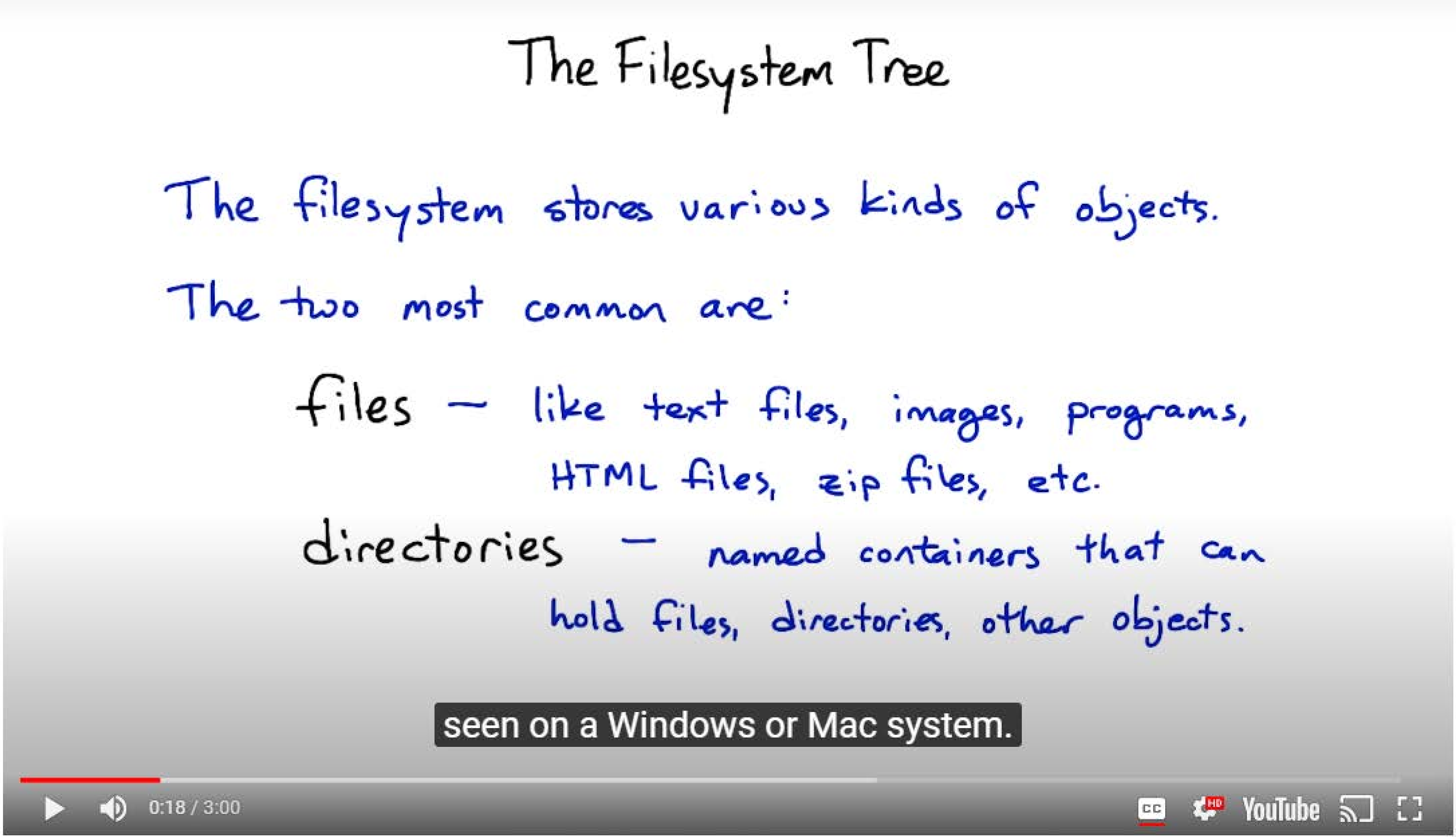


The Filesystem Tree

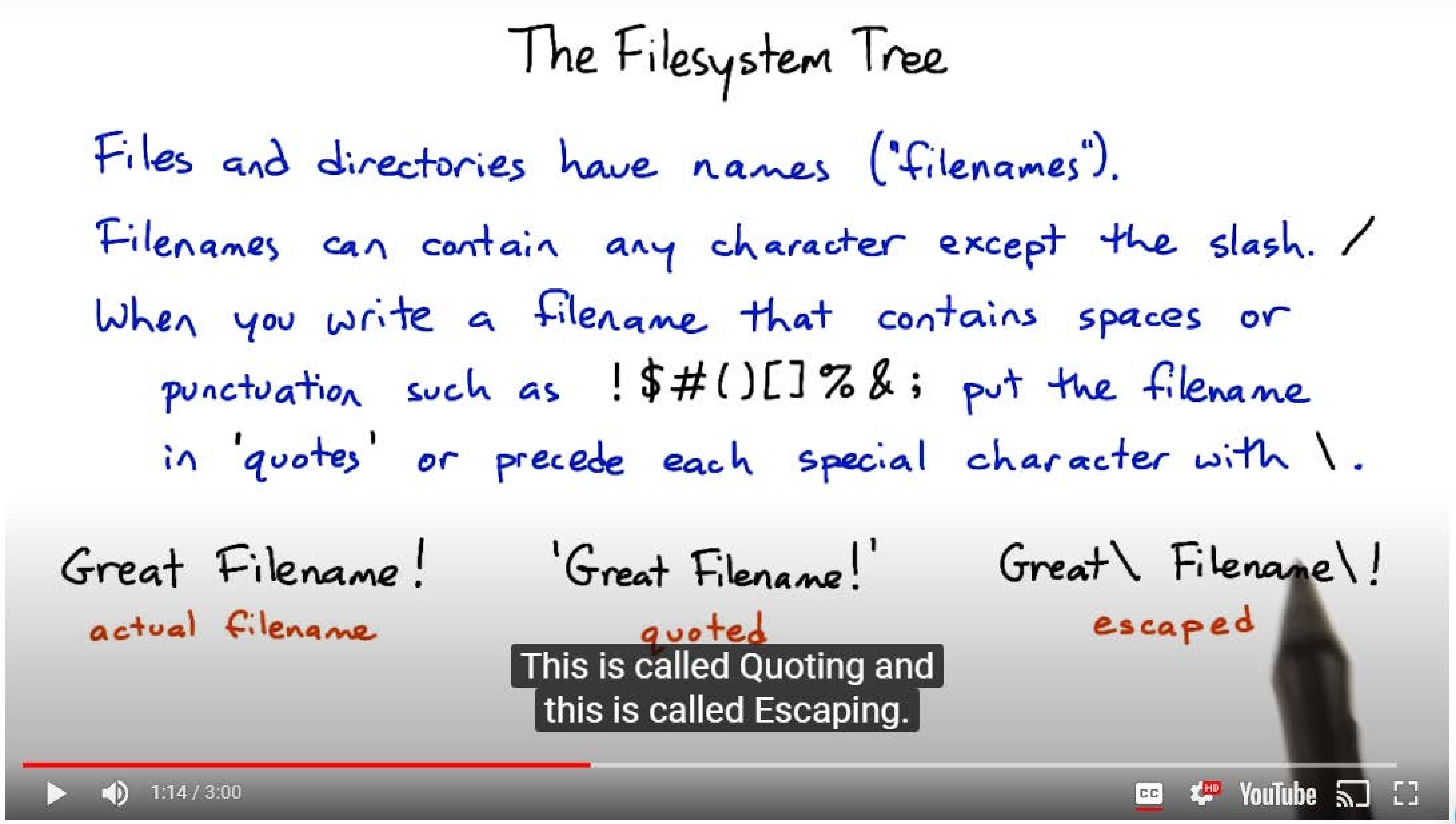
The filesystem stores various kinds of objects.

The two most common are:

files — like text files, images, programs,
HTML files, zip files, etc.

directories — named containers that can
hold files, directories, other objects.

seen on a Windows or Mac system.



The Filesystem Tree

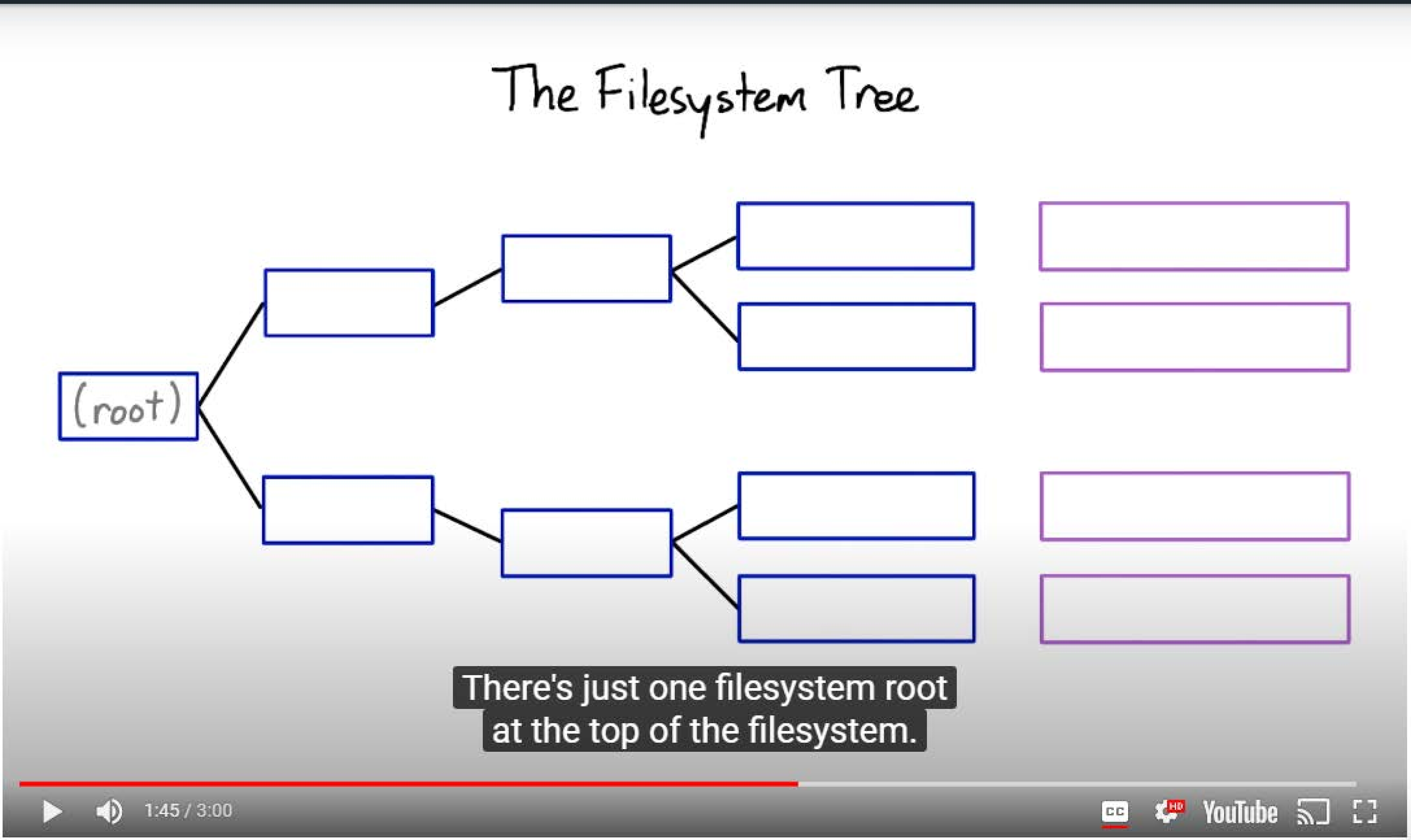
Files and directories have names ("filenames").

Filenames can contain any character except the slash. /

When you write a filename that contains spaces or punctuation such as ! $ # ( ) [ ] % & ; put the filename in 'quotes' or precede each special character with \ .

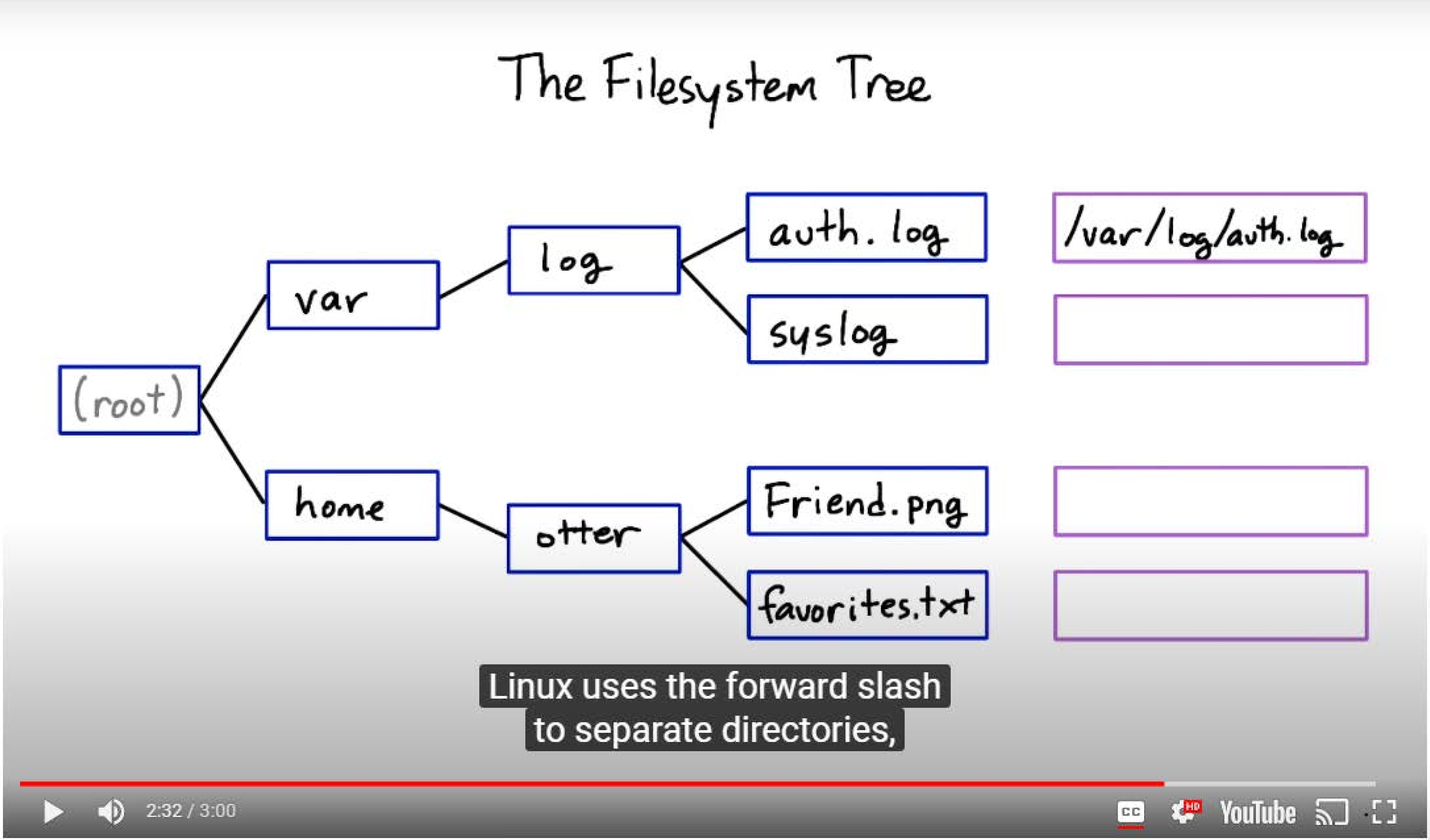
Great Filename!	'Great Filename!'	Great\ Filename\!
actual filename	quoted	escaped

This is called Quoting and this is called Escaping.



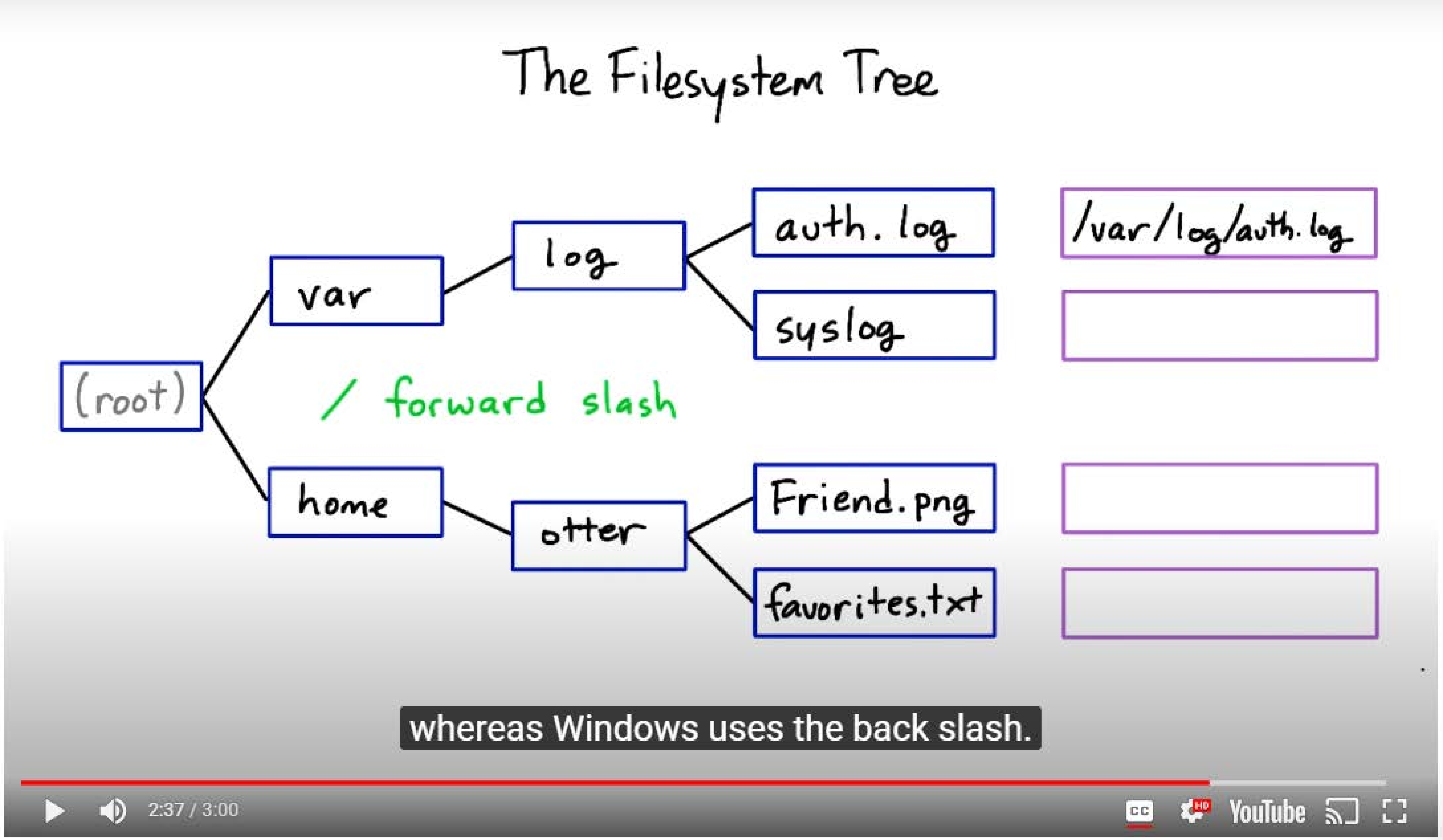
The Filesystem Tree

There's just one filesystem root
at the top of the filesystem.



