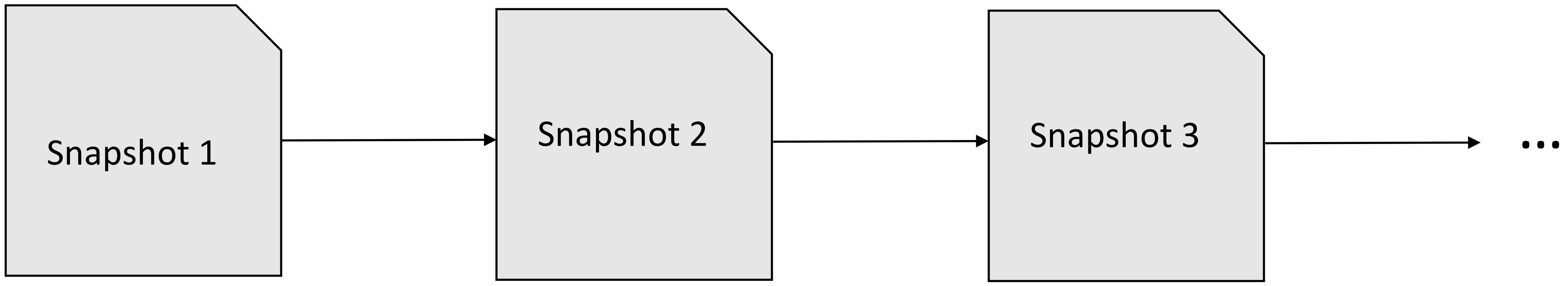
THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Lab 1 - Git

(The longest damn lab ever)

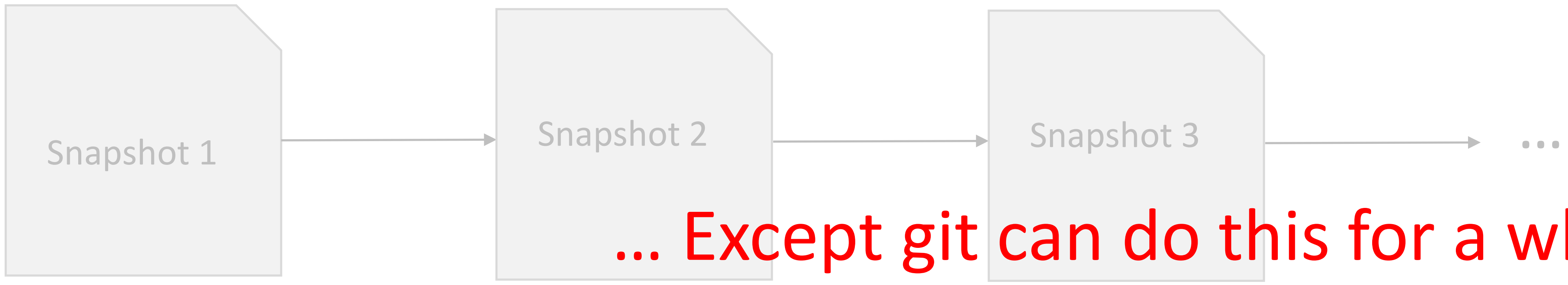
Git is a Version Control System (VCS)

- The simplest case: editing a single file over time.
- You periodically ask git to take snapshots of the file content.
- Then you can view or rollback to any snapshot from the past.

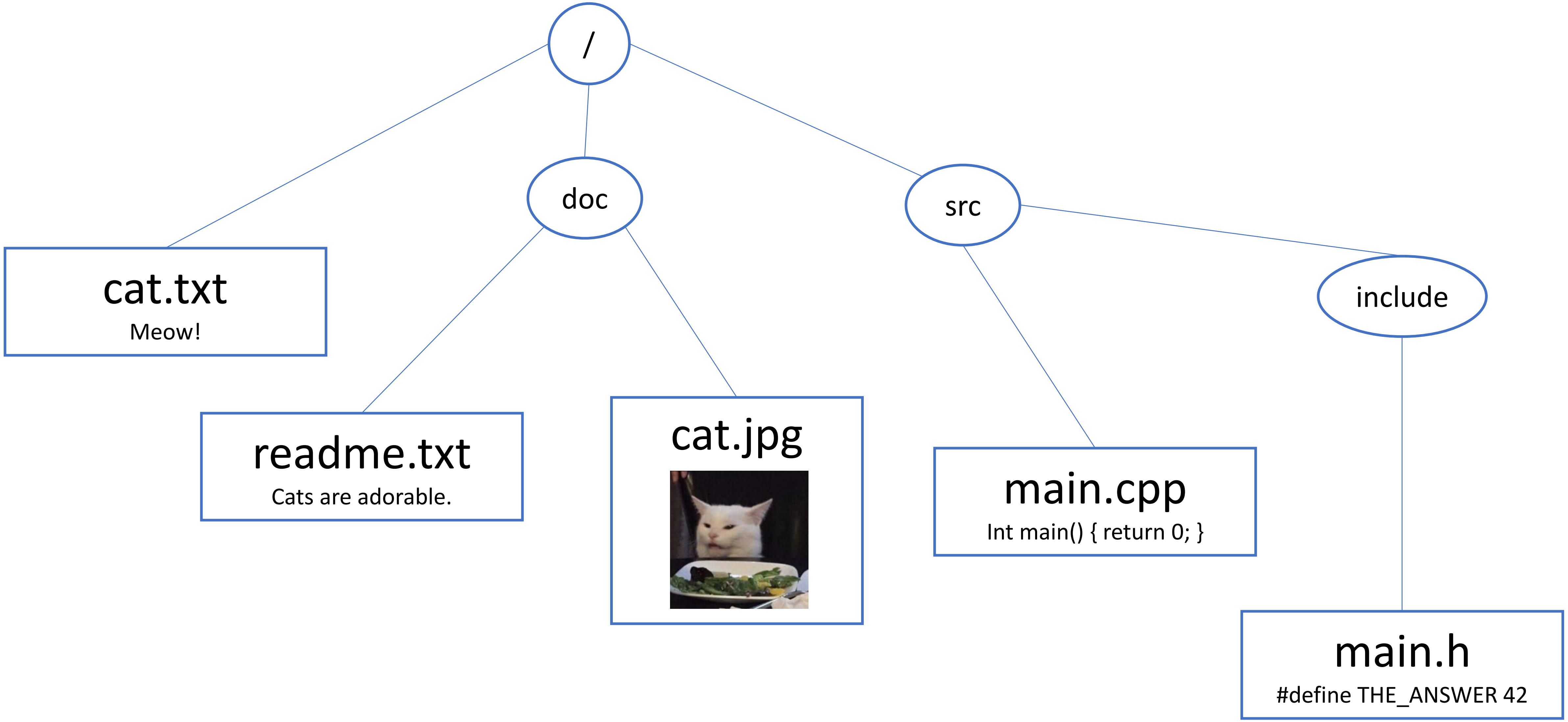


Git is a Version Control System (VCS)

- The simplest case: editing a single file over time.
- You periodically ask git to take snapshots of the file content.
- Then you can view or rollback to any snapshot from the past.



