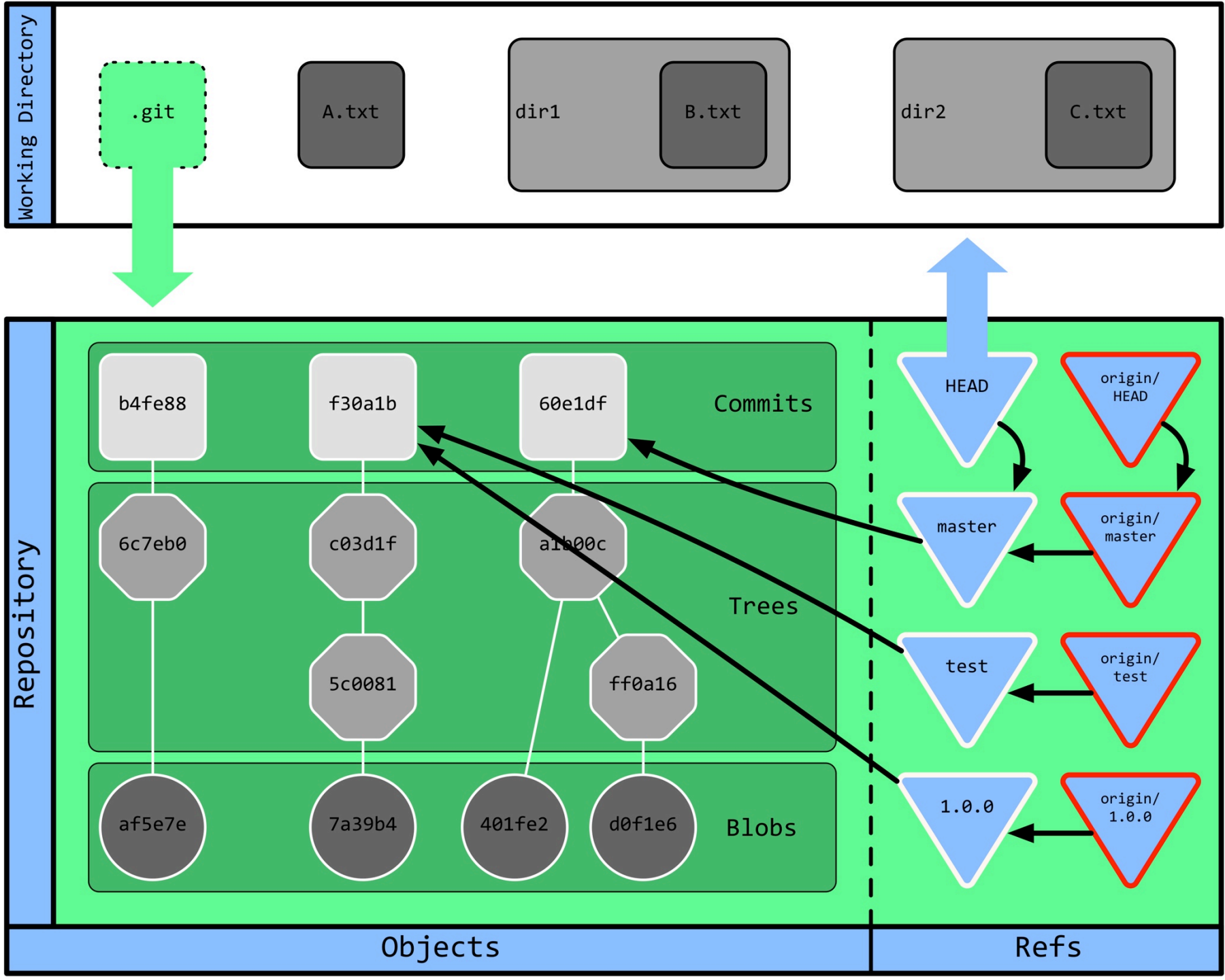

Working Directory
.git
A.txt
dir1
B.txt
dir2
C.txt
Repository
b4fe88
f30a1b
60e1df
Commits
6c7eb0
c03d1f
a1b00c
Trees
5c0081
ff0a16
af5e7e
7a39b4
401fe2
d0f1e6
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
origin/
test
1.0.0
origin/
1.0.0
Objects
Refs