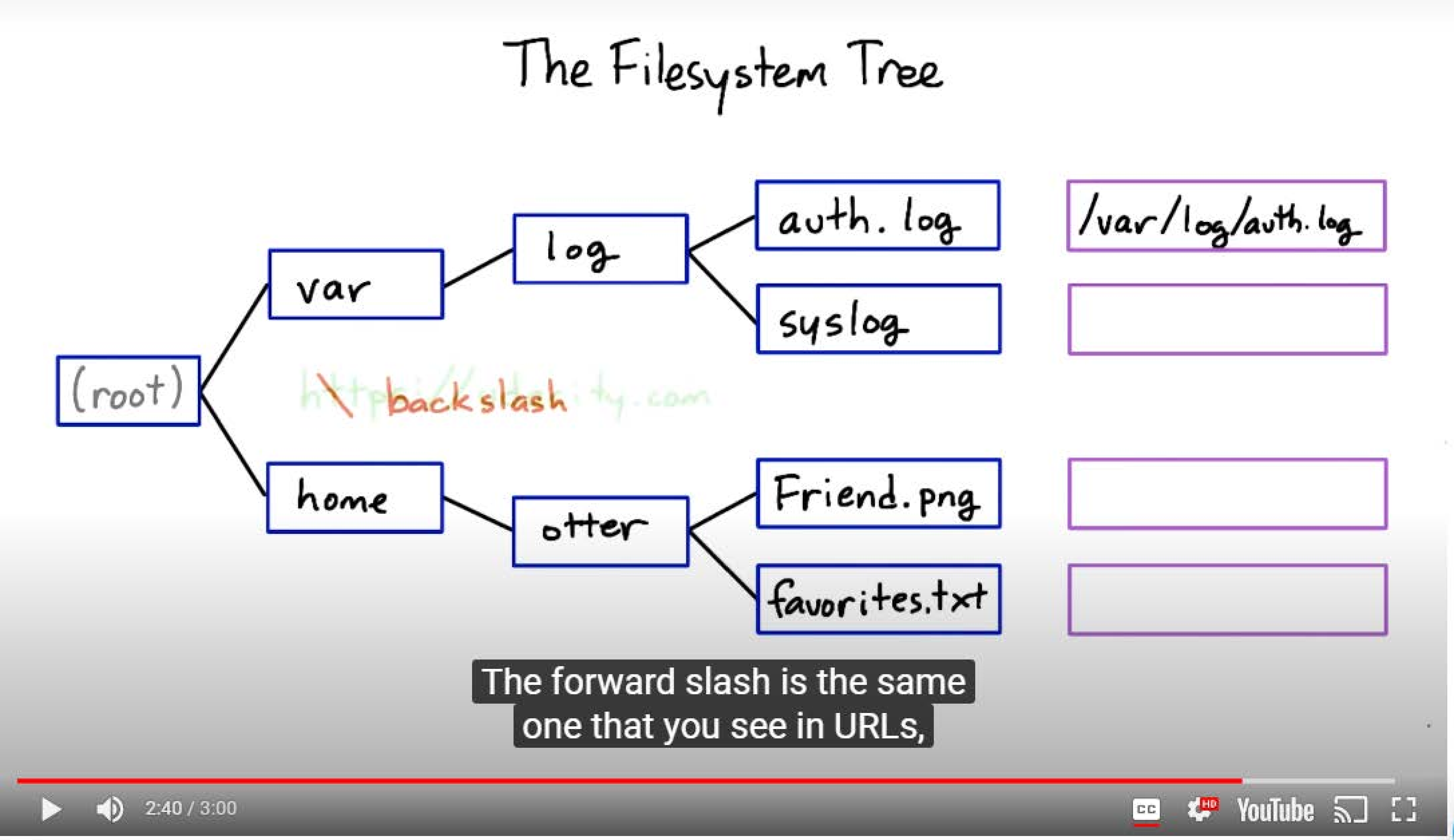
The Filesystem Tree

Linux uses the forward slash
to separate directories,



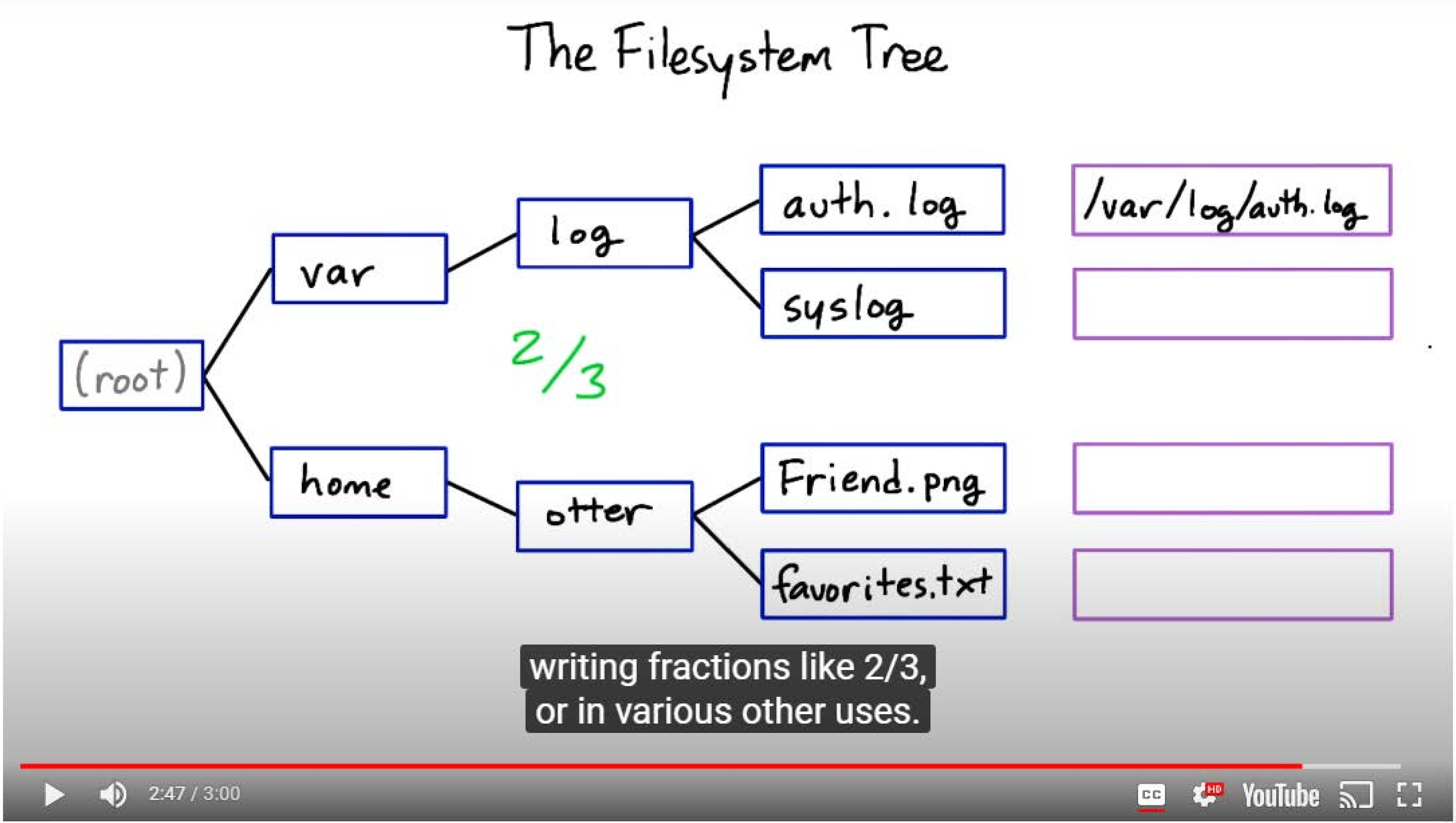
The Filesystem Tree

whereas Windows uses the back slash.



The Filesystem Tree

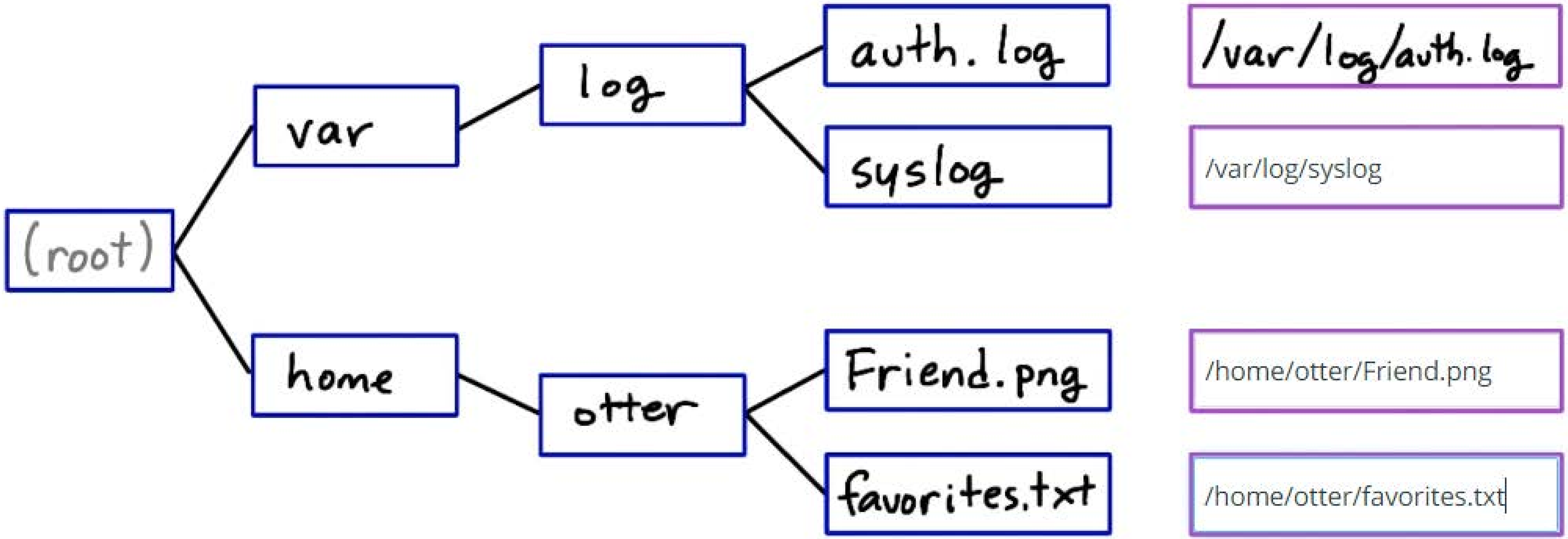
The forward slash is the same one that you see in URLs,

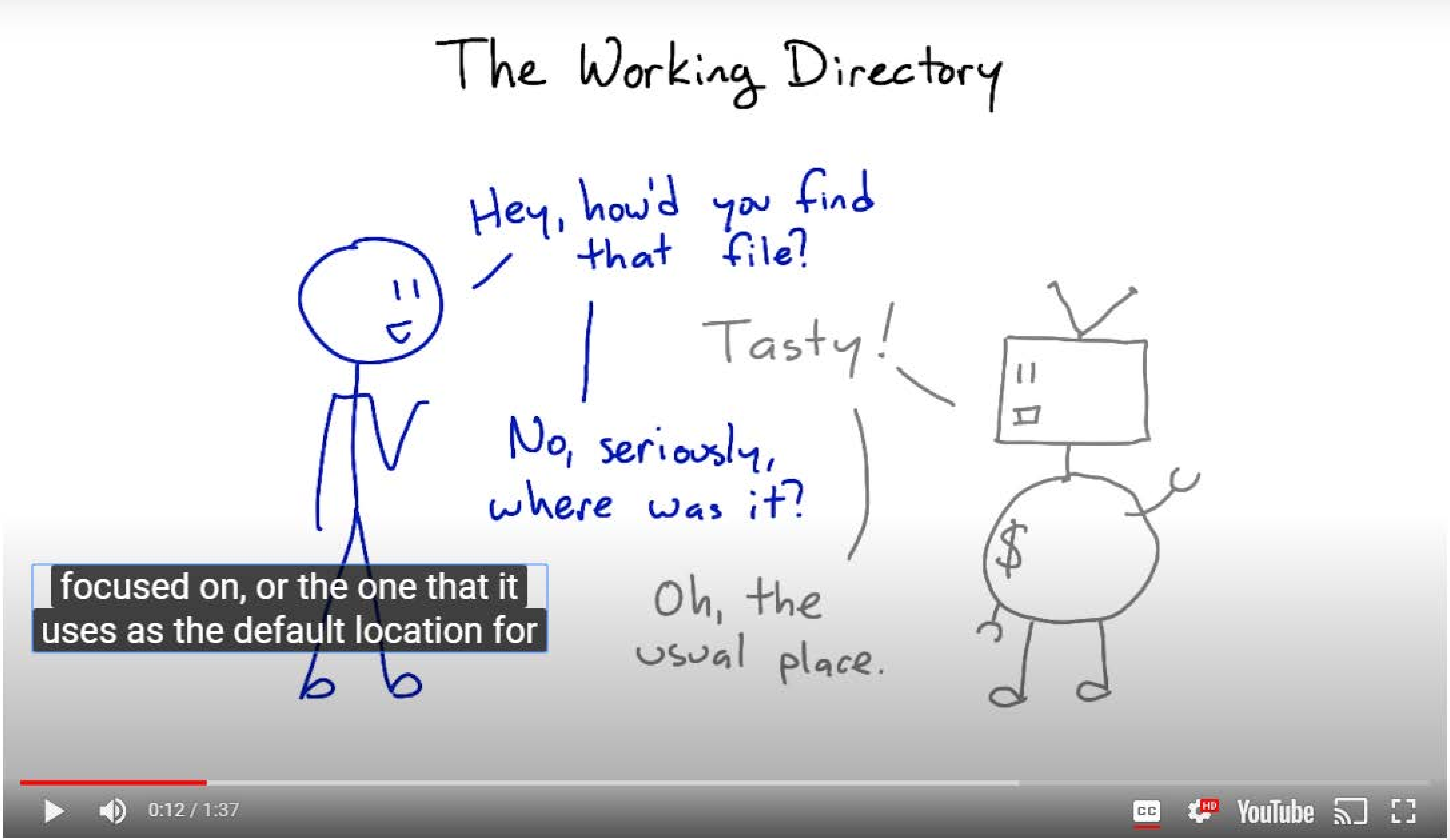


The Filesystem Tree

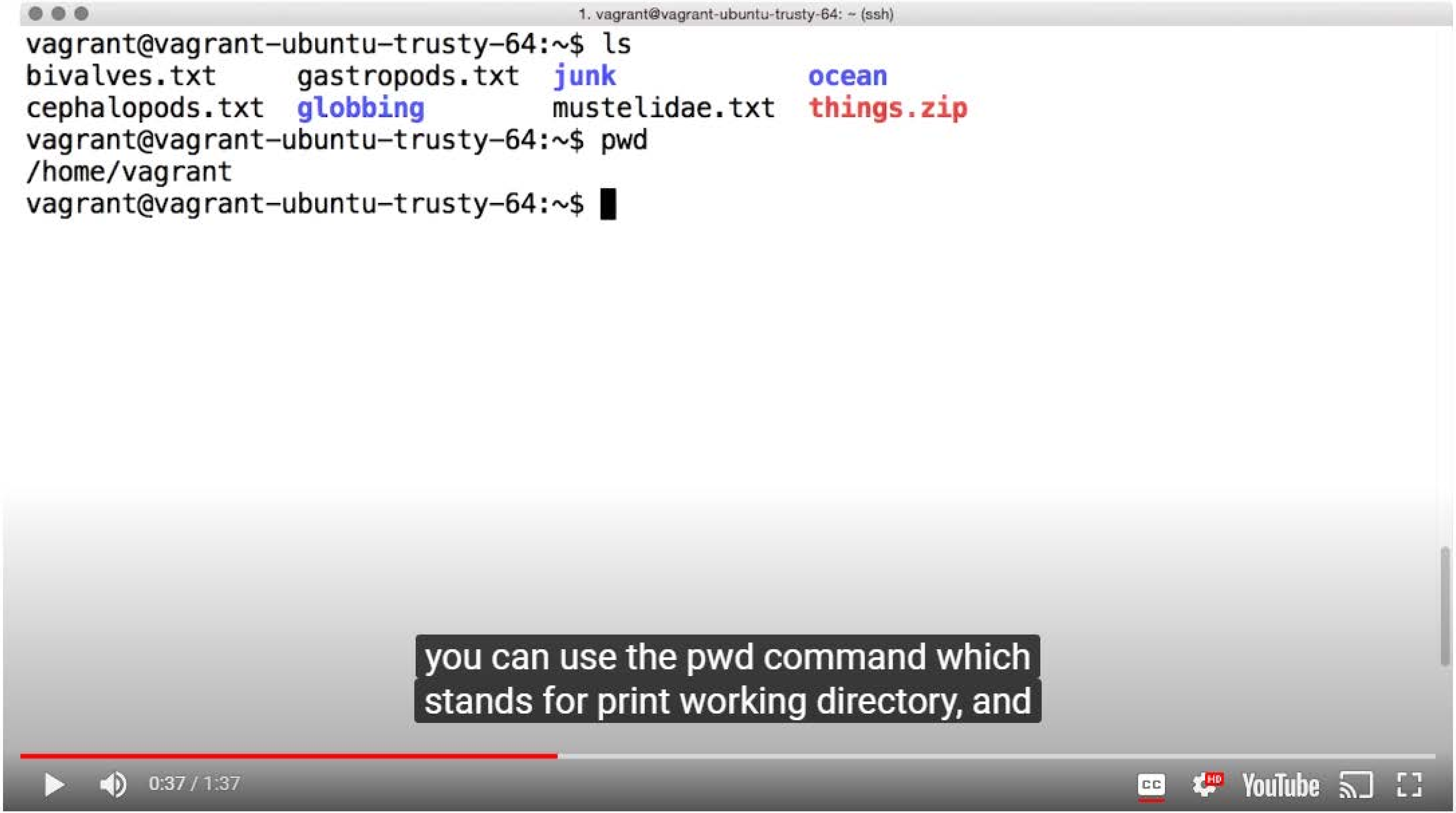
writing fractions like 2/3,
or in various other uses.