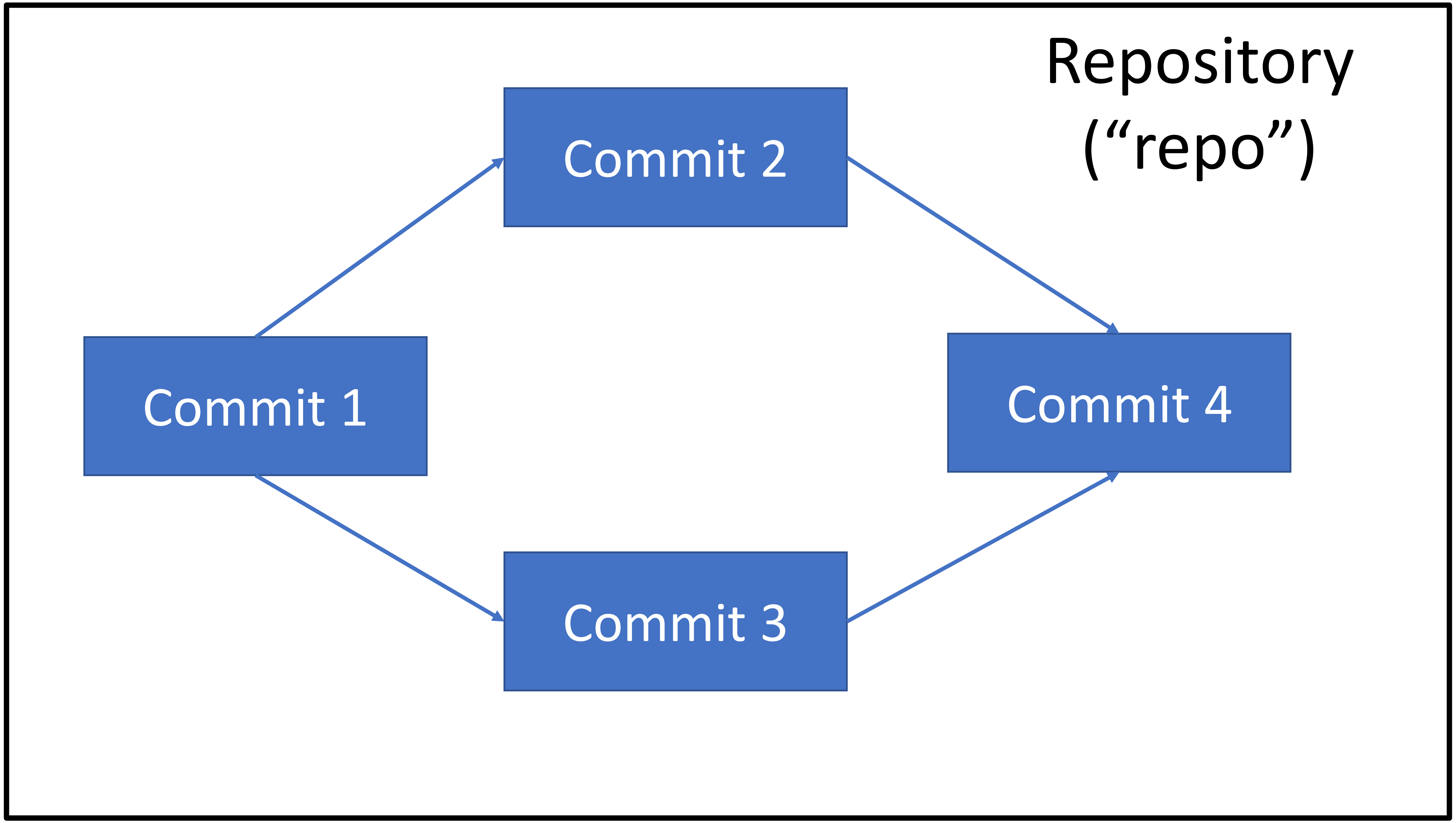
… Except git can do this for a whole directory, instead of a single file


/
doc
src
include
cat.txt
Meow!
readme.txt
Cats are adorable.
cat.jpg
main.cpp
Int main() { return 0; }
main.h
#define THE_ANSWER 42

Path	Content
/cat.txt	Meow!
/doc/readme.txt	Cats are adorable.
/doc/cat.jpg	
/src/main.cpp	int main() { return 0; }
/src/include/main.h	#define THE_ANSWER 42

Commit

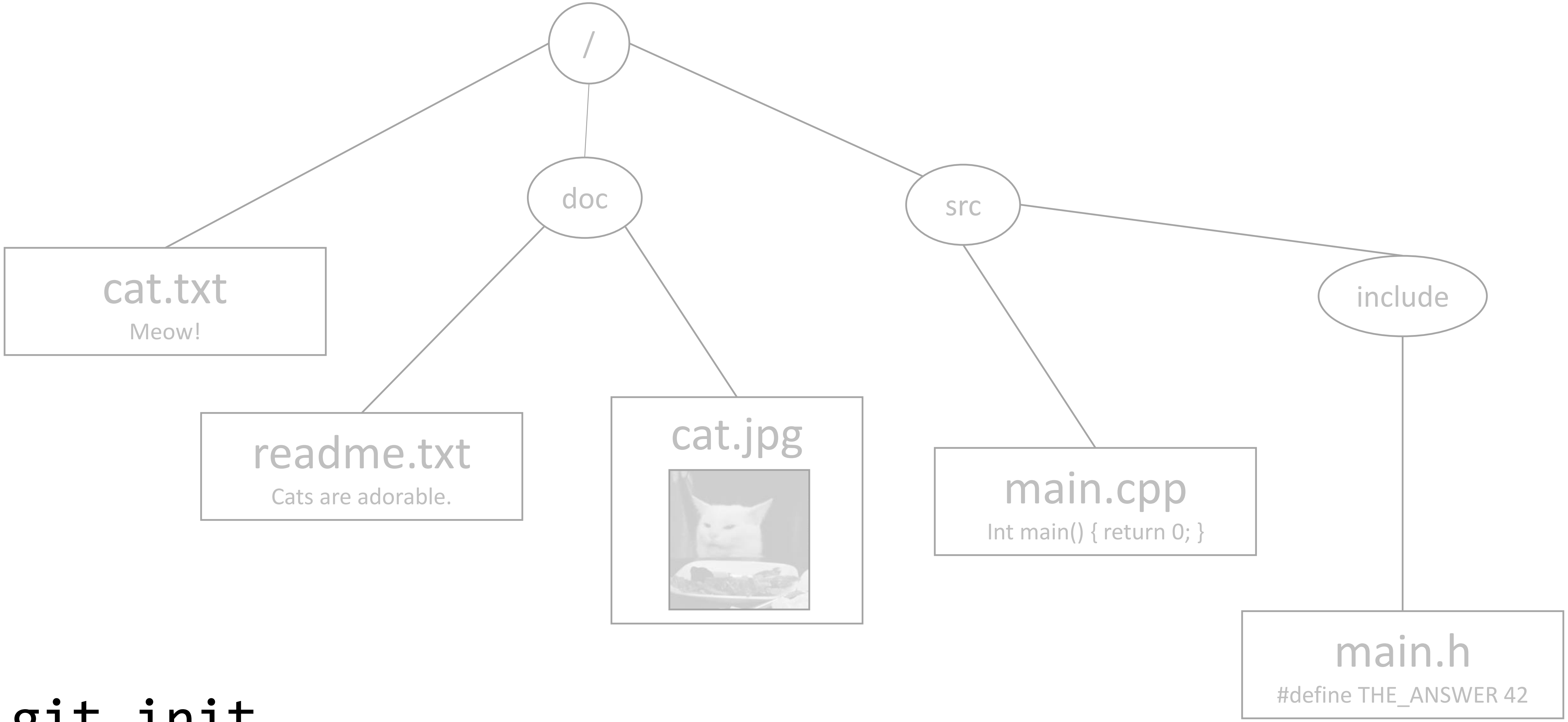
Path	Content
/cat.txt	Meow!
/doc/readme.txt	Cats are adorable.
/doc/cat.jpg	
/src/main.cpp	int main() { return 0; }
/src/include/main.h	#define THE_ANSWER 42