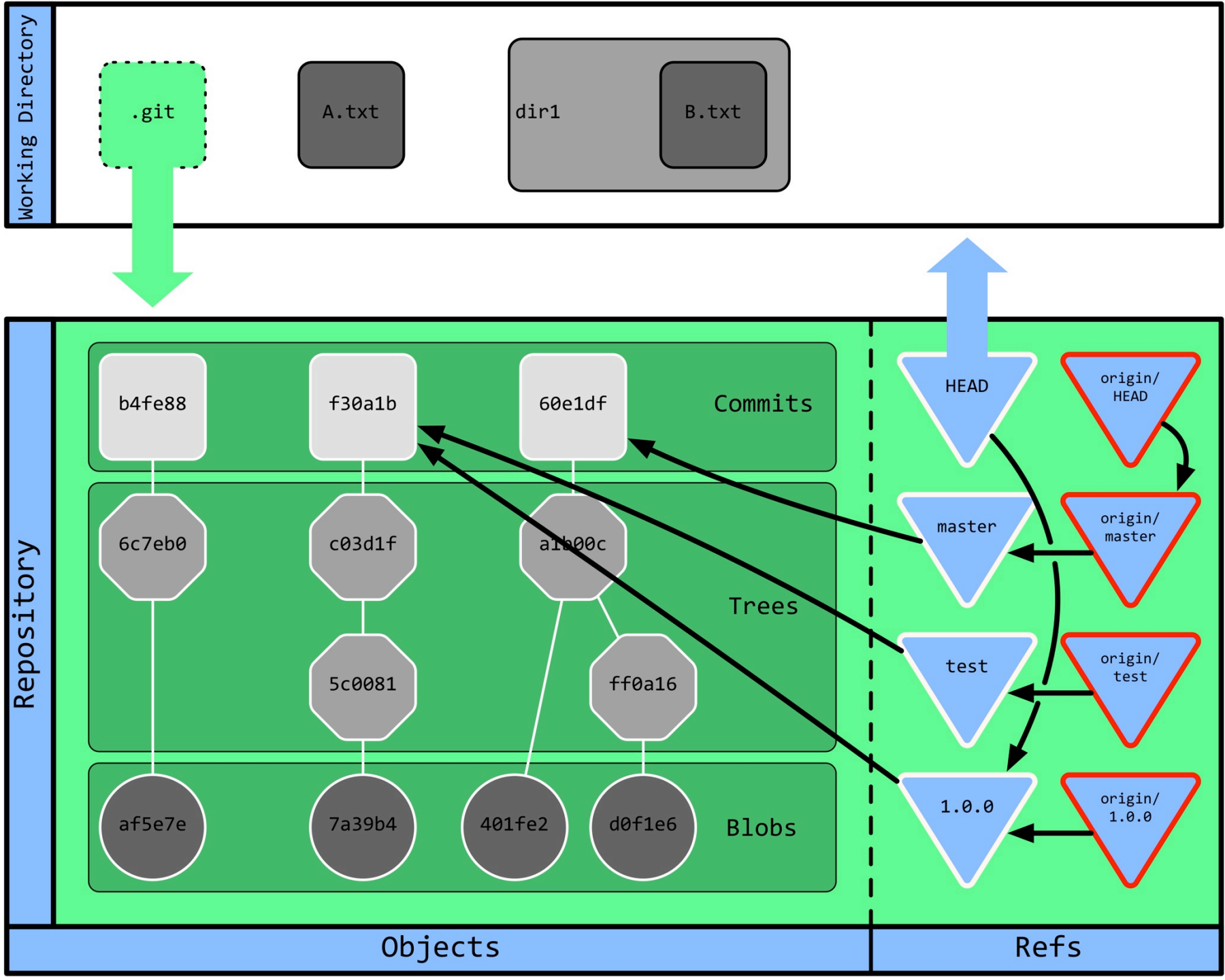
As a side note:

`git pull` is just a `fetch` followed by a `merge`

Everything looks good, but let's see what happens when we want to switch to a previous commit, such as tag `1.0.0`

```
git checkout 1.0.0
```


Working Directory
.git
A.txt
dir1
B.txt
dir2
C.txt
Repository
b4fe88
f30a1b
60e1df
Commits
6c7eb0
c03d1f
a1b00c
Trees
5c0081
ff0a16
af5e7e
7a39b4
401fe2
d0f1e6
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
origin/
test
1.0.0
origin/
1.0.0
Objects
Refs


Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
f30a1b
60e1df
Commits
6c7eb0
c03d1f
a1b00c
Trees
5c0081
ff0a16
af5e7e
7a39b4
401fe2
d0f1e6
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
origin/
test
1.0.0
origin/
1.0.0
Objects
Refs

A Few Things to Note

- All the new refs, commits, trees, and blobs still exist, even though we “went back” to a previous commit

- **C.txt** and **dir2 STILL EXIST!** They are blob **d0f1e6** and tree **ff0a16**, respectively

- To switch back to the latest commit, we can just type **git checkout master**

A Few Things to Note

- The **checkout** command did two things:
  - 1. **HEAD** pointed to the ref **1.0.0**
  - 2. The working directory (the files you see) were replaced with ones from the blobs referenced by the commit/trees that **1.0.0** points to

- Typing any of the following three commands would have done the same thing:
  - **git checkout test**
  - **git checkout f30a1b** **# <-- detached HEAD!**
  - **git checkout HEAD^** **# <-- detached HEAD!**

  - That last one is shorthand for “HEAD’s parent”