The Filesystem Tree



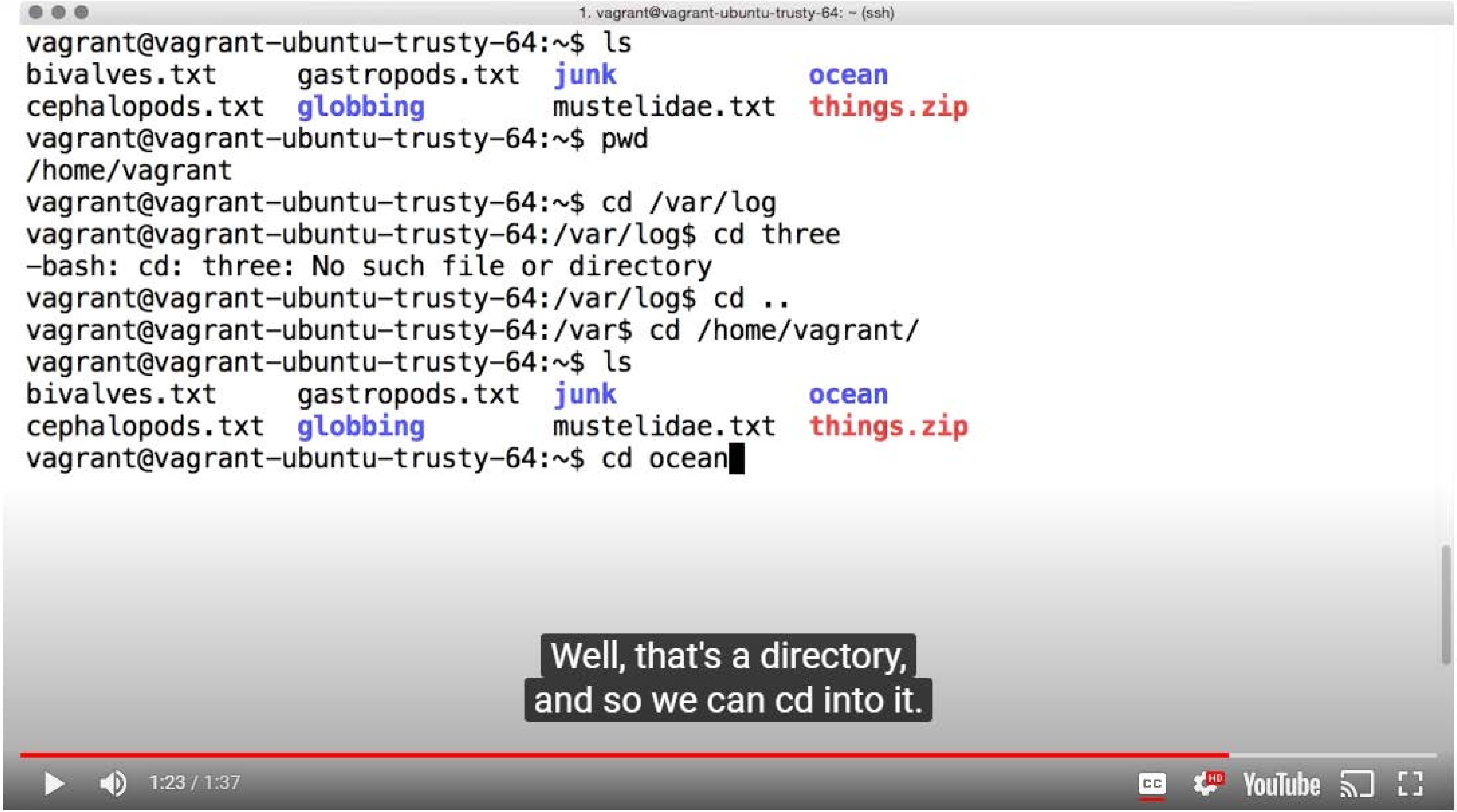


focused on, or the one that it uses as the default location for



```
vagrant@vagrant-ubuntu-trusty-64:~$ ls
bivalves.txt     gastropods.txt  junk            ocean
cephalopods.txt  globbing        mustelidae.txt  things.zip
vagrant@vagrant-ubuntu-trusty-64:~$ pwd
/home/vagrant
vagrant@vagrant-ubuntu-trusty-64:~$ 
```

you can use the pwd command which
stands for print working directory, and

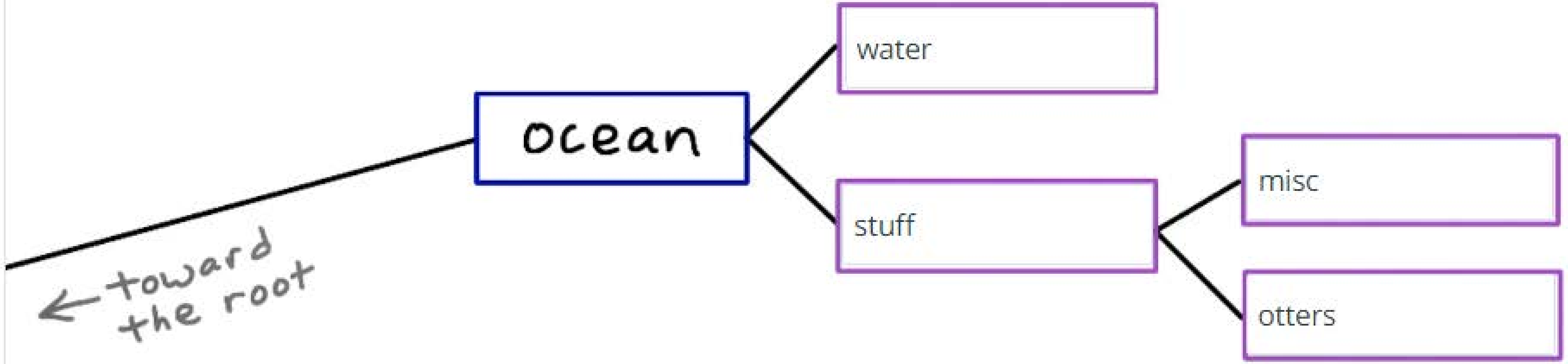


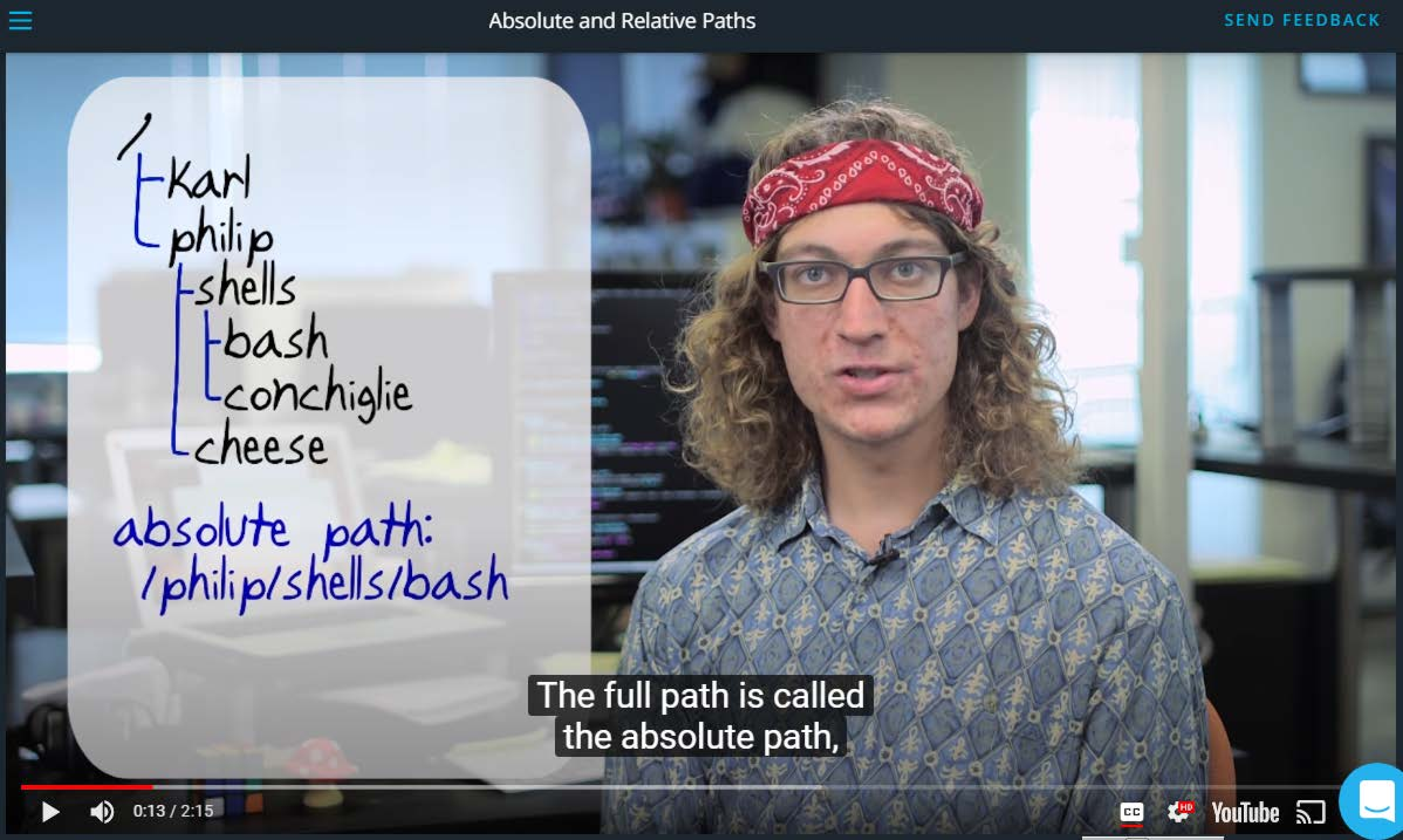
```
vagrant@vagrant-ubuntu-trusty-64:~$ ls
bivalves.txt     gastropods.txt  junk            ocean
cephalopods.txt  globbing        mustelidae.txt  things.zip
vagrant@vagrant-ubuntu-trusty-64:~$ pwd
/home/vagrant
vagrant@vagrant-ubuntu-trusty-64:~$ cd /var/log
vagrant@vagrant-ubuntu-trusty-64:/var/log$ cd three
-bash: cd: three: No such file or directory
vagrant@vagrant-ubuntu-trusty-64:/var/log$ cd ..
vagrant@vagrant-ubuntu-trusty-64:/var$ cd /home/vagrant/
vagrant@vagrant-ubuntu-trusty-64:~$ ls
bivalves.txt     gastropods.txt  junk            ocean
cephalopods.txt  globbing        mustelidae.txt  things.zip
vagrant@vagrant-ubuntu-trusty-64:~$ cd ocean
```

Well, that's a directory,
and so we can cd into it.

The Working Directory

Using cd and ls, map out the subdirectories and files within the ocean directory.

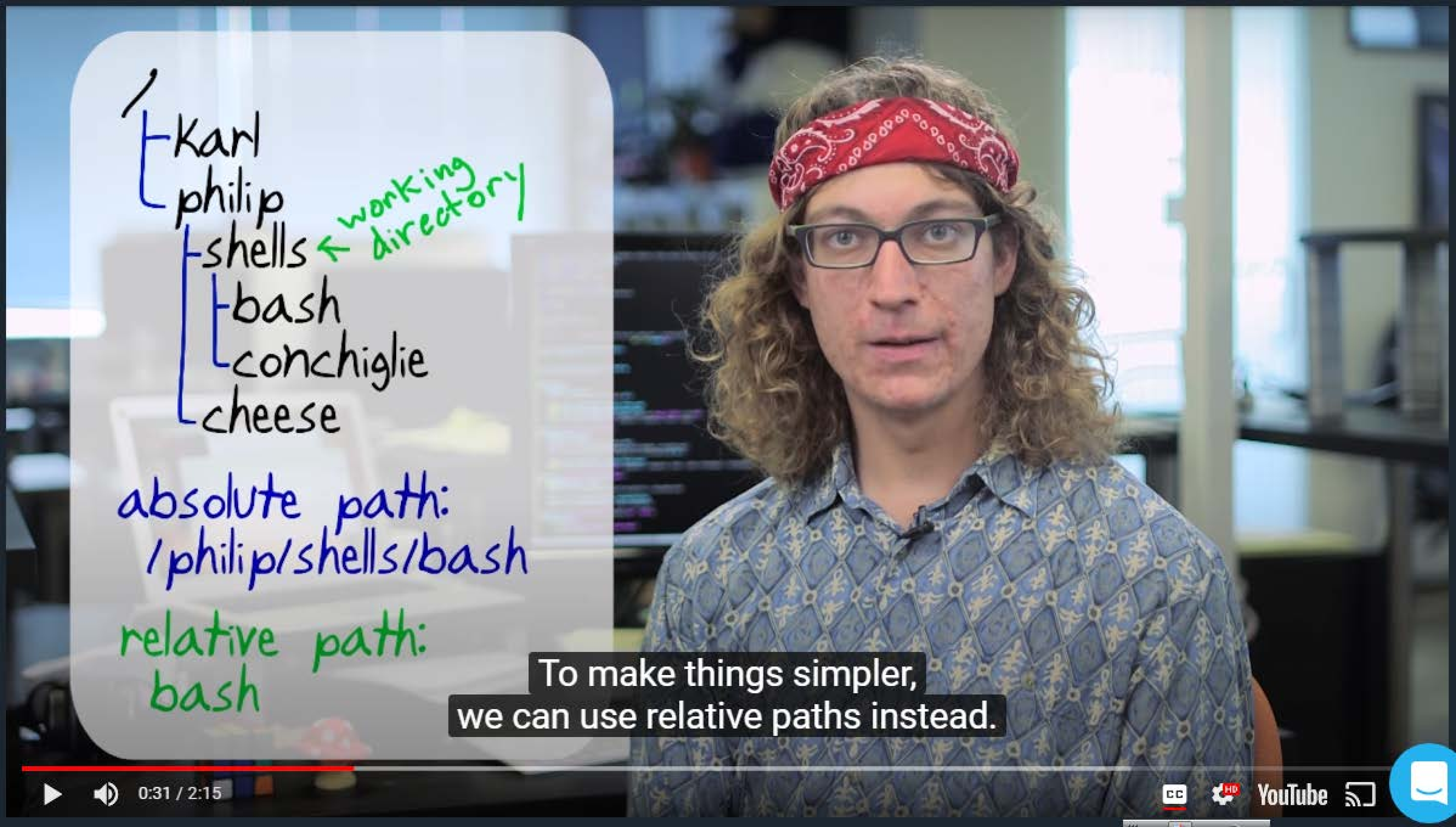




```
/
├ karl
└ philip
  ├ shells
  │ ├ bash
  │ └ conchiglie
  └ cheese
```

absolute path:
/philip/shells/bash

The full path is called
the absolute path,

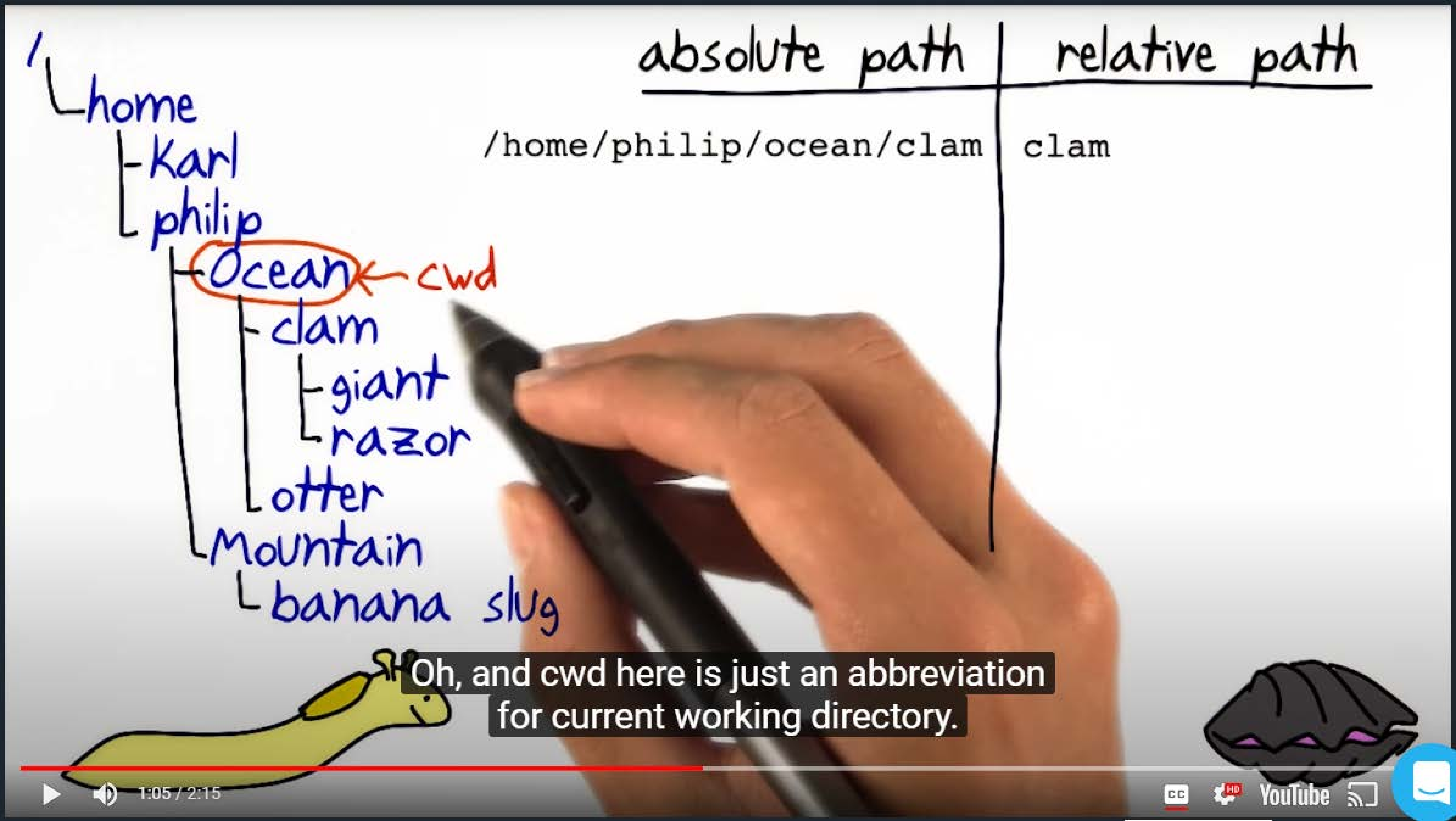


```
/
├karl
└philip
  ├shells ← working directory
  │ ├bash
  │ └conchiglie
  └cheese
```

absolute path:
/philip/shells/bash

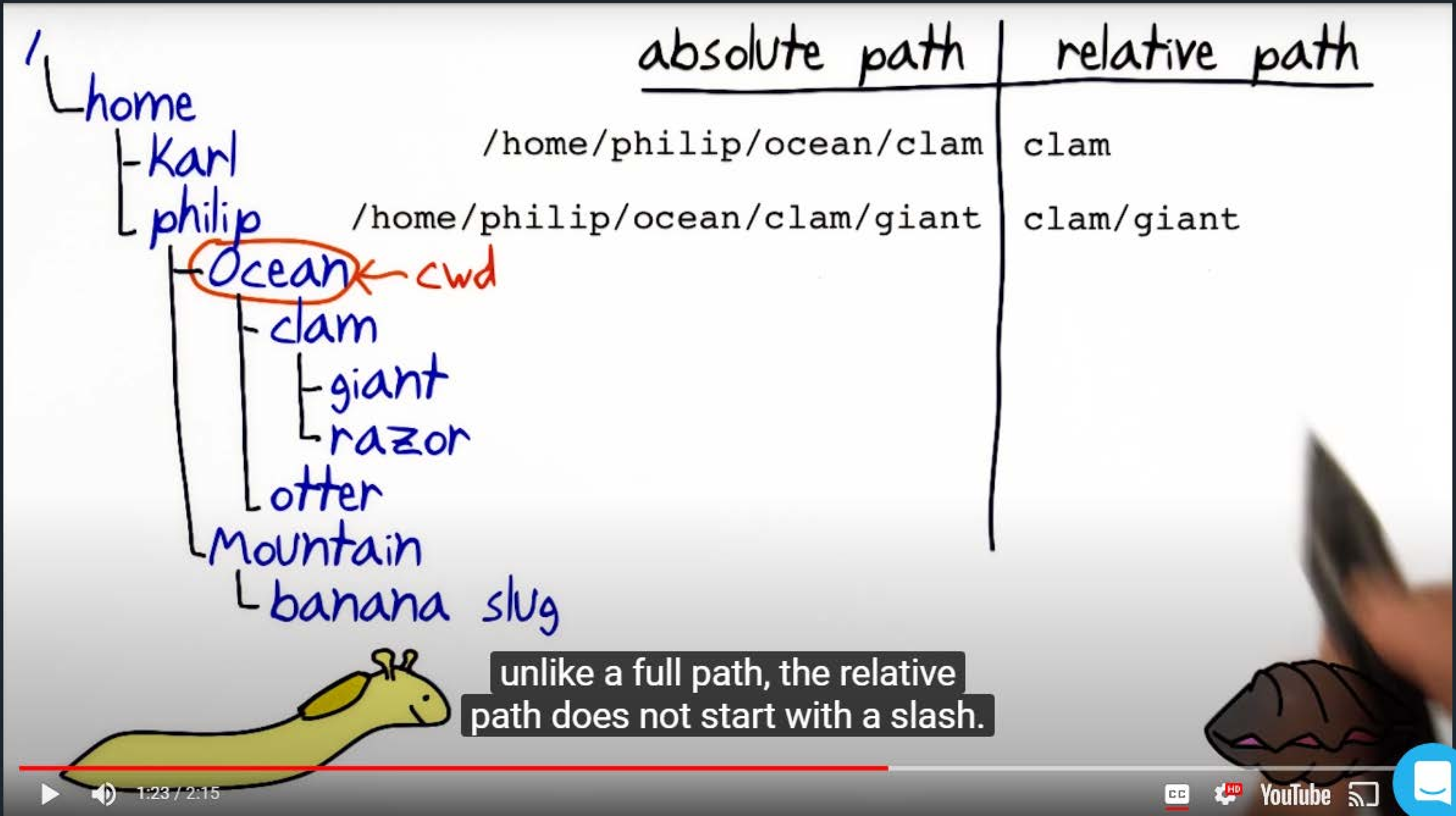
relative path:
bash

To make things simpler,
we can use relative paths instead.