Repository
("repo")
Commit 1
Commit 2
Commit 3
Commit 4

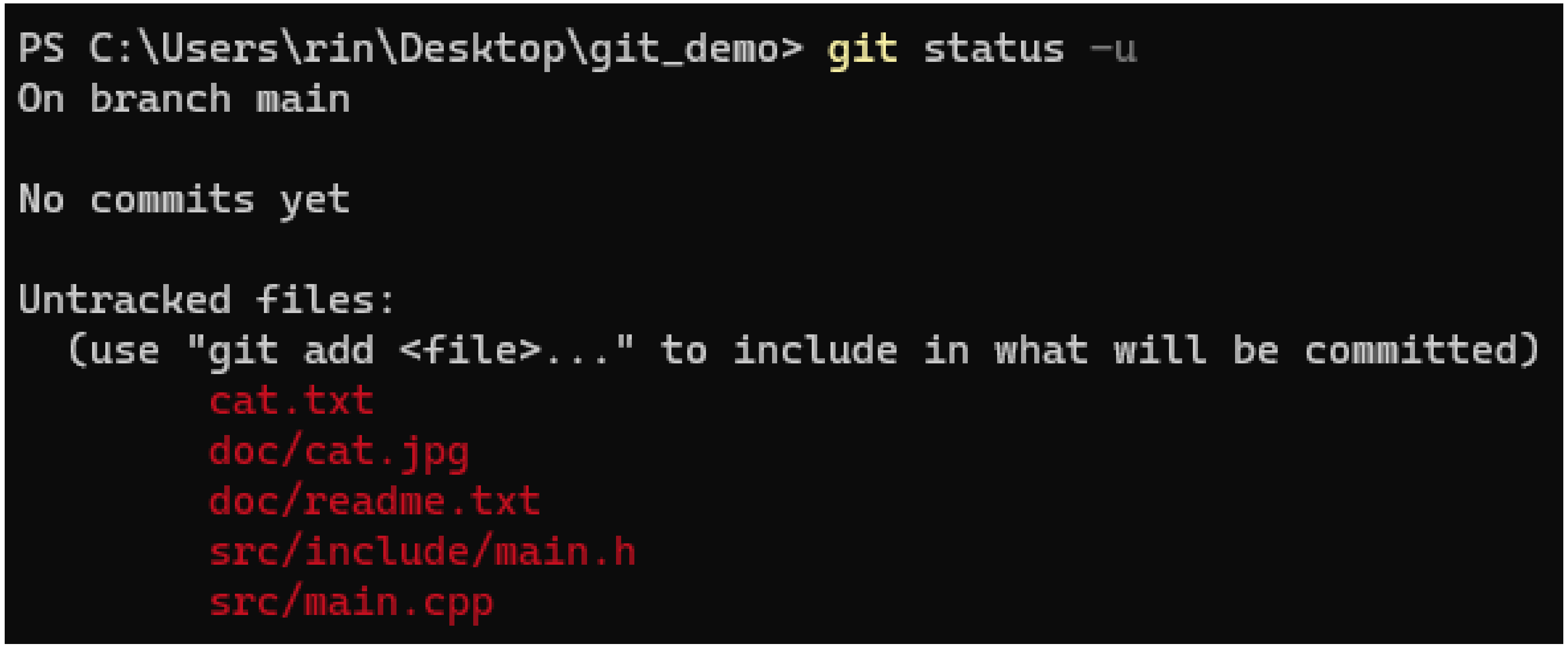
Using the git command line interface

- **`cd your_directory`**
- **`git init` to create a repository.**
- `git add filename` to stage changes to the repo.
- `git commit -m “your message”` to create a commit.

Grey = Untracked



```
git init
```



```
PS C:\Users\rin\Desktop\git_demo> git status -u
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        cat.txt
        doc/cat.jpg
        doc/readme.txt
        src/include/main.h
        src/main.cpp
```

`git status -u`

Tells git to list all untracked files in the directory (“u” for “untracked”)

Using the git command line interface

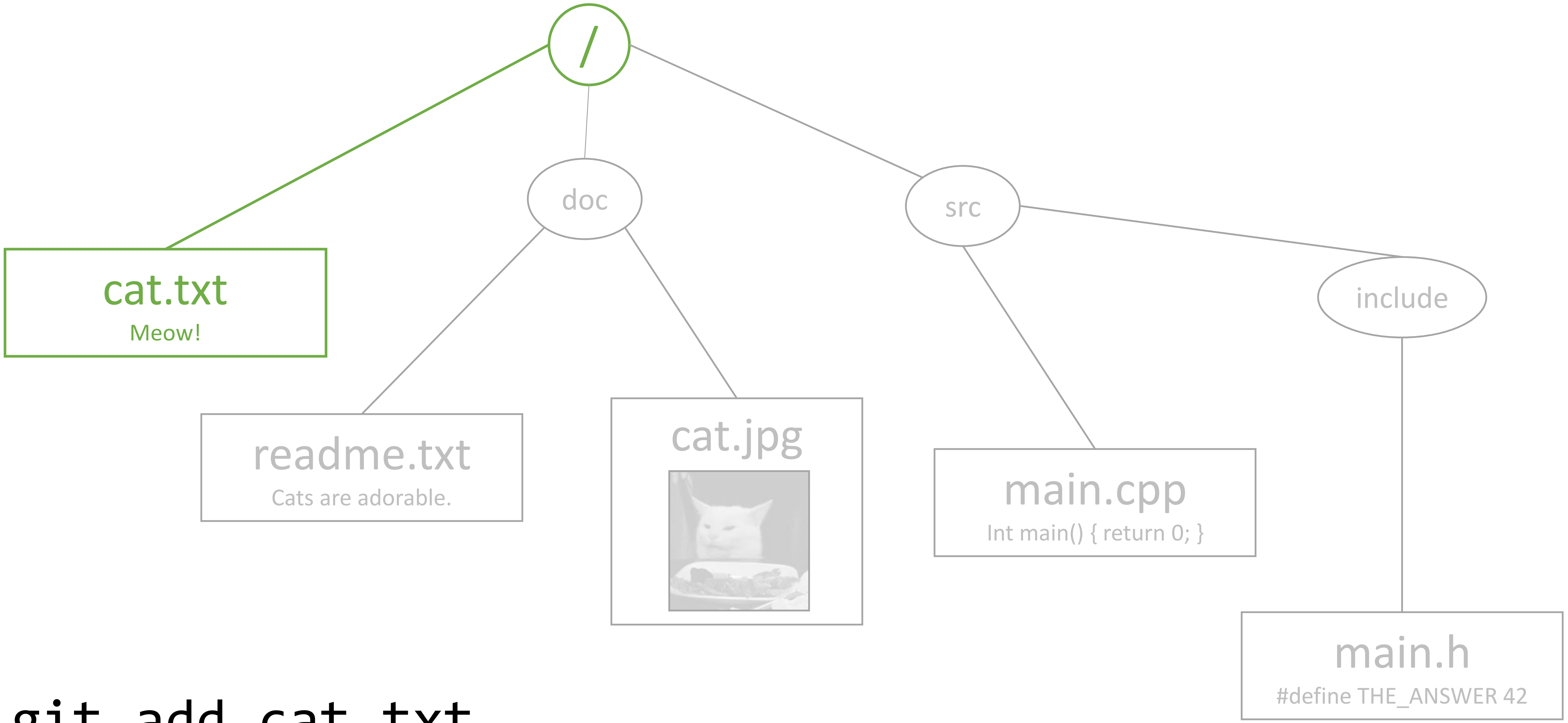
- `cd your_directory`
- `git init` to create a repository.
- **`git add filename` to *stage* changes to the repo.**
- `git commit –m “your message”` to create a commit.