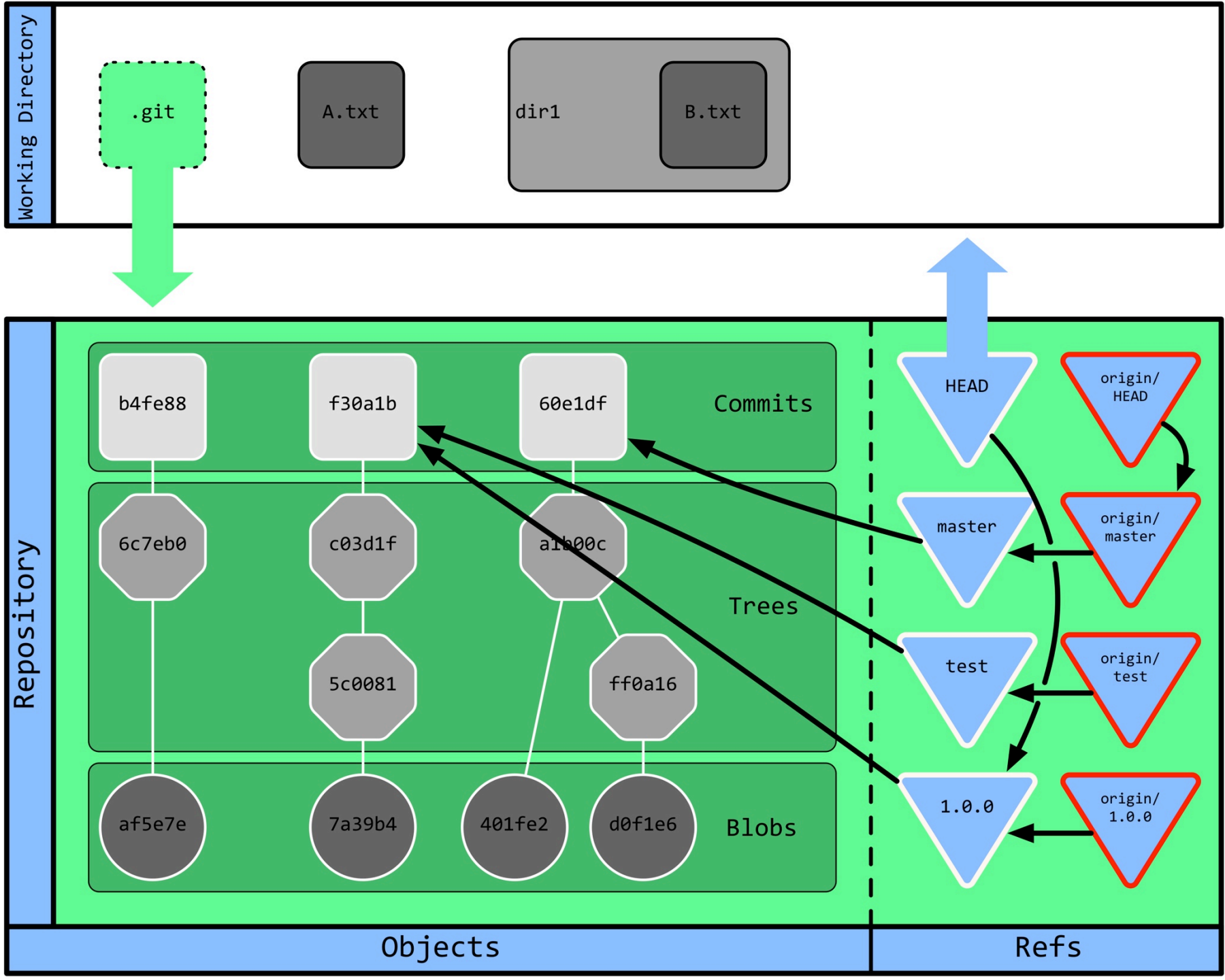
Detached HEAD State

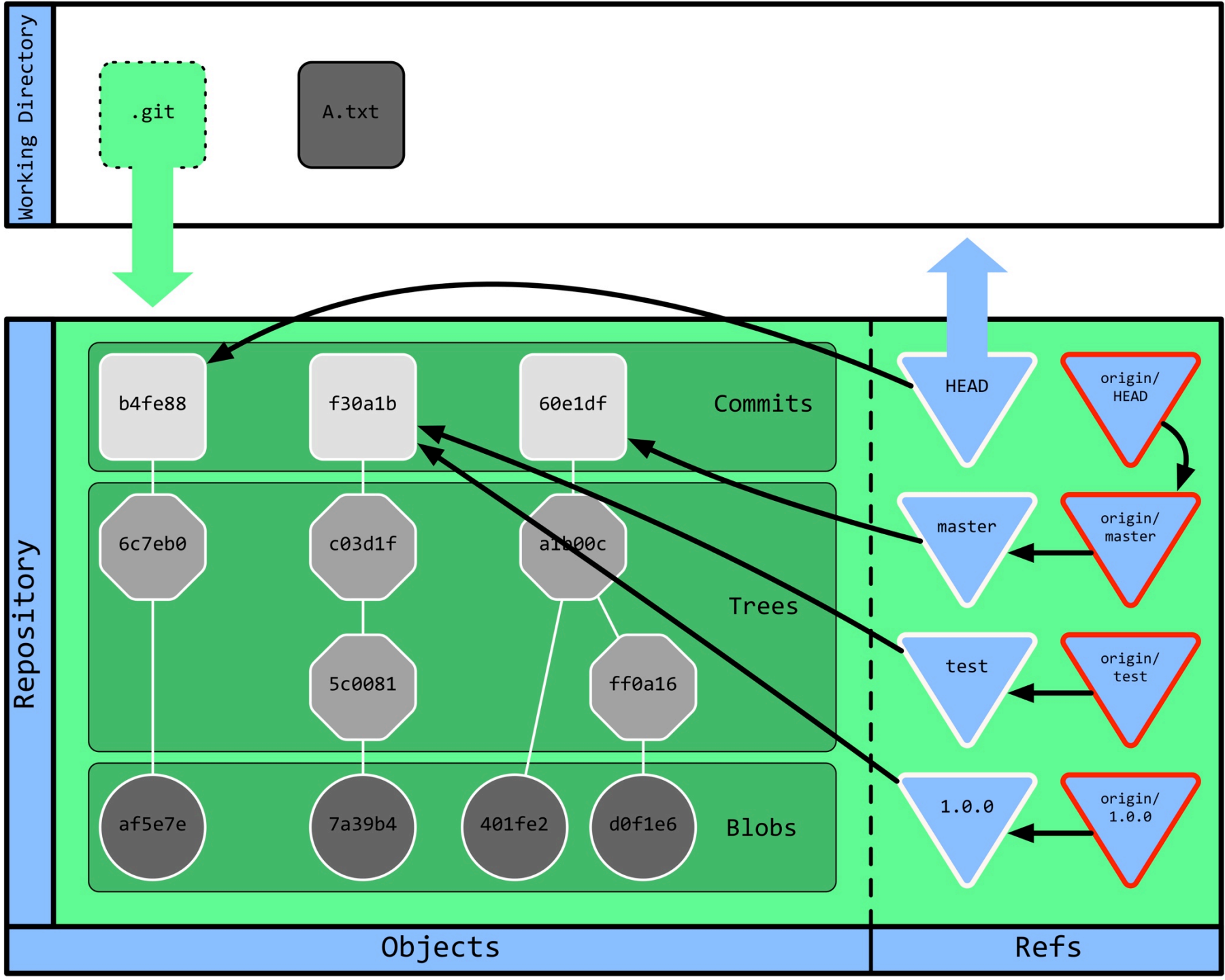
- At some point you may see this message:

```
$ git checkout HEAD^
Note: checking out 'HEAD^'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 3e6d961... Add file B

$ cat .git/HEAD
3e6d9614ec3cfc76e440aca6dafa8e6e0c14f34a
```

- A detached HEAD state is when **HEAD** points directly to a **commit** instead of a **branch ref**.