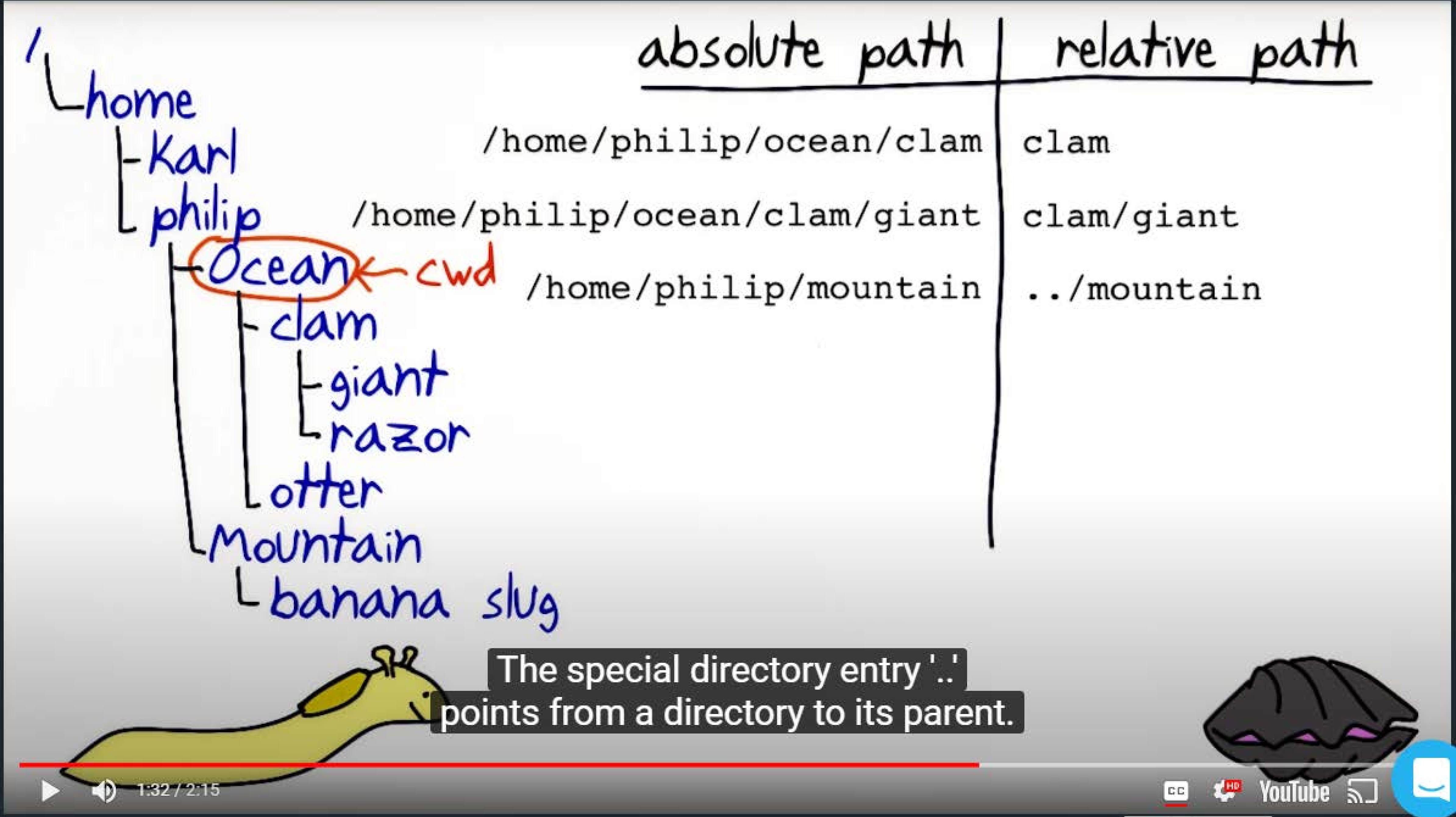
absolute path	relative path
/home/philip/ocean/clam	clam

Oh, and cwd here is just an abbreviation for current working directory.



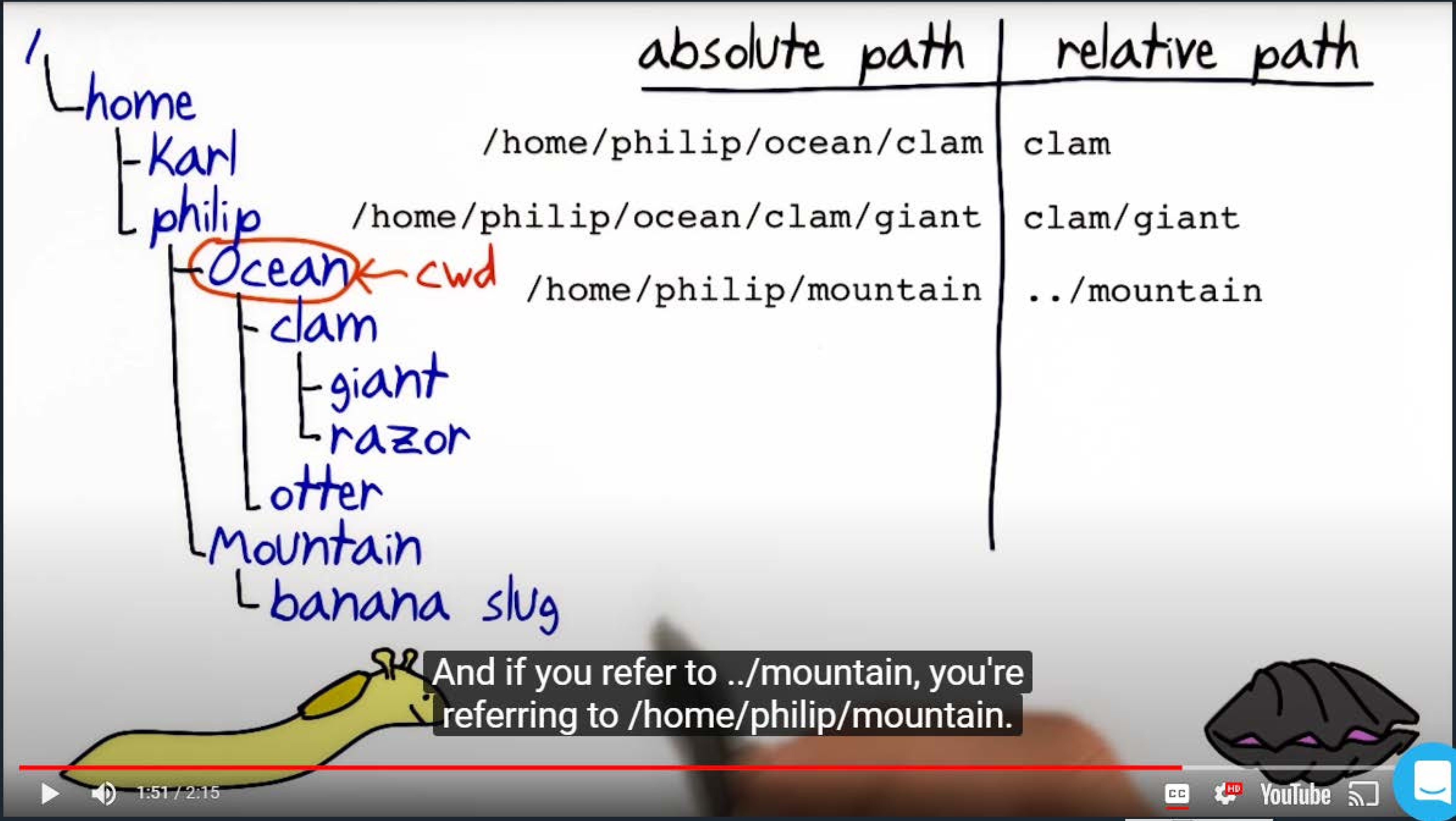
absolute path	relative path
/home/philip/ocean/clam	clam
/home/philip/ocean/clam/giant	clam/giant

unlike a full path, the relative
path does not start with a slash.



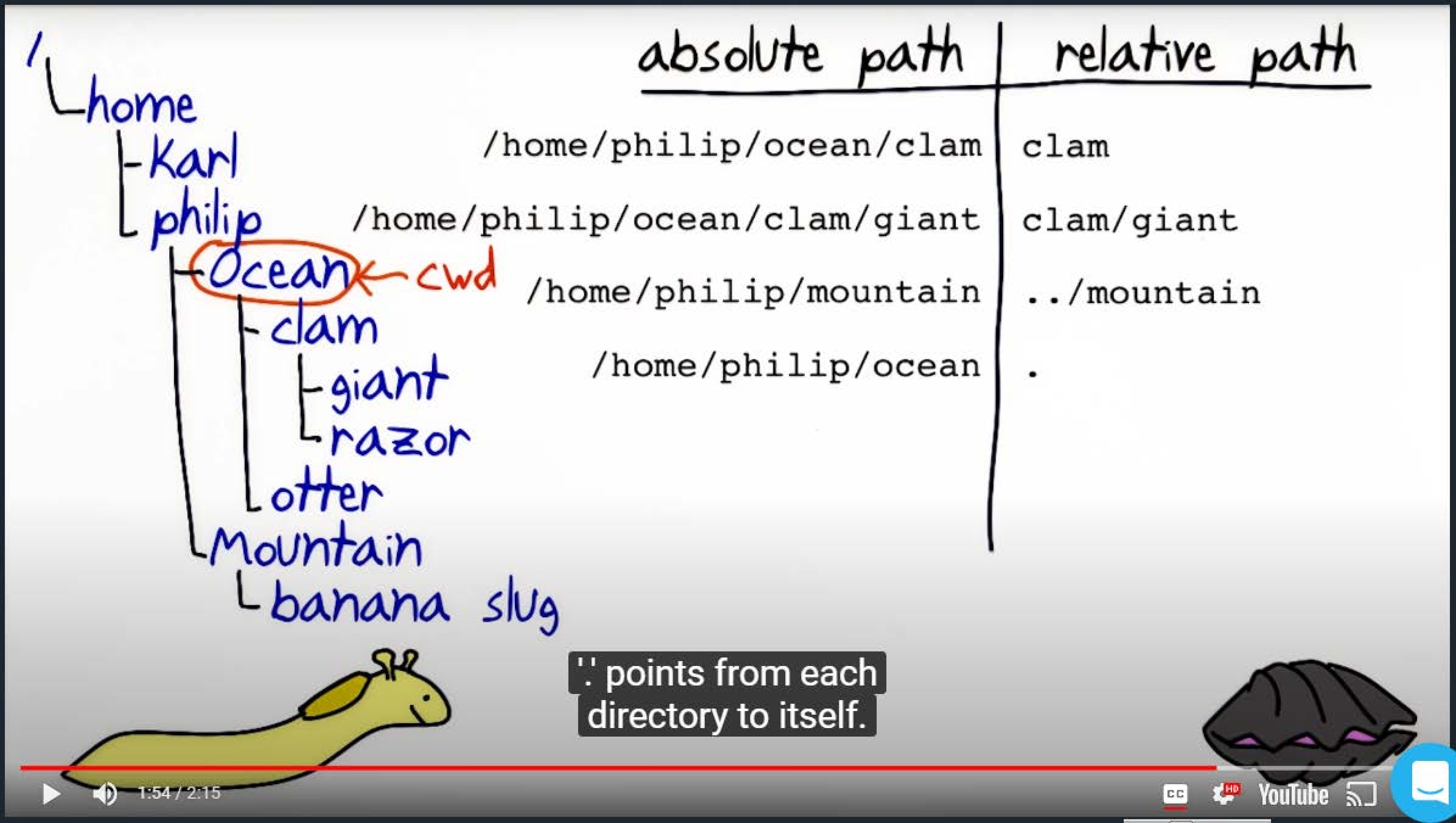
absolute path	relative path
/home/philip/ocean/clam	clam
/home/philip/ocean/clam/giant	clam/giant
/home/philip/mountain	../mountain

The special directory entry '..'
points from a directory to its parent.



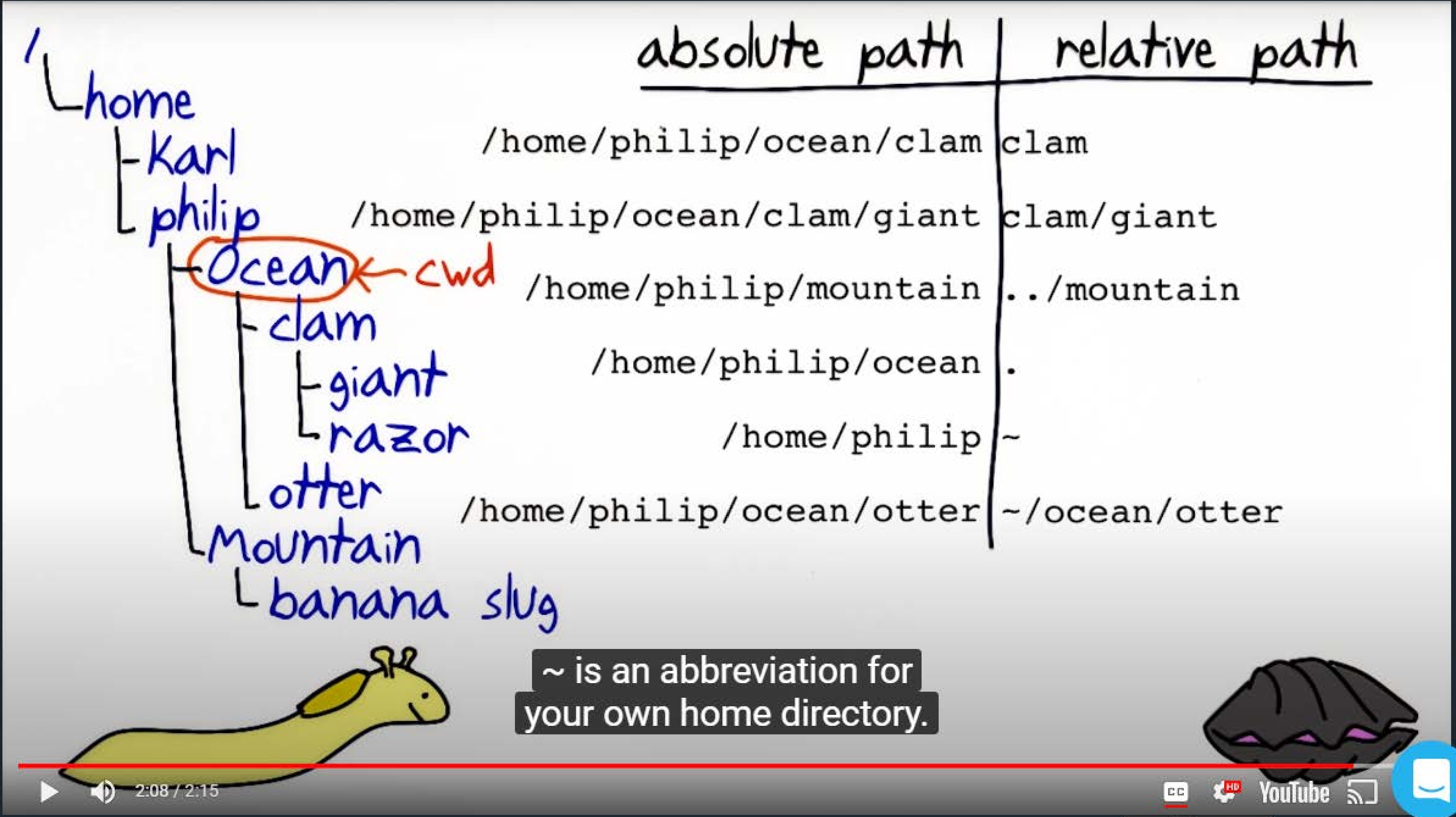
absolute path	relative path
/home/philip/ocean/clam	clam
/home/philip/ocean/clam/giant	clam/giant
/home/philip/mountain	../mountain

And if you refer to ../mountain, you're referring to /home/philip/mountain.