Using the git command line interface

- `cd your_directory`
- `git init` to create a repository.
- **`git add filename` to *stage* changes to the repo.**
- `git commit –m "your message"` to create a commit.

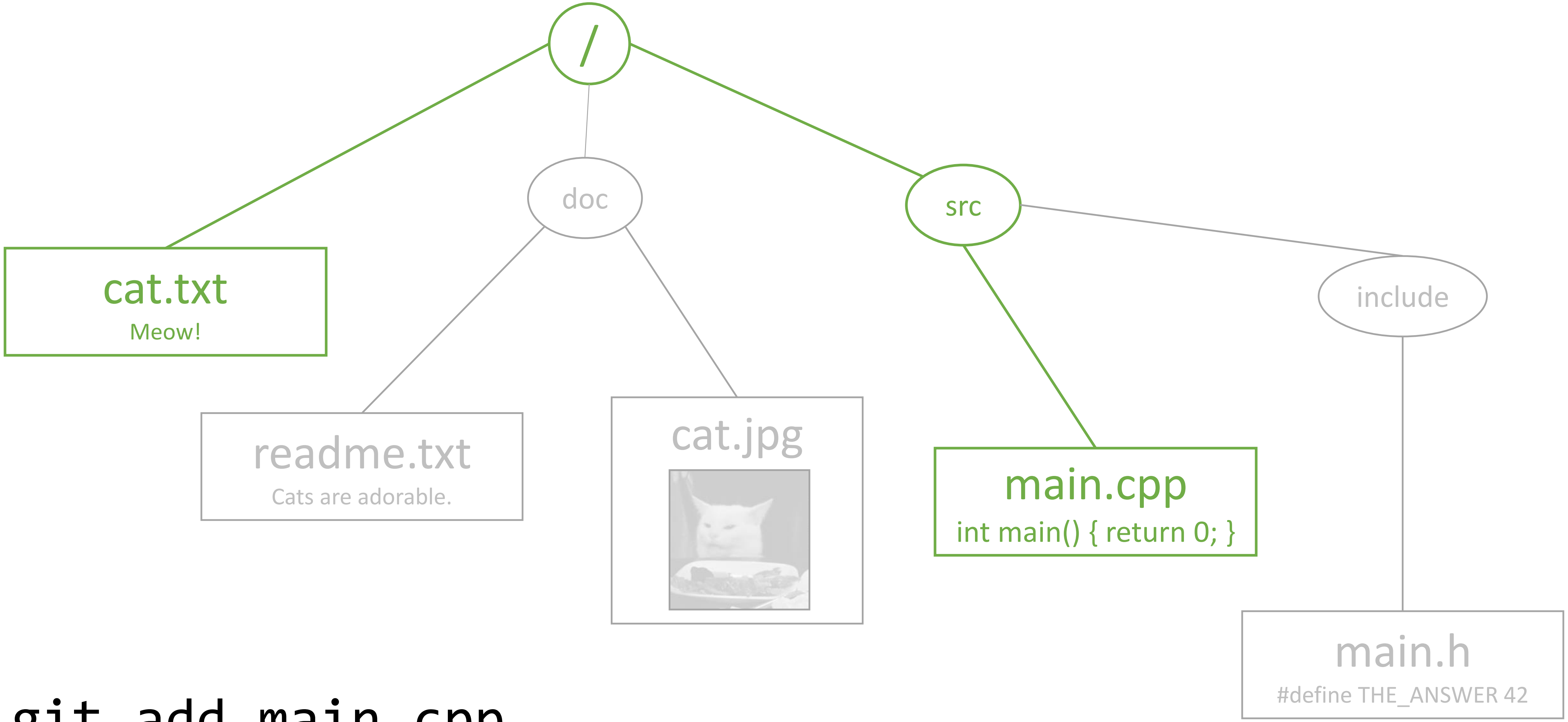
You must explicitly tell git to *stage* changes to your repo!

Green = Staged



```
git add cat.txt
```

Green = Staged



```
git add main.cpp
```

Using the git command line interface

- `cd your_directory`
- `git init` to create a repository.
- `git add filename` to track files.
- **`git commit –m “your message”` to create a commit.**

Using the git command line interface

- `cd your_directory`
- `git init` to create a repository.
- `git add filename` to track files.
- **`git commit –m “your message”` to create a commit.**

The commit would include only changes you have staged!

Entries marked as grey are for reference only. They are **NOT** in the commit.

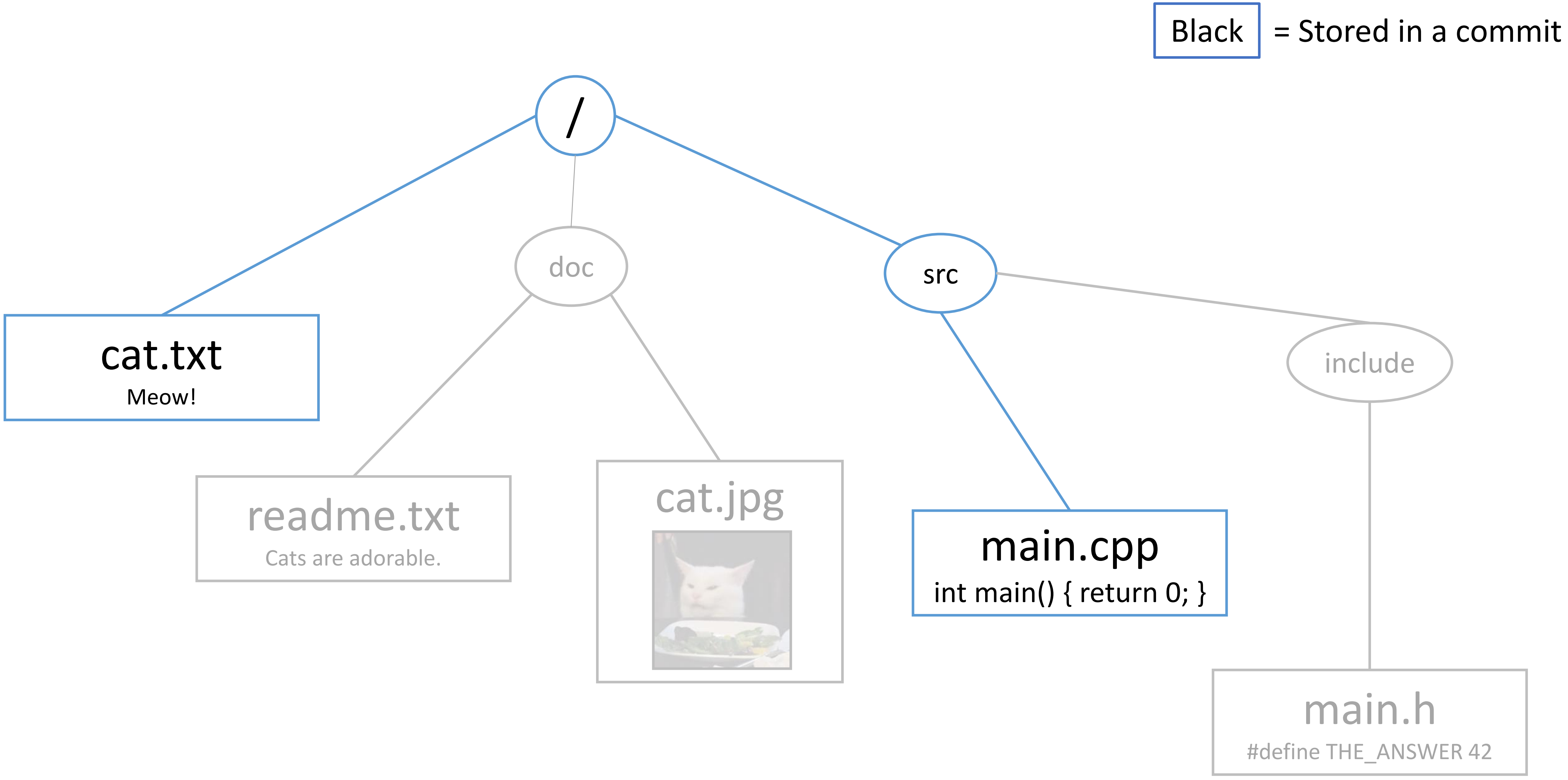
Path	Content
/cat.txt	Meow!
/doc/readme.txt	Cats are adorable.
/doc/cat.jpg	
/src/main.cpp	int main() { return 0; }
/src/include/main.h	#define THE_ANSWER 42