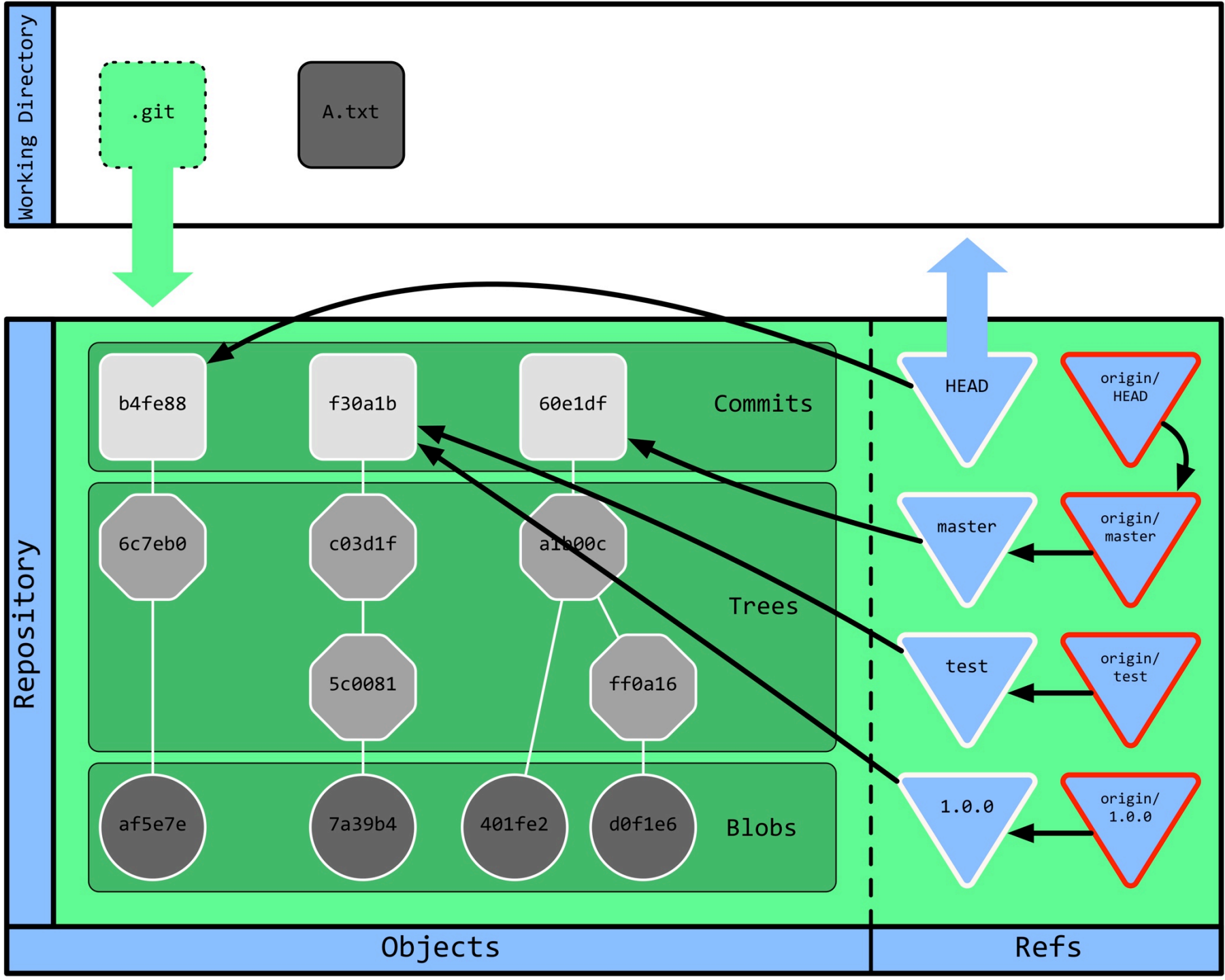
```
git checkout b4fe88
```

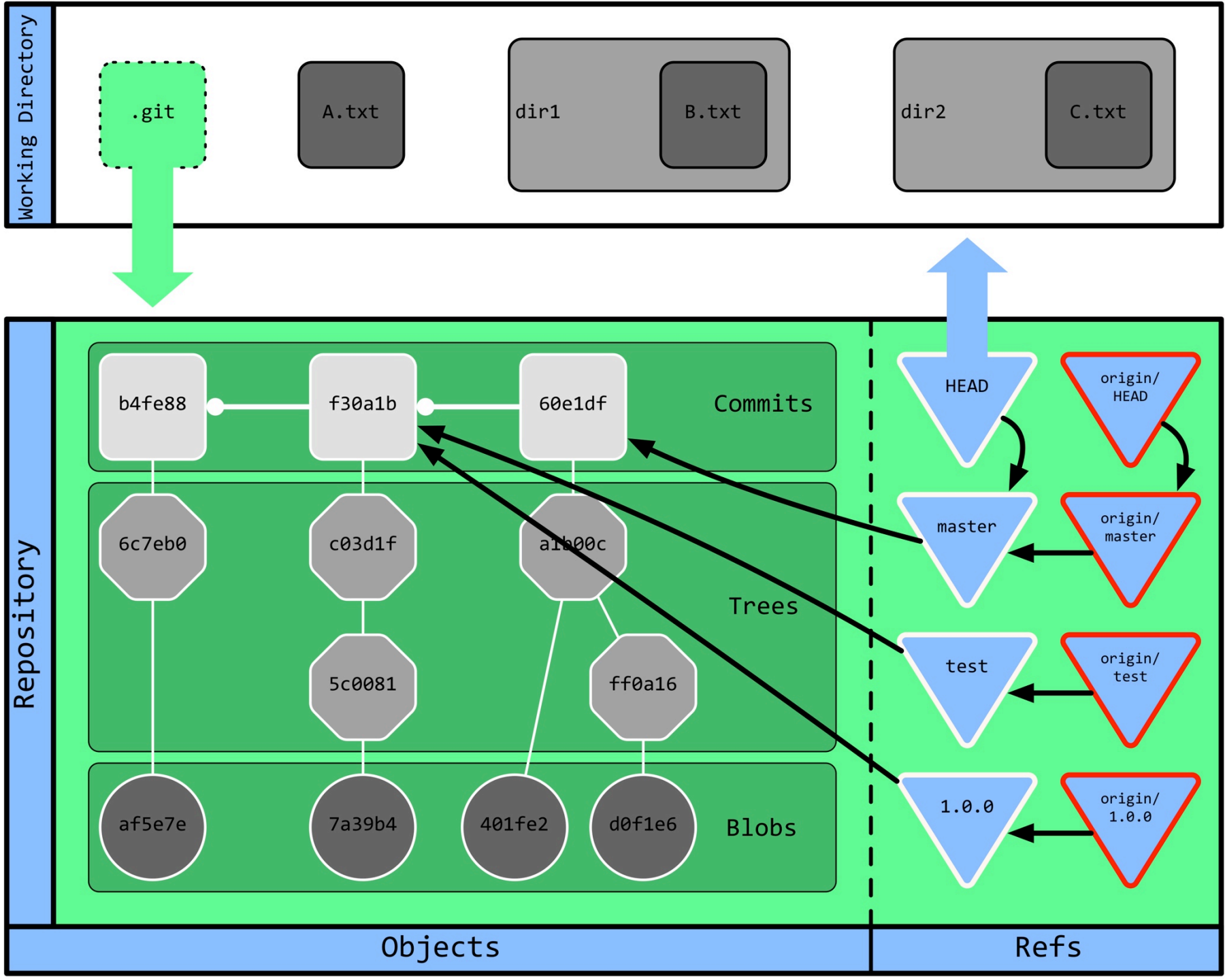

Working Directory
.git
A.txt
dir1
B.txt
Repository
b4fe88
f30a1b
60e1df
Commits
6c7eb0
c03d1f
a1b00c
Trees
5c0081
ff0a16
af5e7e
7a39b4
401fe2
d0f1e6
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
origin/
test
1.0.0
origin/
1.0.0
Objects
Refs


Working Directory
.git
A.txt
Repository
b4fe88
f30a1b
60e1df
Commits
6c7eb0
c03d1f
a1b00c
Trees
5c0081
ff0a16
af5e7e
7a39b4
401fe2
d0f1e6
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
origin/
test
1.0.0
origin/
1.0.0
Objects
Refs

Finally, we'll "attach our HEAD" and go back to `master`

```
git checkout master
```


Working Directory
.git
A.txt
Repository
b4fe88
f30a1b
60e1df
Commits
6c7eb0
c03d1f
a1b00c
Trees
5c0081
ff0a16
af5e7e
7a39b4
401fe2
d0f1e6
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
origin/
test
1.0.0
origin/
1.0.0
Objects
Refs


Working Directory
.git
A.txt
dir1
B.txt
dir2
C.txt
Repository
b4fe88
f30a1b
60e1df
Commits
6c7eb0
c03d1f
a1b00c
Trees
5c0081
ff0a16
af5e7e
7a39b4
401fe2
d0f1e6
Blobs
HEAD
origin/
HEAD
master
origin/
master
test
origin/
test
1.0.0
origin/
1.0.0
Objects
Refs

Questions?

References

- https://git-scm.com/book/en/v2/Git-Internals-Git-Objects
- https://jwiegley.github.io/git-from-the-bottom-up/