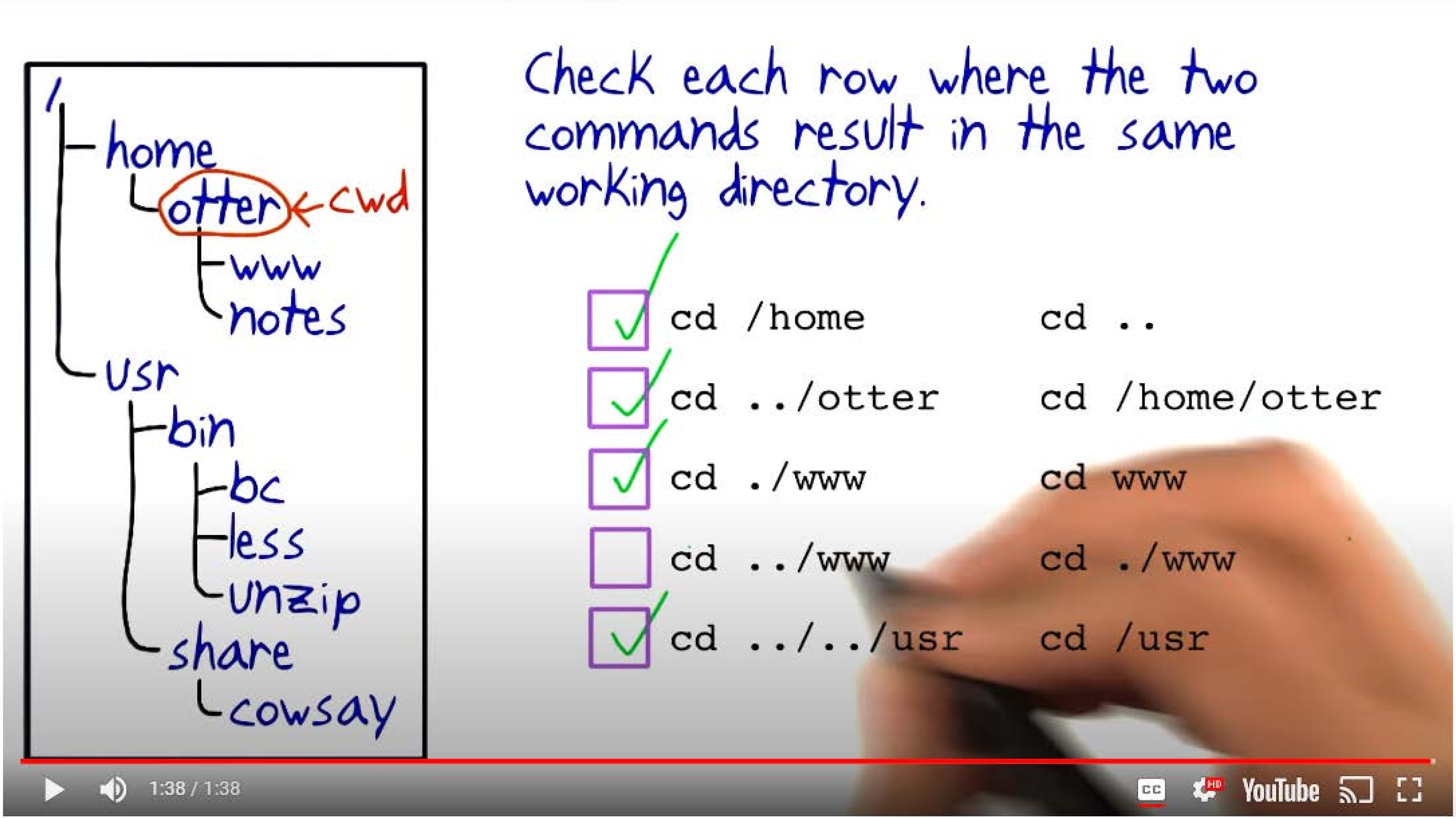
absolute path	relative path
/home/philip/ocean/clam	clam
/home/philip/ocean/clam/giant	clam/giant
/home/philip/mountain	../mountain
/home/philip/ocean	.

'.' points from each
directory to itself.



absolute path	relative path
/home/philip/ocean/clam	clam
/home/philip/ocean/clam/giant	clam/giant
/home/philip/mountain	../mountain
/home/philip/ocean	.
/home/philip	~
/home/philip/ocean/otter	~/ocean/otter

~ is an abbreviation for
your own home directory.



Check each row where the two commands result in the same working directory.

☑ `cd /home` `cd ..`

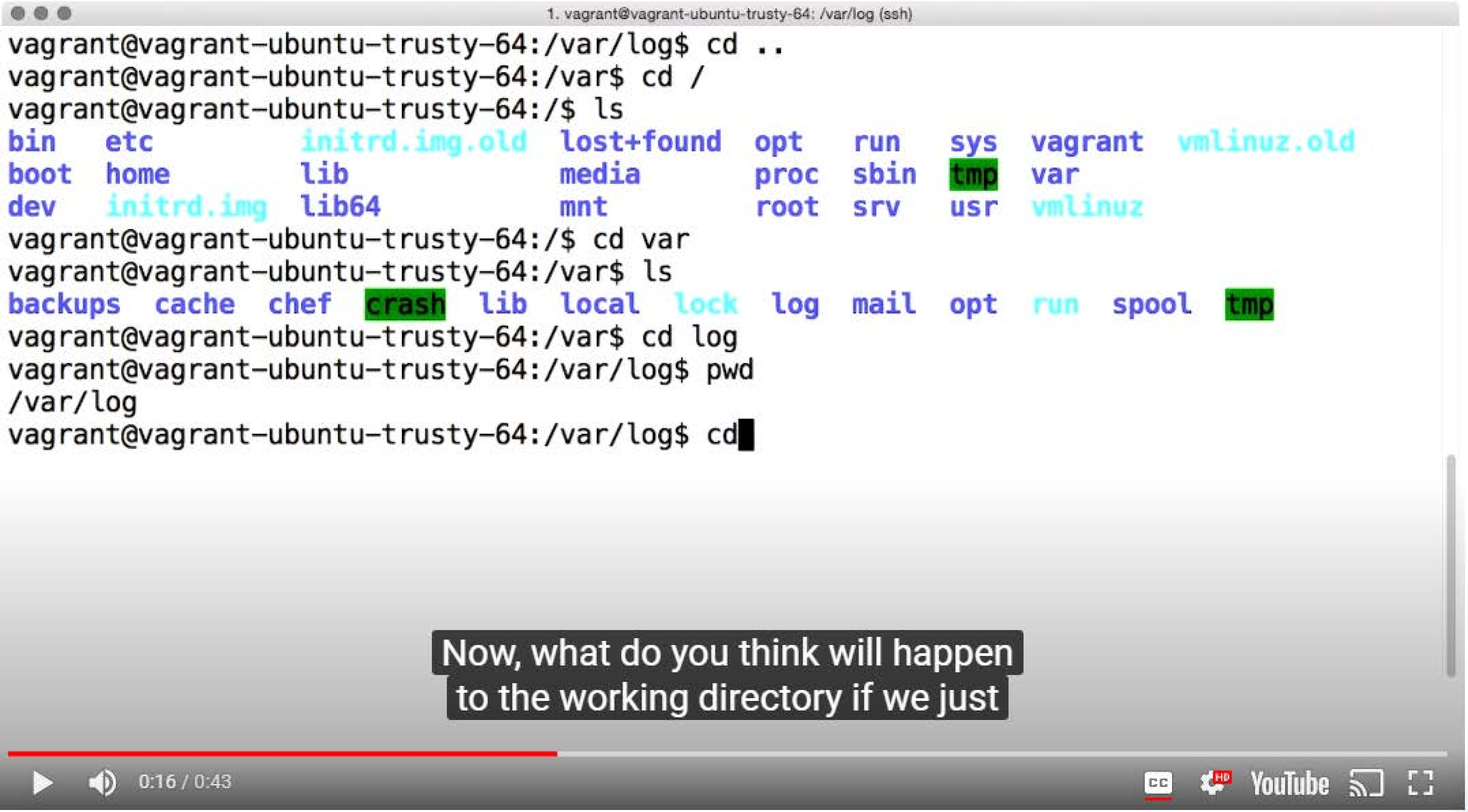
☑ `cd ../otter` `cd /home/otter`

☑ `cd ./www` `cd www`

□ `cd ../www` `cd ./www`

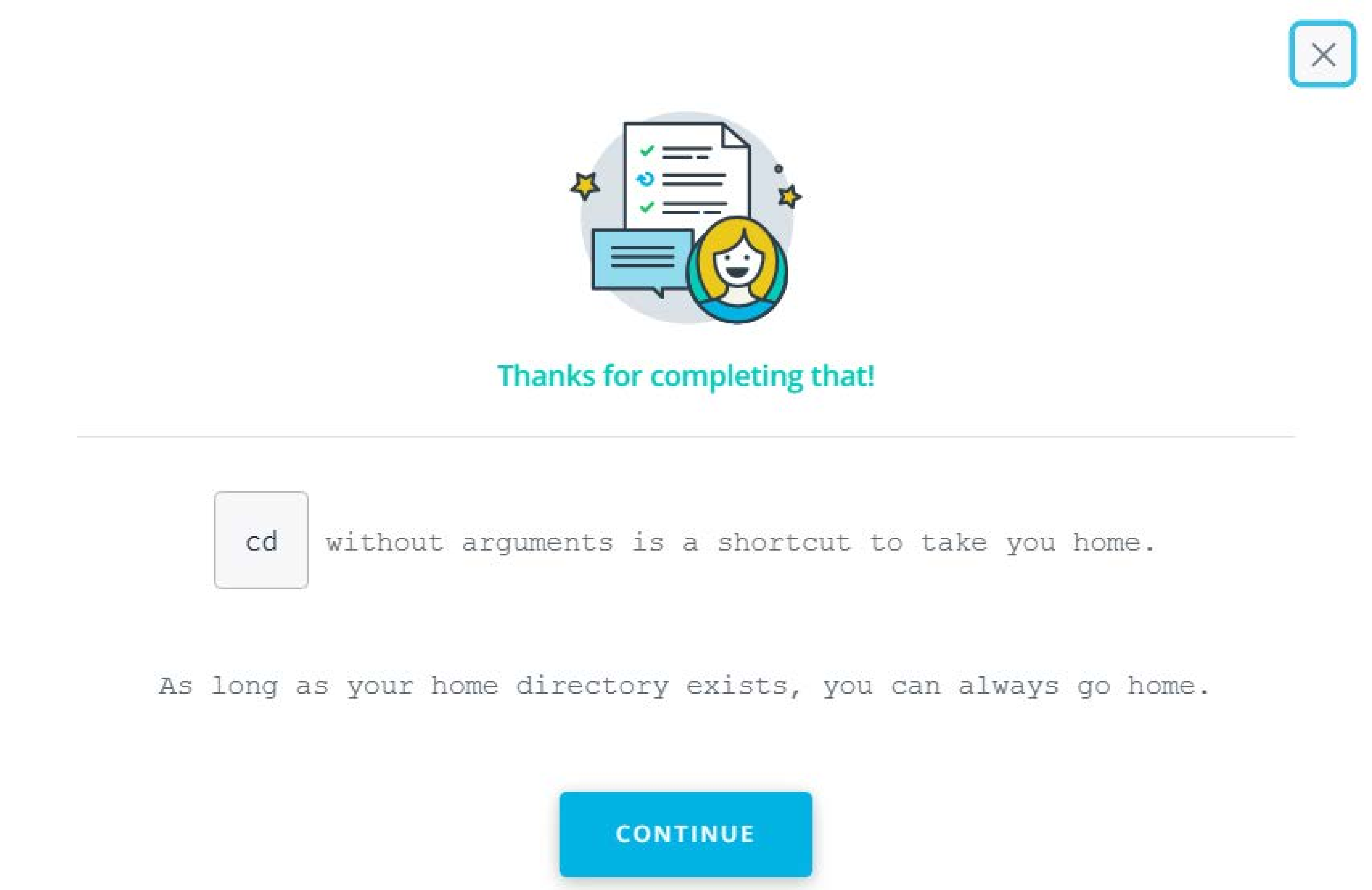
☑ `cd ../../usr` `cd /usr`

cd without args



```
vagrant@vagrant-ubuntu-trusty-64:/var/log$ cd ..
vagrant@vagrant-ubuntu-trusty-64:/var$ cd /
vagrant@vagrant-ubuntu-trusty-64:/$ ls
bin   etc         initrd.img.old  lost+found  opt   run   sys  vagrant  vmlinuz.old
boot  home        lib             media       proc  sbin  tmp  var
dev   initrd.img  lib64           mnt         root  srv   usr  vmlinuz
vagrant@vagrant-ubuntu-trusty-64:/$ cd var
vagrant@vagrant-ubuntu-trusty-64:/var$ ls
backups  cache  chef  crash  lib  local  lock  log  mail  opt  run  spool  tmp
vagrant@vagrant-ubuntu-trusty-64:/var$ cd log
vagrant@vagrant-ubuntu-trusty-64:/var/log$ pwd
/var/log
vagrant@vagrant-ubuntu-trusty-64:/var/log$ cd
```

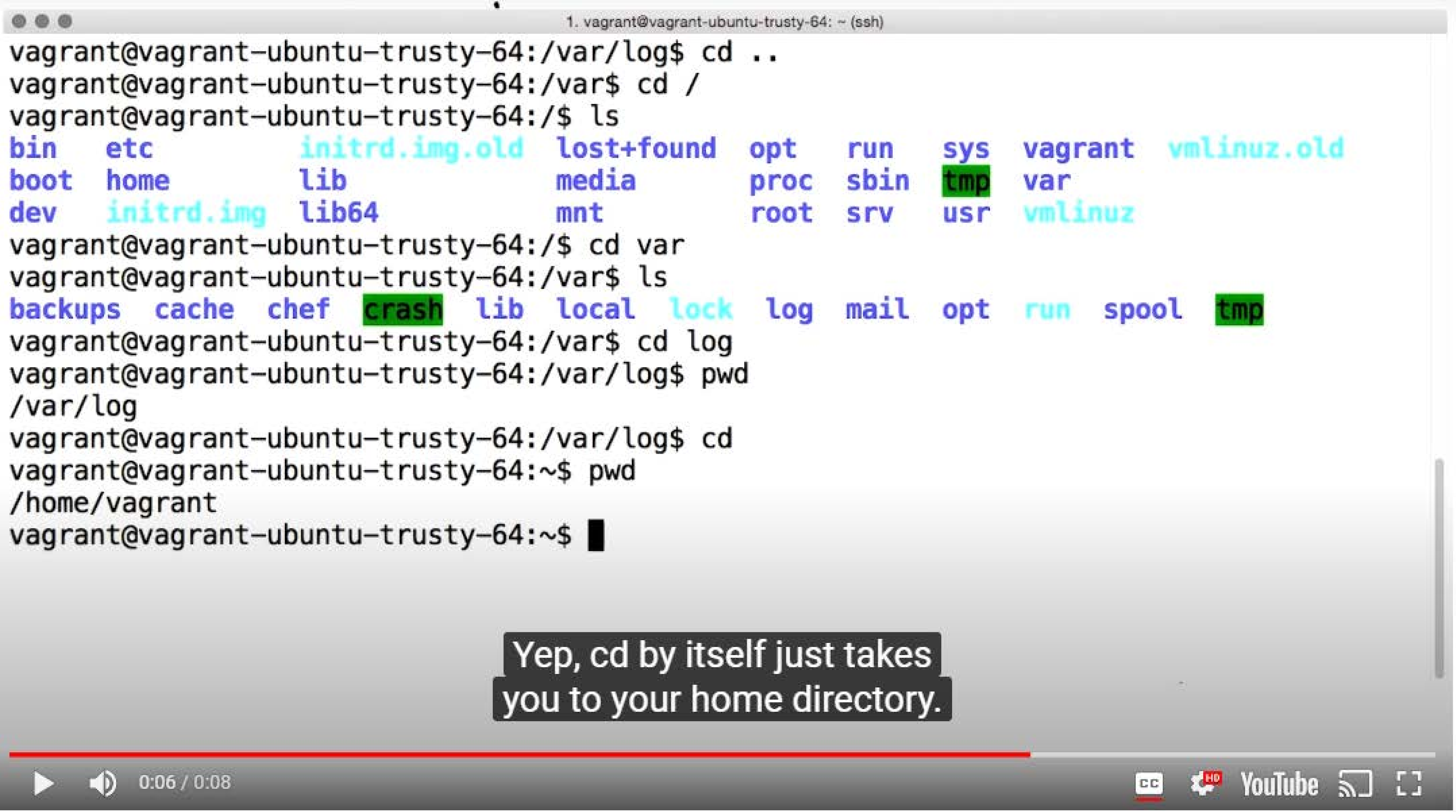
Now, what do you think will happen
to the working directory if we just



Thanks for completing that!

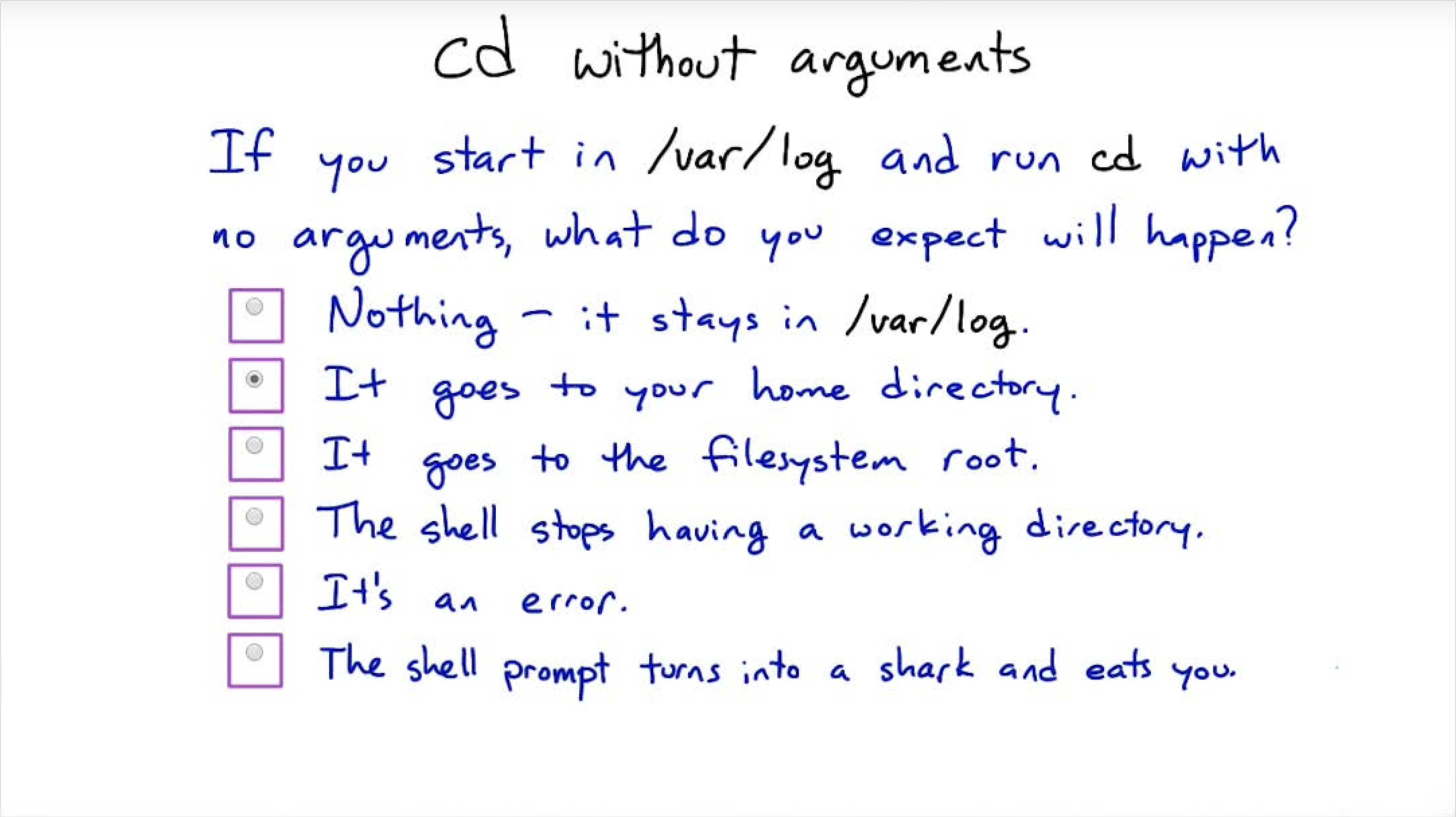
`cd` without arguments is a shortcut to take you home.