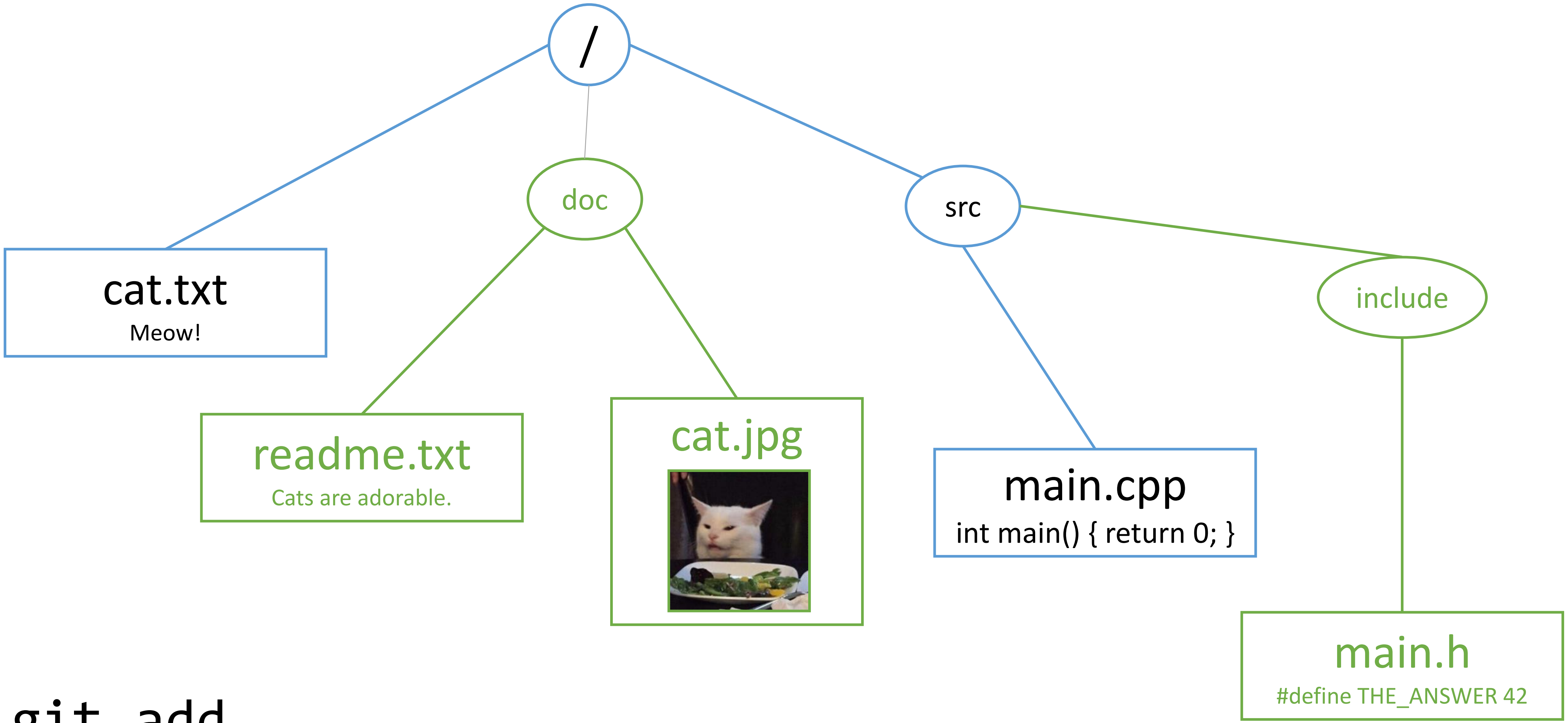
```
git commit -m “initial commit”
```

initial commit

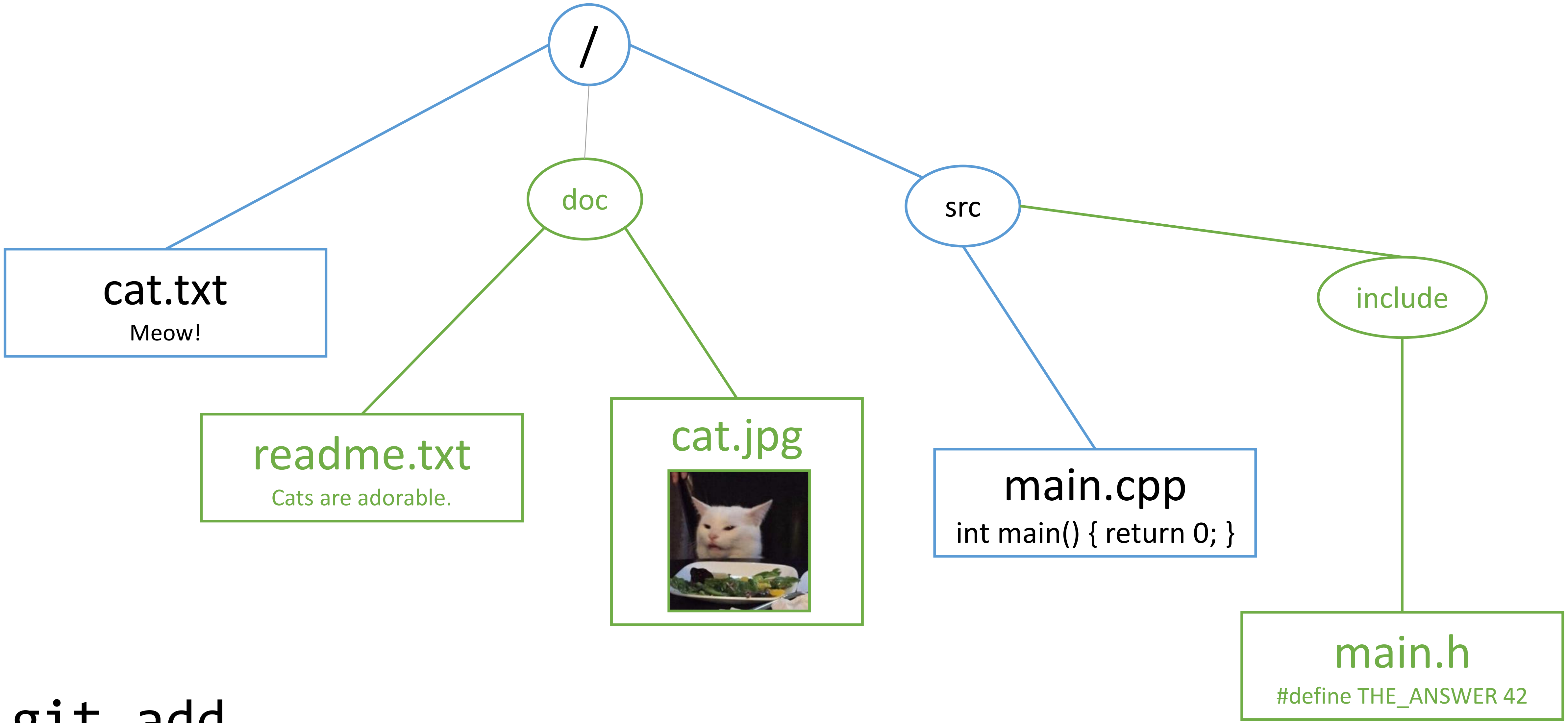
Path	Content
/cat.txt	Meow!
/src/main.cpp	int main() { return 0; }

The repository now looks like this


Black
= Stored in a commit
/
doc
src
include
cat.txt
Meow!
readme.txt
Cats are adorable.
cat.jpg
main.cpp
int main() { return 0; }
main.h
#define THE_ANSWER 42


/
doc
src
include
cat.txt
Meow!
readme.txt
Cats are adorable.
cat.jpg
main.cpp
int main() { return 0; }
main.h
#define THE_ANSWER 42

```
git add .
```



```
git add .
```

(Tells git to automatically stage all changes it detects)

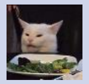
Path	Content
/cat.txt	Meow!
/doc/readme.txt	Cats are adorable.
/doc/cat.jpg	
/src/main.cpp	int main() { return 0; }
/src/include/main.h	#define THE_ANSWER 42

```
git commit -m “added all”
```