As long as your home directory exists, you can always go home.



```
vagrant@vagrant-ubuntu-trusty-64:/var/log$ cd ..
vagrant@vagrant-ubuntu-trusty-64:/var$ cd /
vagrant@vagrant-ubuntu-trusty-64:/$ ls
bin   etc         initrd.img.old  lost+found  opt   run   sys   vagrant  vmlinuz.old
boot  home        lib             media       proc  sbin  tmp   var
dev   initrd.img  lib64           mnt         root  srv   usr   vmlinuz
vagrant@vagrant-ubuntu-trusty-64:/$ cd var
vagrant@vagrant-ubuntu-trusty-64:/var$ ls
backups  cache  chef  crash  lib  local  lock  log  mail  opt  run  spool  tmp
vagrant@vagrant-ubuntu-trusty-64:/var$ cd log
vagrant@vagrant-ubuntu-trusty-64:/var/log$ pwd
/var/log
vagrant@vagrant-ubuntu-trusty-64:/var/log$ cd
vagrant@vagrant-ubuntu-trusty-64:~$ pwd
/home/vagrant
vagrant@vagrant-ubuntu-trusty-64:~$
```

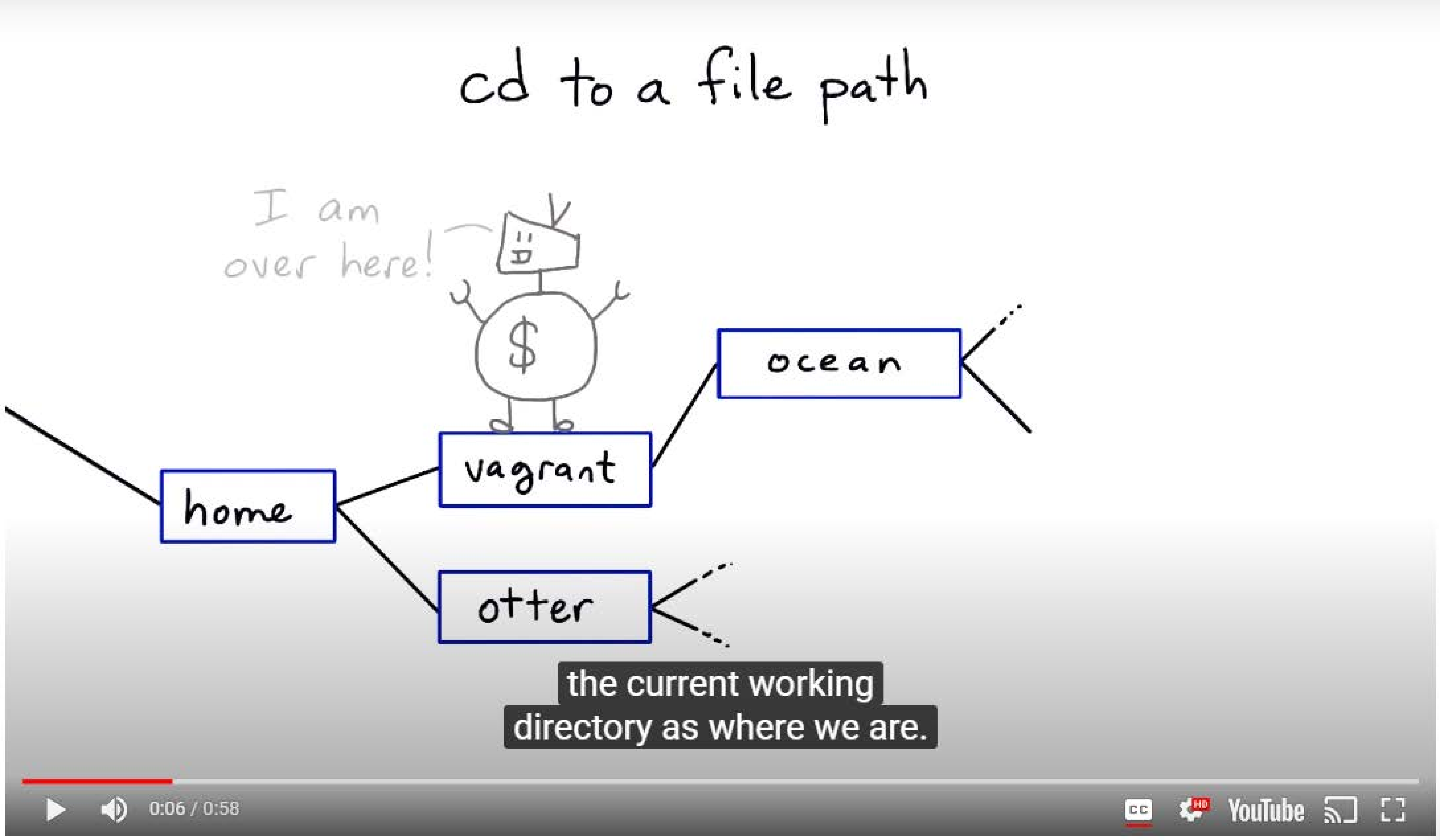
Yep, cd by itself just takes
you to your home directory.



cd without arguments

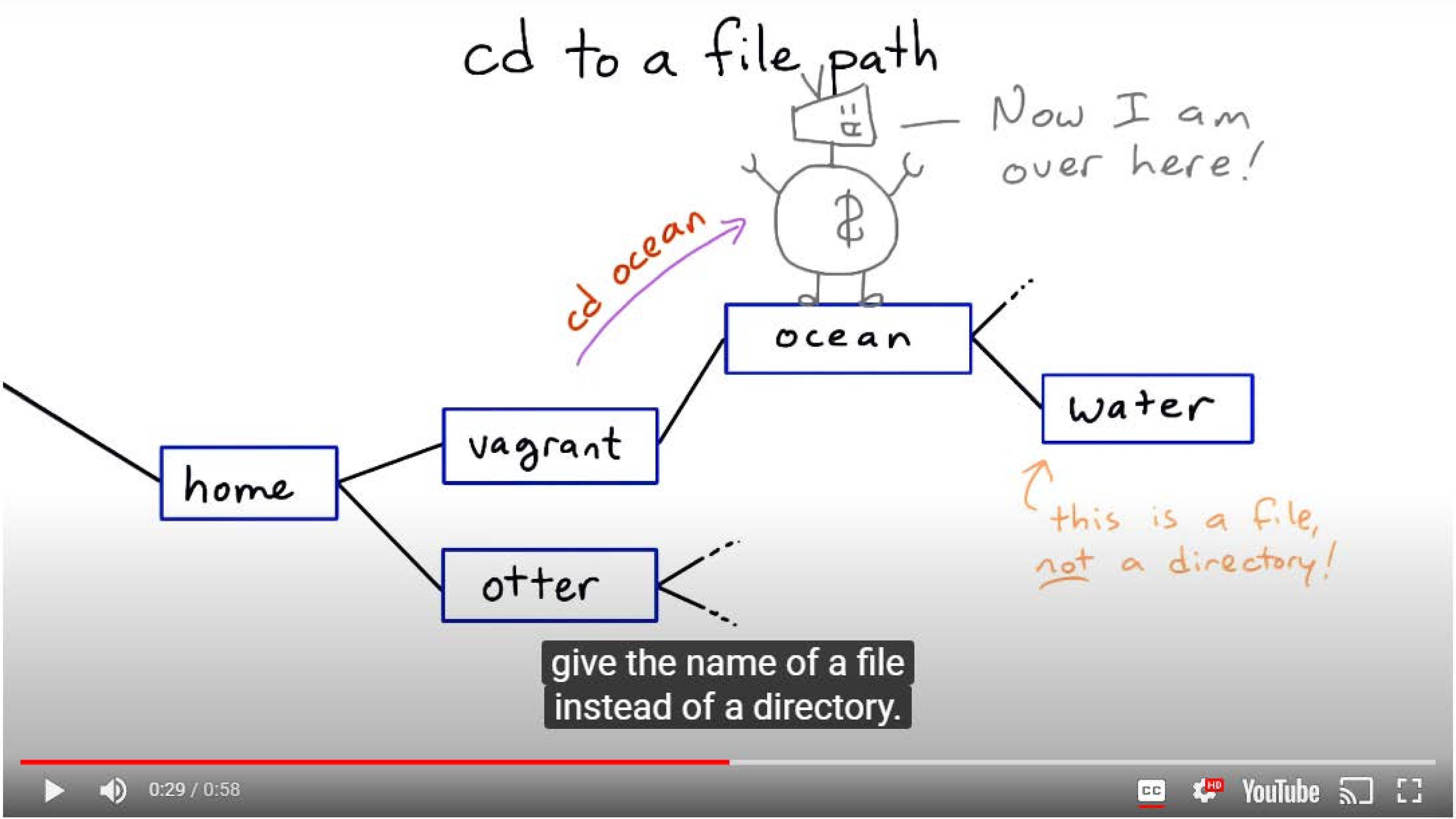
If you start in /var/log and run cd with no arguments, what do you expect will happen?

- ○ Nothing – it stays in /var/log.
- ● It goes to your home directory.
- ○ It goes to the filesystem root.
- ○ The shell stops having a working directory.
- ○ It's an error.
- ○ The shell prompt turns into a shark and eats you.



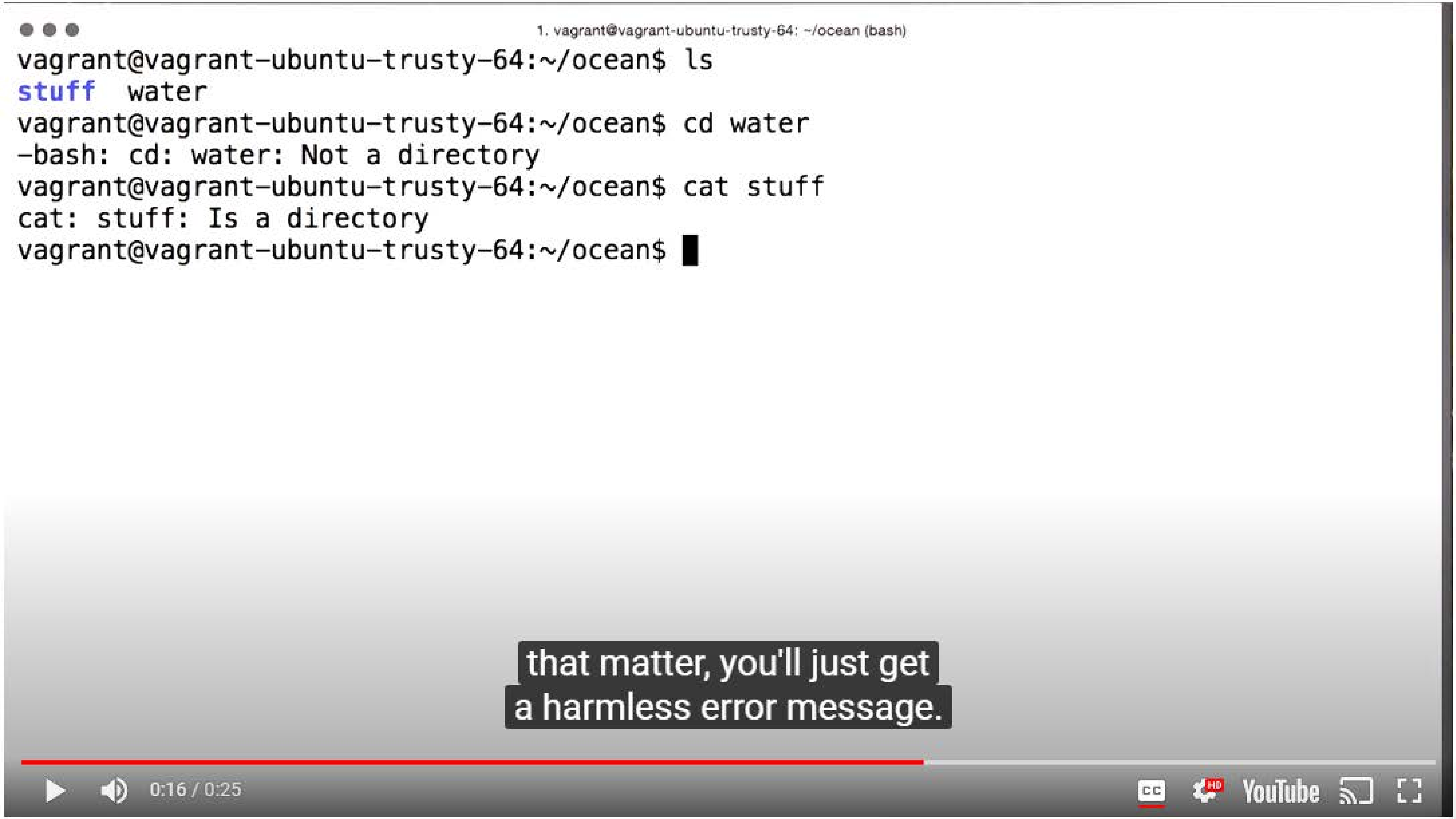
cd to a file path

the current working directory as where we are.



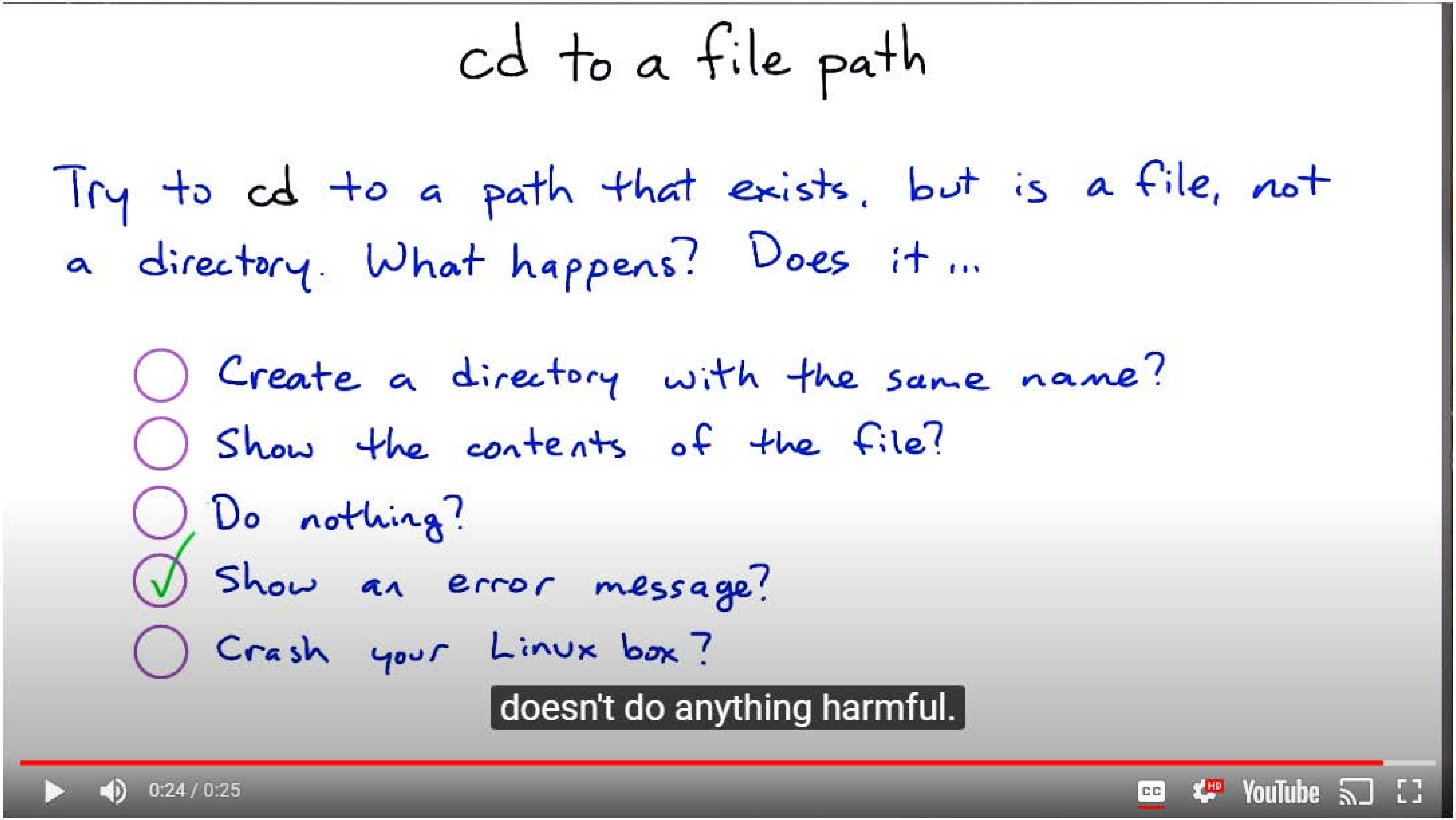
cd to a file path

give the name of a file instead of a directory.



```
vagrant@vagrant-ubuntu-trusty-64:~/ocean$ ls
stuff  water
vagrant@vagrant-ubuntu-trusty-64:~/ocean$ cd water
-bash: cd: water: Not a directory
vagrant@vagrant-ubuntu-trusty-64:~/ocean$ cat stuff
cat: stuff: Is a directory
vagrant@vagrant-ubuntu-trusty-64:~/ocean$ 
```

that matter, you'll just get
a harmless error message.



cd to a file path

Try to cd to a path that exists, but is a file, not a directory. What happens? Does it ...

○ Create a directory with the same name?

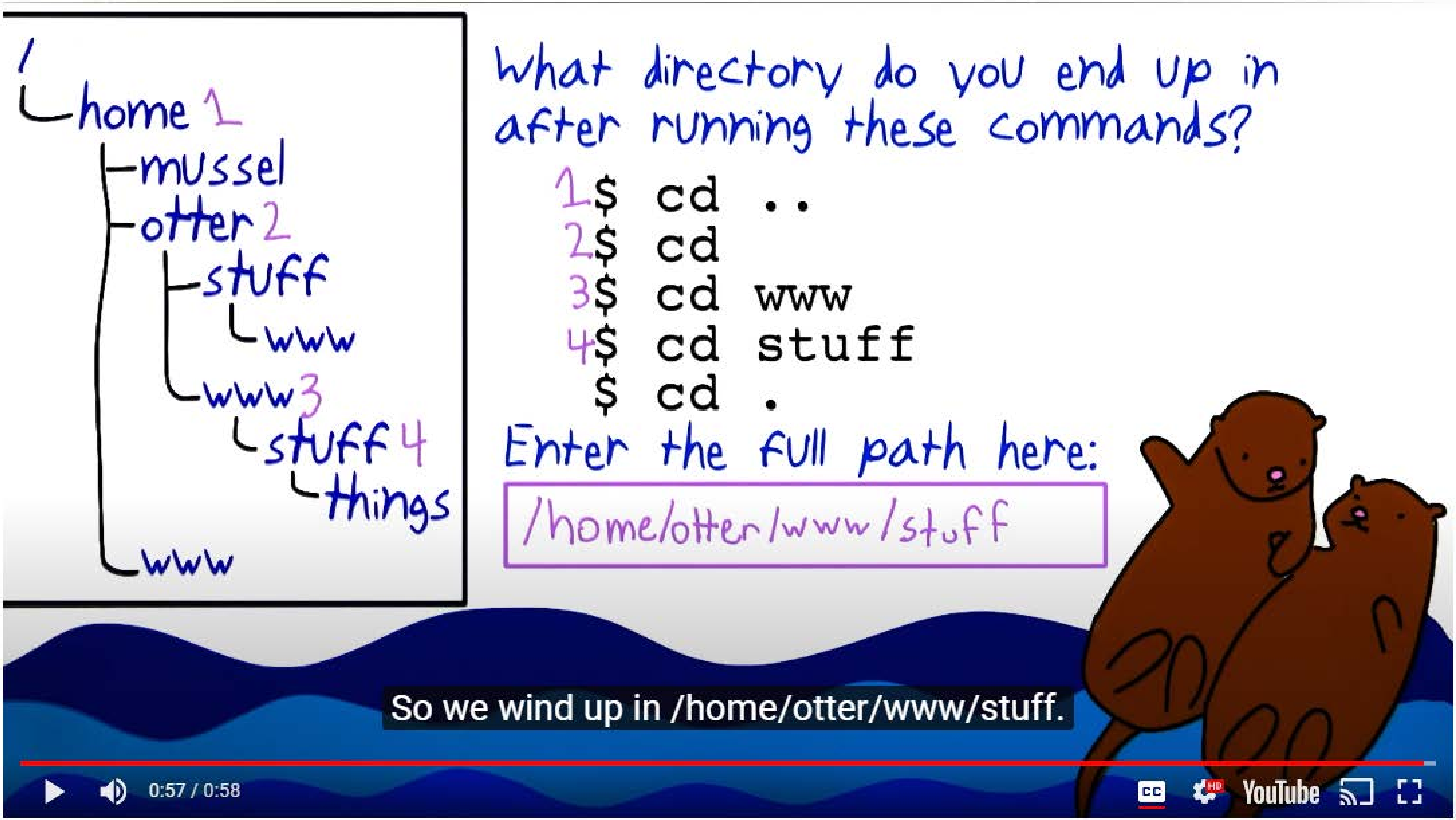
○ Show the contents of the file?

○ Do nothing?

○ ✓ Show an error message?

○ Crash your Linux box?

doesn't do anything harmful.



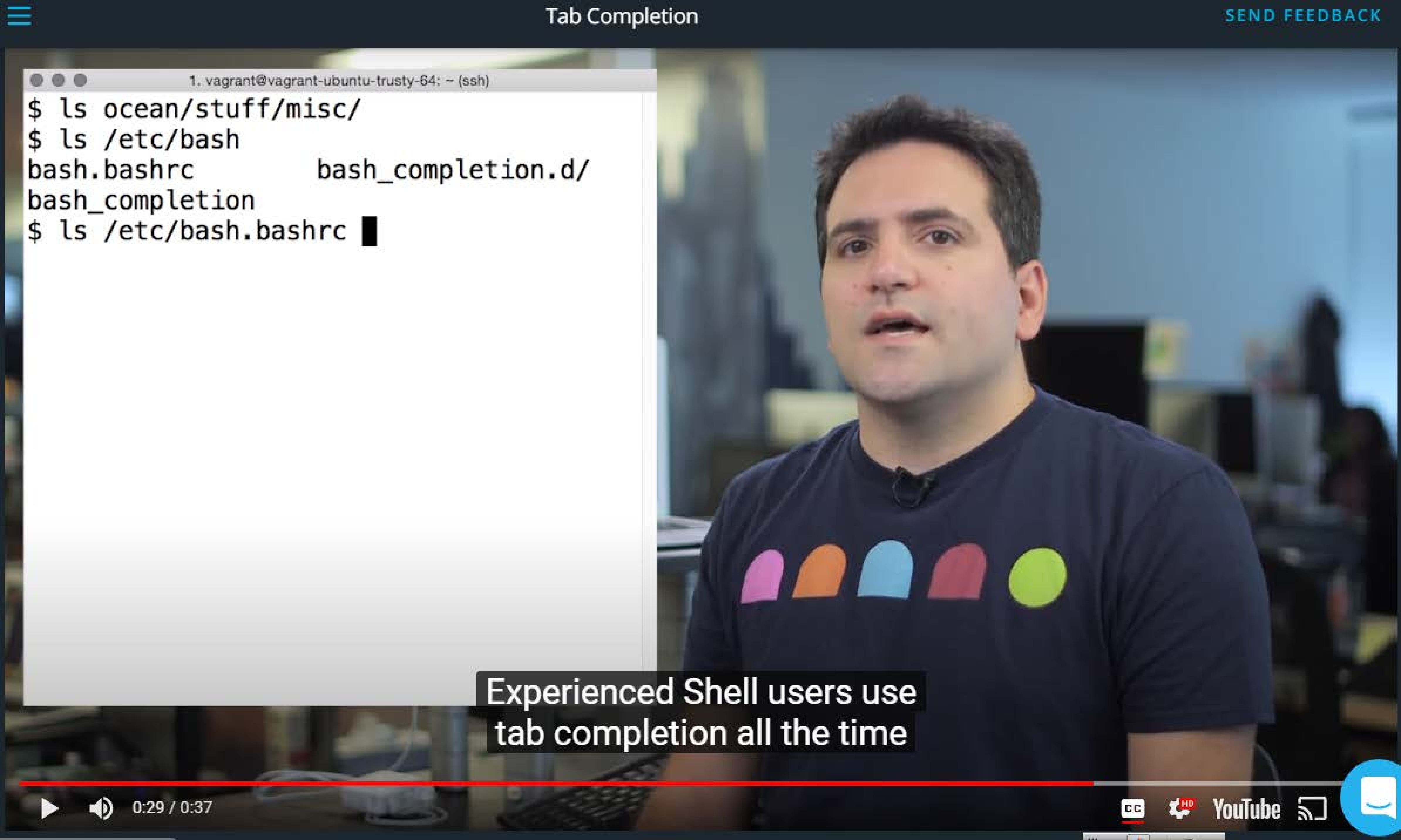
What directory do you end up in after running these commands?

```
1$ cd ..
2$ cd
3$ cd www
4$ cd stuff
 $ cd .
```

Enter the full path here:

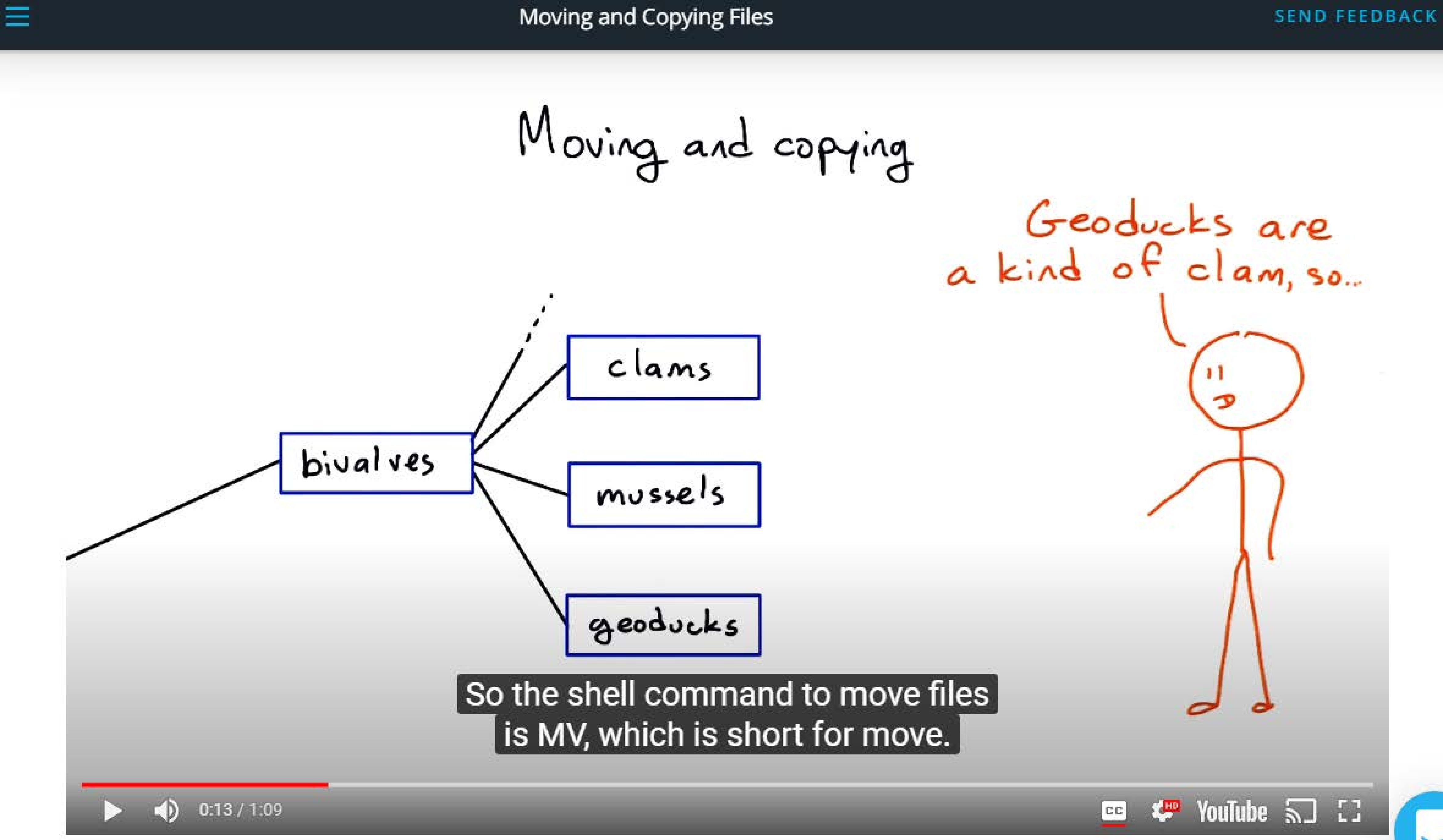
/home/otter/www/stuff

So we wind up in /home/otter/www/stuff.



```
$ ls ocean/stuff/misc/
$ ls /etc/bash
bash.bashrc          bash_completion.d/
bash_completion
$ ls /etc/bash.bashrc
```

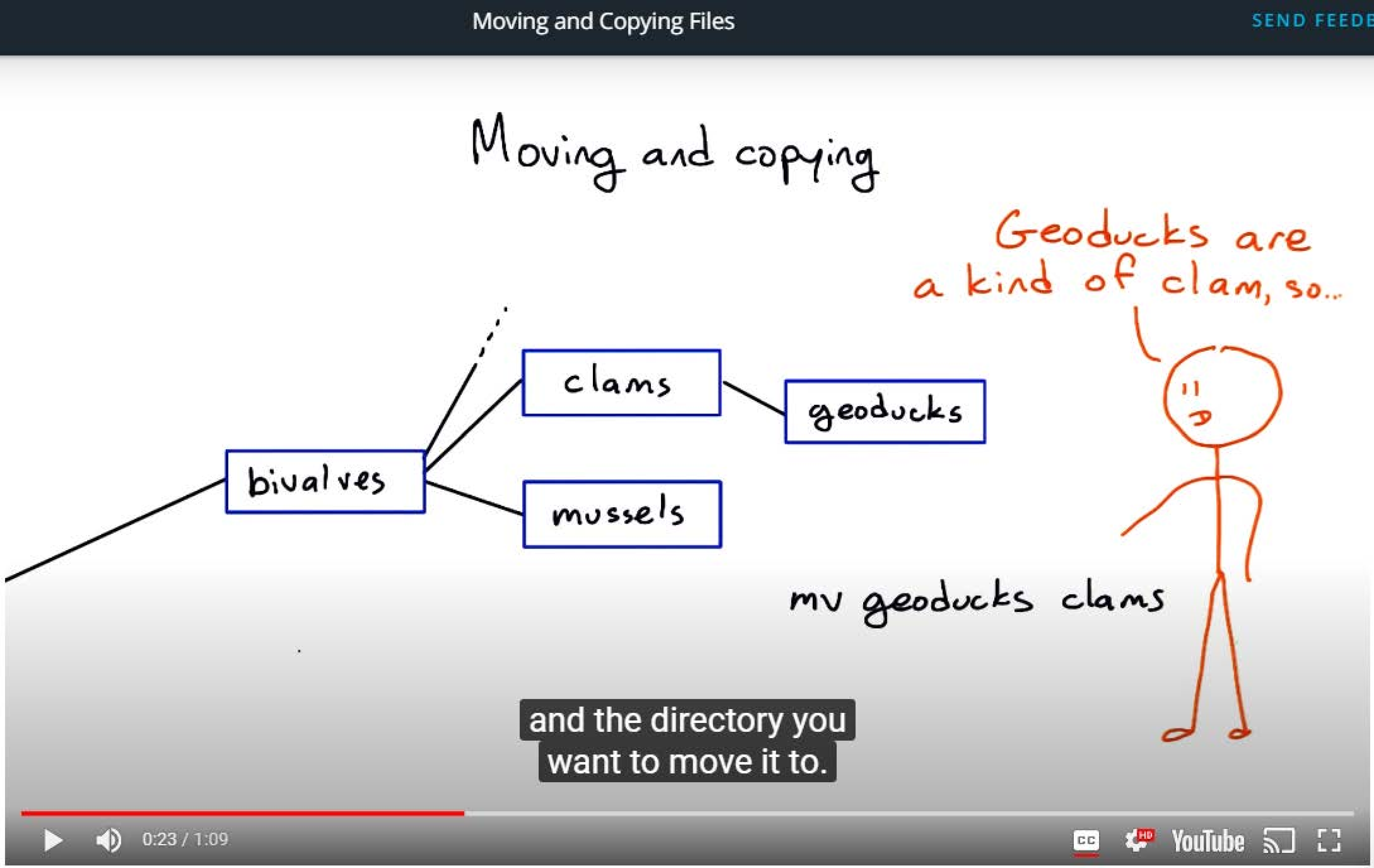
Experienced Shell users use
tab completion all the time



Moving and Copying Files

Moving and copying

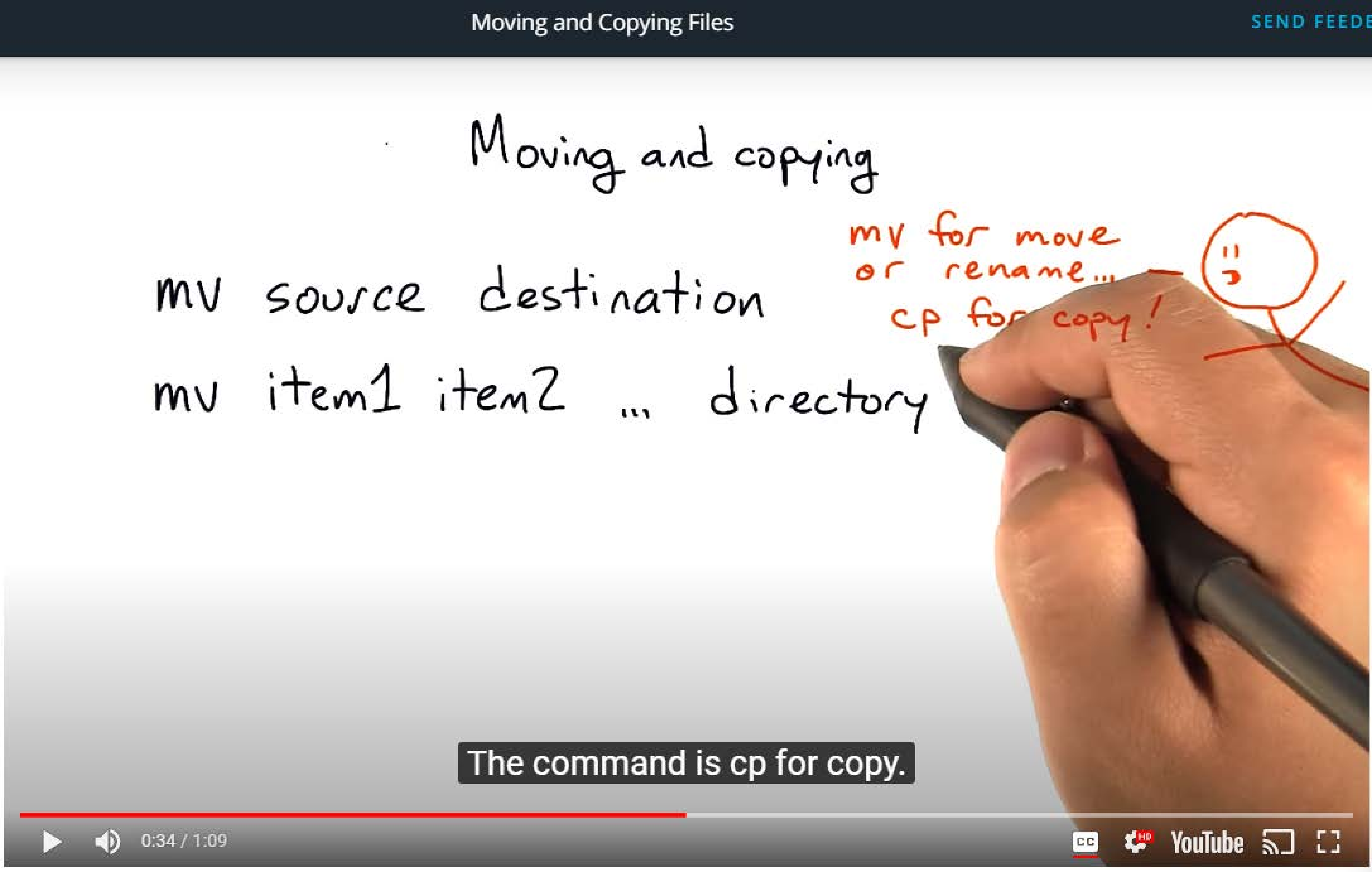
So the shell command to move files is MV, which is short for move.



Moving and copying

```
mv geoducks clams
```

and the directory you want to move it to.