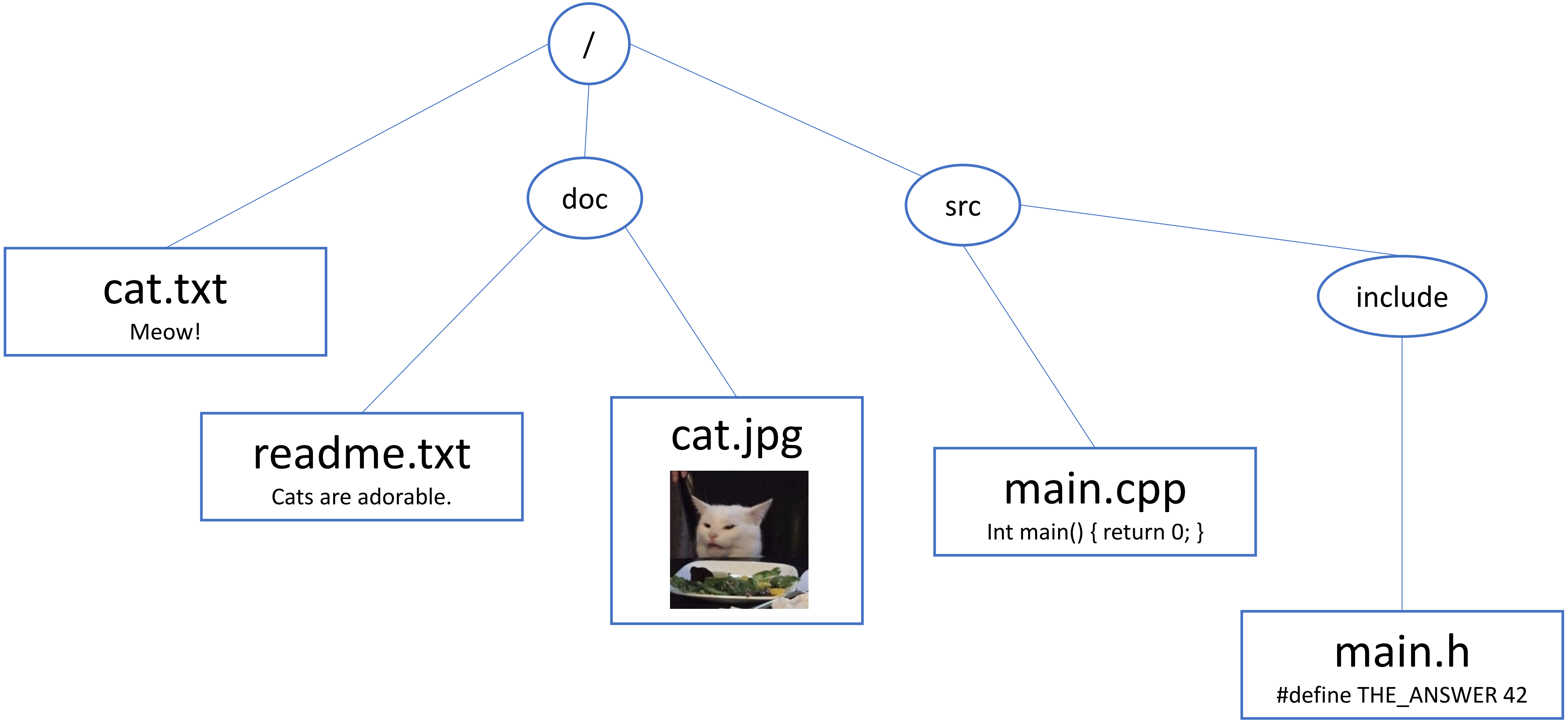
initial commit

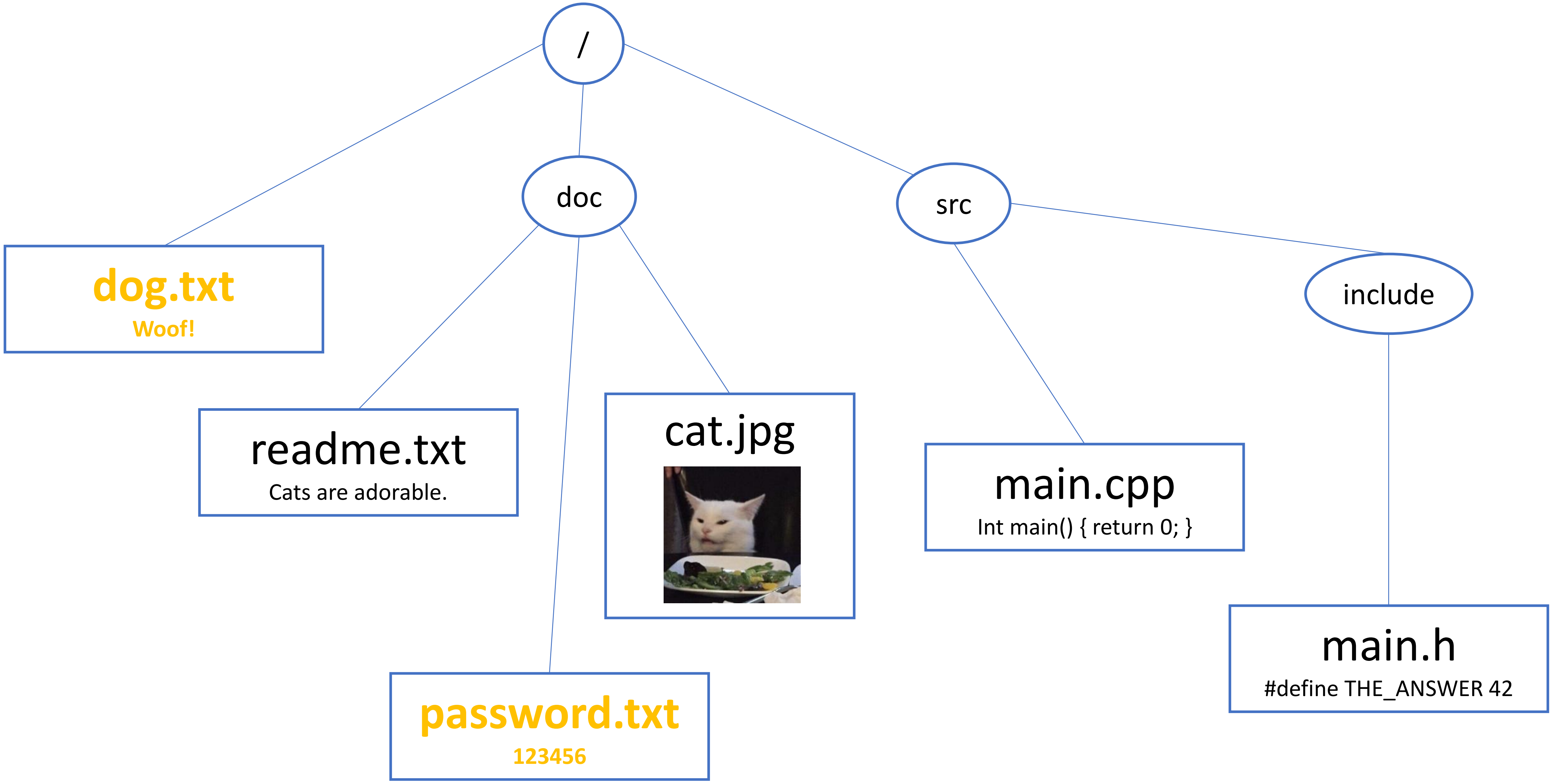
Path	Content
/cat.txt	Meow!
/src/main.cpp	int main() { return 0; }

added all

Path	Content
/cat.txt	Meow!
/doc/readme.txt	Cats are adorable.
/doc/cat.jpg	
/src/main.cpp	int main() { return 0; }
/src/include/main.h	#define THE_ANSWER 42

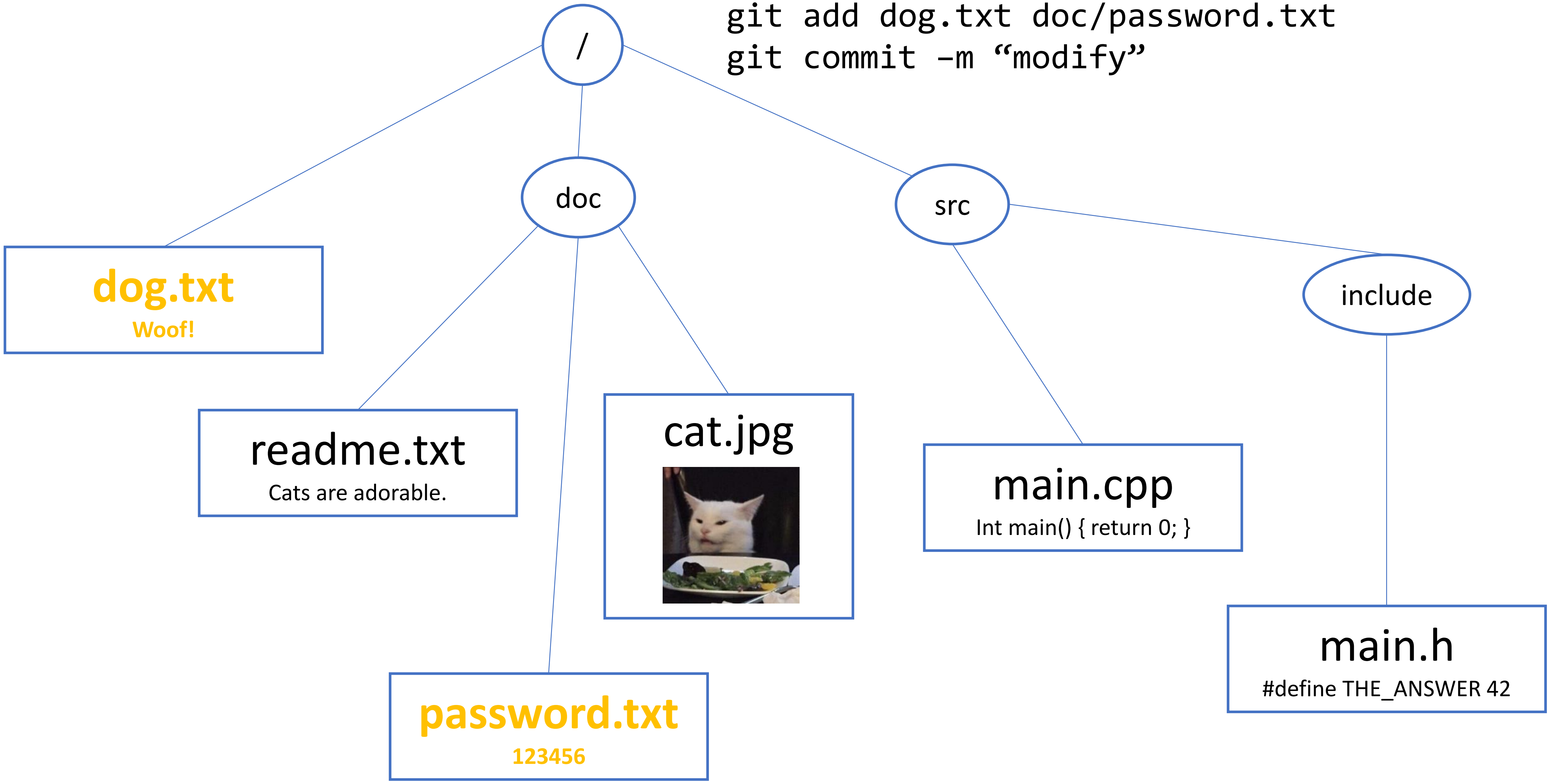
The repository now looks like this


/
doc
src
include
cat.txt
Meow!
readme.txt
Cats are adorable.
cat.jpg
main.cpp
Int main() { return 0; }
main.h
#define THE_ANSWER 42


/
doc
src
include
dog.txt
Woof!
readme.txt
Cats are adorable.
cat.jpg
password.txt
123456
main.cpp
Int main() { return 0; }
main.h
#define THE_ANSWER 42

What will the commit look like if I do:

```
git add dog.txt doc/password.txt
git commit -m “modify”
```



What happens after `git commit –m “modify”`?

- There are no doubts that two new files are added to the commit:
  - dog.txt
  - doc/password.txt
- But what about “cat.txt”?

What happens after `git commit -m “modify”`?

- There are no doubts that two new files are added to the commit:
  - dog.txt
  - doc/password.txt
- But what about “cat.txt”?
  - Git does not know that dog.txt is a modified version of cat.txt.
  - In Git’s point of view, you simply removed cat.txt and added a new file called dog.txt.
  - However, you have not **explicitly** asked git to track the removal of cat.txt.

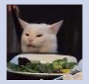
Path	Content
/doc/cat.txt	Meow!
/doc/dog.txt	Woof!
/doc/readme.txt	Cats are adorable.
/doc/cat.jpg	
/src/main.cpp	int main() { return 0; }
/src/include/main.h	#define THE_ANSWER 42

```
git commit -m “modify”
```

initial commit

Path	Content
/cat.txt	Meow!
/src/main.cpp	int main() { return 0; }

added all

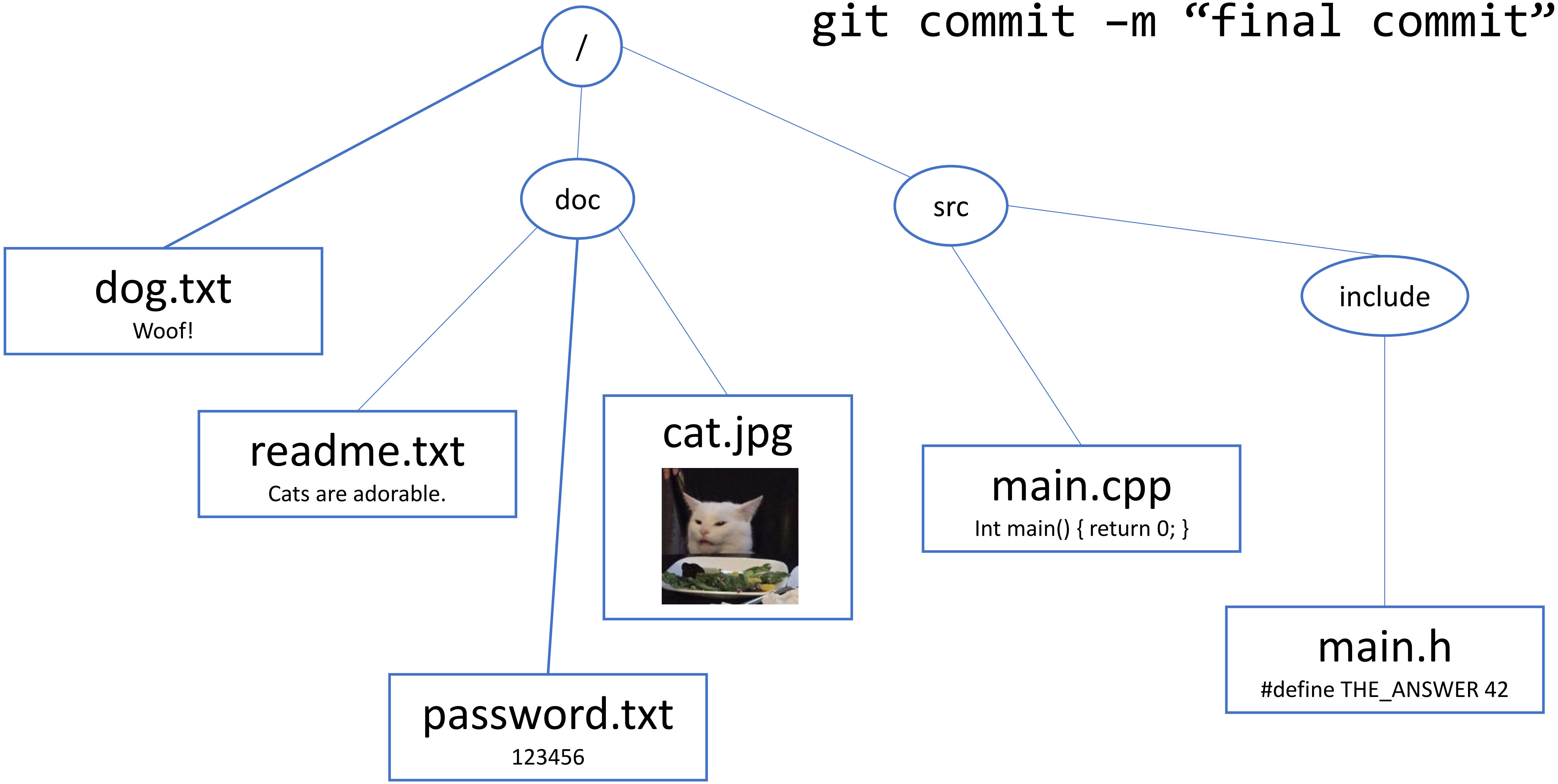
Path	Content
/cat.txt	Meow!
/doc/readme.txt	Cats are adorable.
/doc/cat.jpg	
/src/main.cpp	int main() { return 0; }
/src/include/main.h	#define THE_ANSWER 42

modify

Path	Content
/cat.txt	Meow!
/dog.txt	Woof!
/doc/readme.txt	Cats are adorable.
/doc/cat.jpg	
/src/main.cpp	int main() { return 0; }
/src/include/main.h	#define THE_ANSWER 42

The repository now looks like this

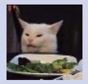
```
git add .
git commit -m “final commit”
```



initial commit

Path	Content
/cat.txt	Meow!
/src/main.cpp	int main() { return 0; }

added all

Path	Content
/cat.txt	Meow!
/doc/readme.txt	Cats are adorable.
/doc/cat.jpg	
/src/main.cpp	int main() { return 0; }
/src/include/main.h	#define THE_ANSWER 42

modify

Path	Content
/cat.txt	Meow!
/dog.txt	Woof!
/doc/readme.txt	Cats are adorable.
/doc/cat.jpg	
/src/main.cpp	int main() { return 0; }
/src/include/main.h	#define THE_ANSWER 42

final commit

Path	Content
/dog.txt	Woof!
/doc/readme.txt	Cats are adorable.
/doc/password.txt	123456
/doc/cat.jpg	
/src/main.cpp	int main() { return 0; }
/src/include/main.h	#define THE_ANSWER 42

The repository now looks like this

Working with GitHub

- GitHub stores your repository the same way you store a repository on your local machine.
- When you ***clone*** a repo from GitHub, you copy the whole repo from GitHub onto your computer.
  - `git clone git@github.com:username/repo_name.git`
- When you ***pull*** from GitHub, you download all the commits that are present on GitHub but not in your local repo.
  - `git pull`
- When you ***push*** to GitHub, you upload all commits present in your local repo but not on GitHub.
  - `git push`

Ignoring files with .gitignore

- .gitignore is a file that contains a list of filenames.
- If git sees a file with .gitignore in a directory, it will ignore the files listed when you do “`git add .`”
- You can use “*” as a wildcard to match multiple files
  - E.g., “*.txt” would ignore all files with the “txt” extension
- You can put “/” at the end of a name to ignore entire directories
  - E.g., “ignored/” would ignore all **directories** named “ignored”