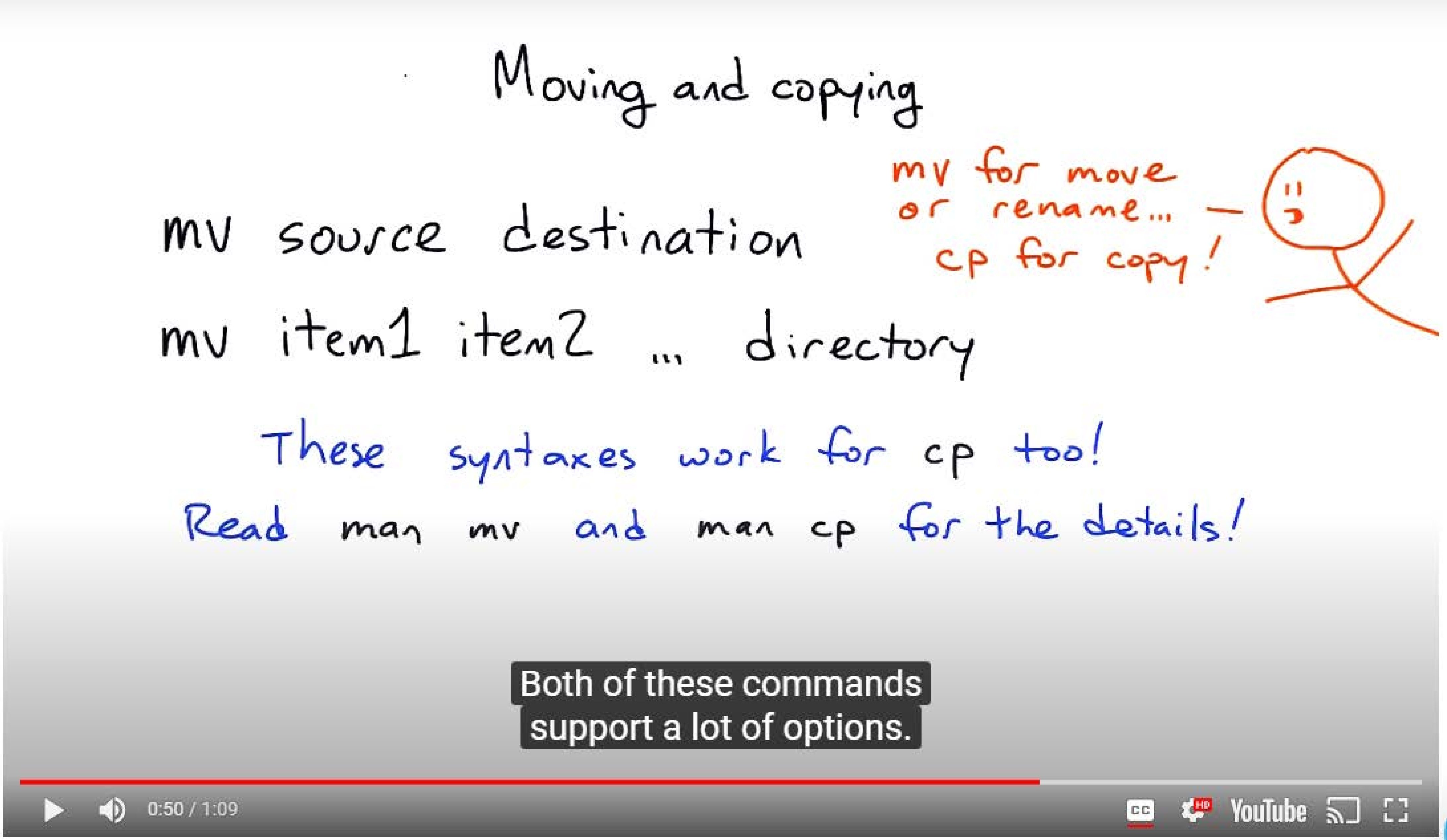
Moving and copying

```
mv source destination

mv item1 item2 ... directory
```

mv for move
or rename...
cp for copy!

The command is cp for copy.



Moving and copying

mv source destination

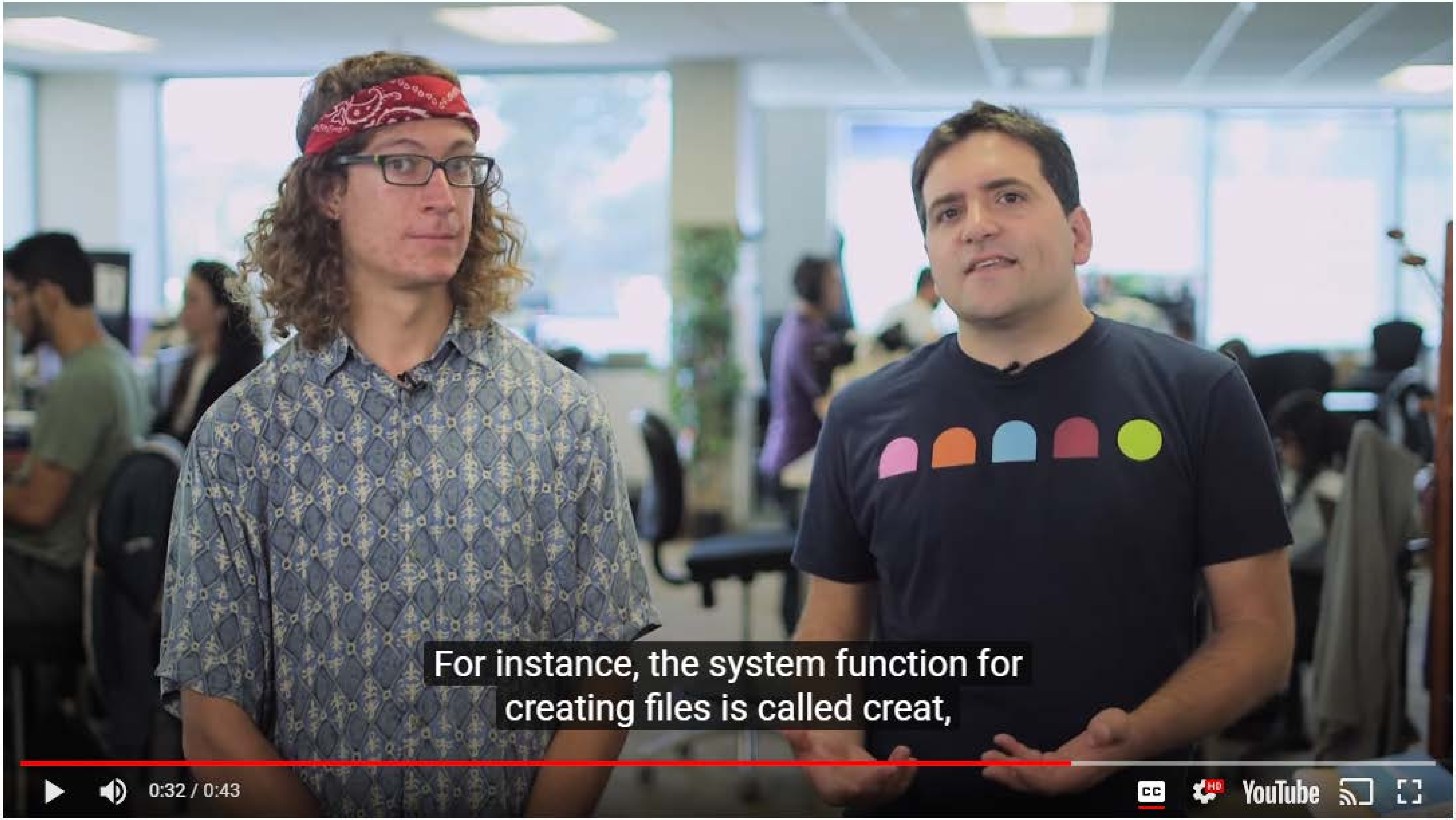
mv item1 item2 ... directory

mv for move or rename... cp for copy!

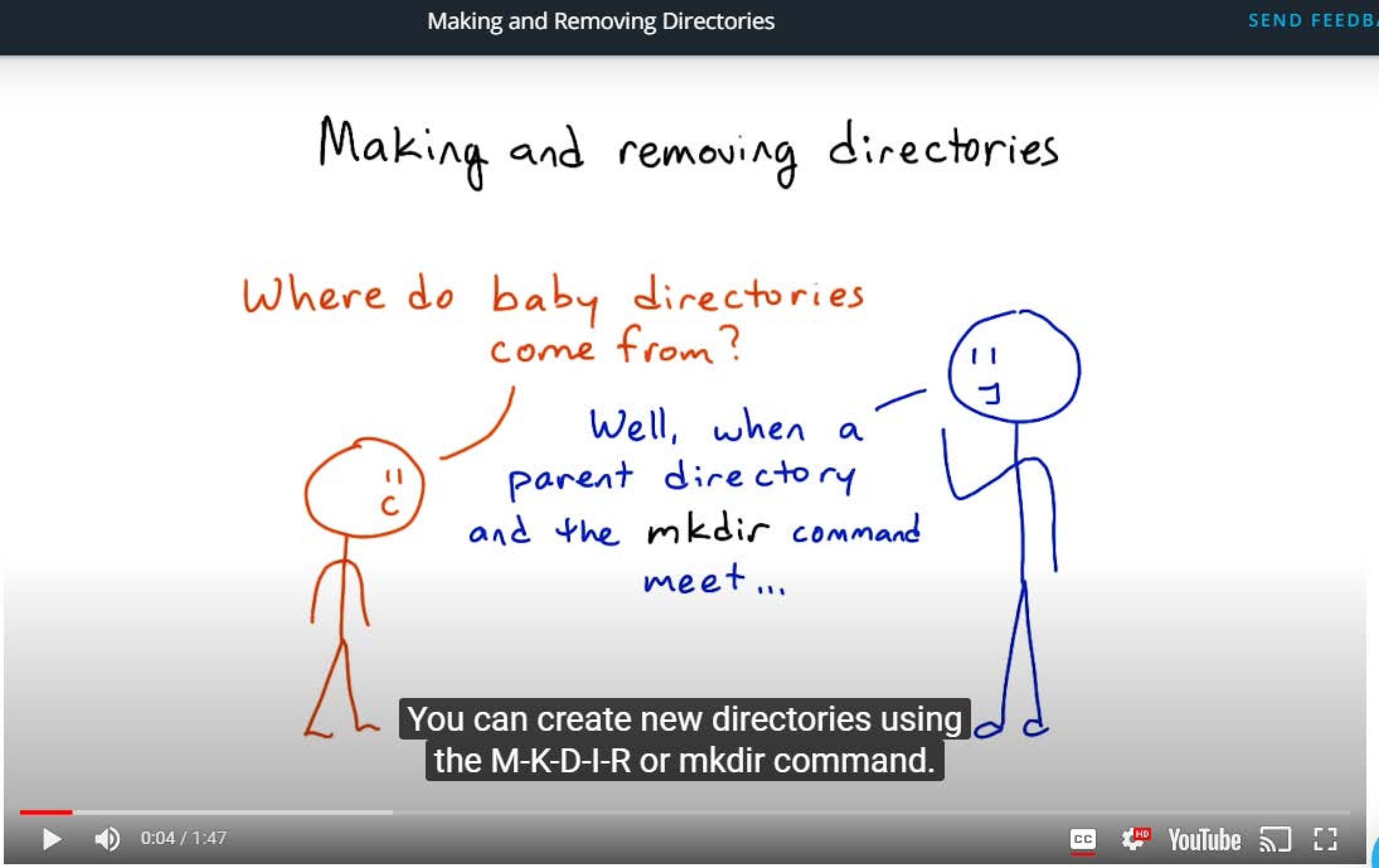
These syntaxes work for cp too!

Read man mv and man cp for the details!

Both of these commands support a lot of options.



For instance, the system function for
creating files is called creat,



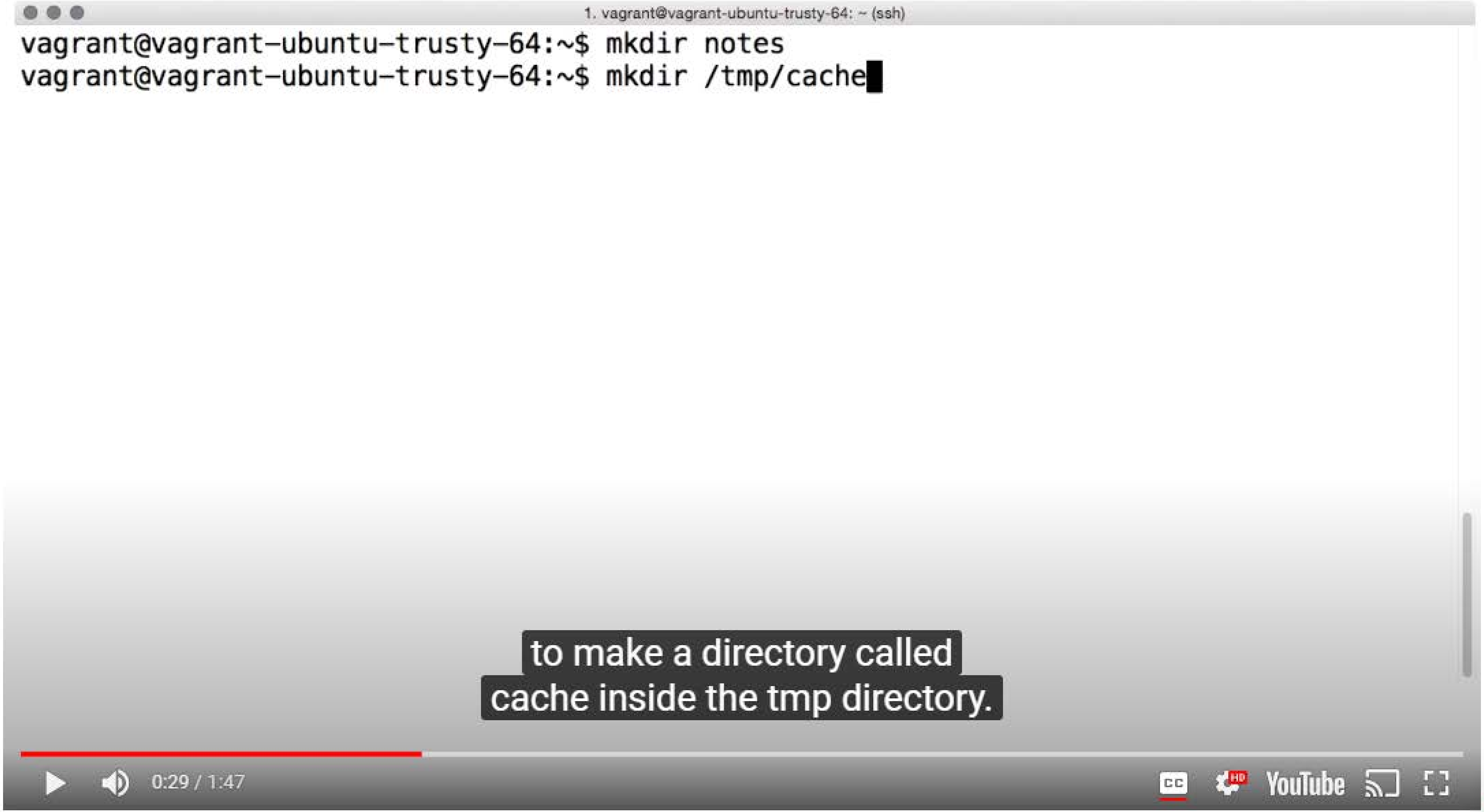
Making and Removing Directories

Making and removing directories

Where do baby directories come from?

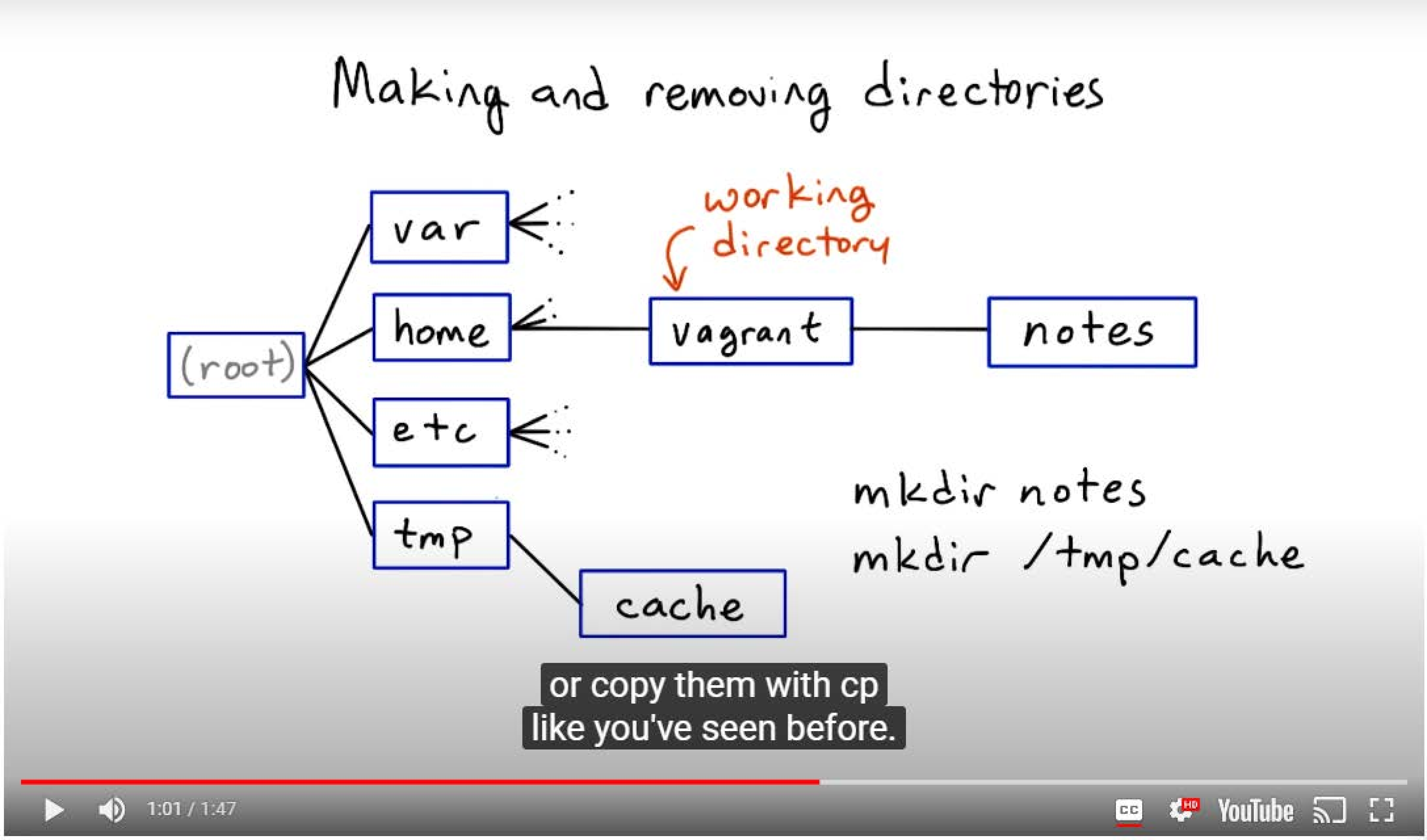
Well, when a parent directory and the mkdir command meet...

You can create new directories using the M-K-D-I-R or mkdir command.



```
vagrant@vagrant-ubuntu-trusty-64:~$ mkdir notes
vagrant@vagrant-ubuntu-trusty-64:~$ mkdir /tmp/cache
```

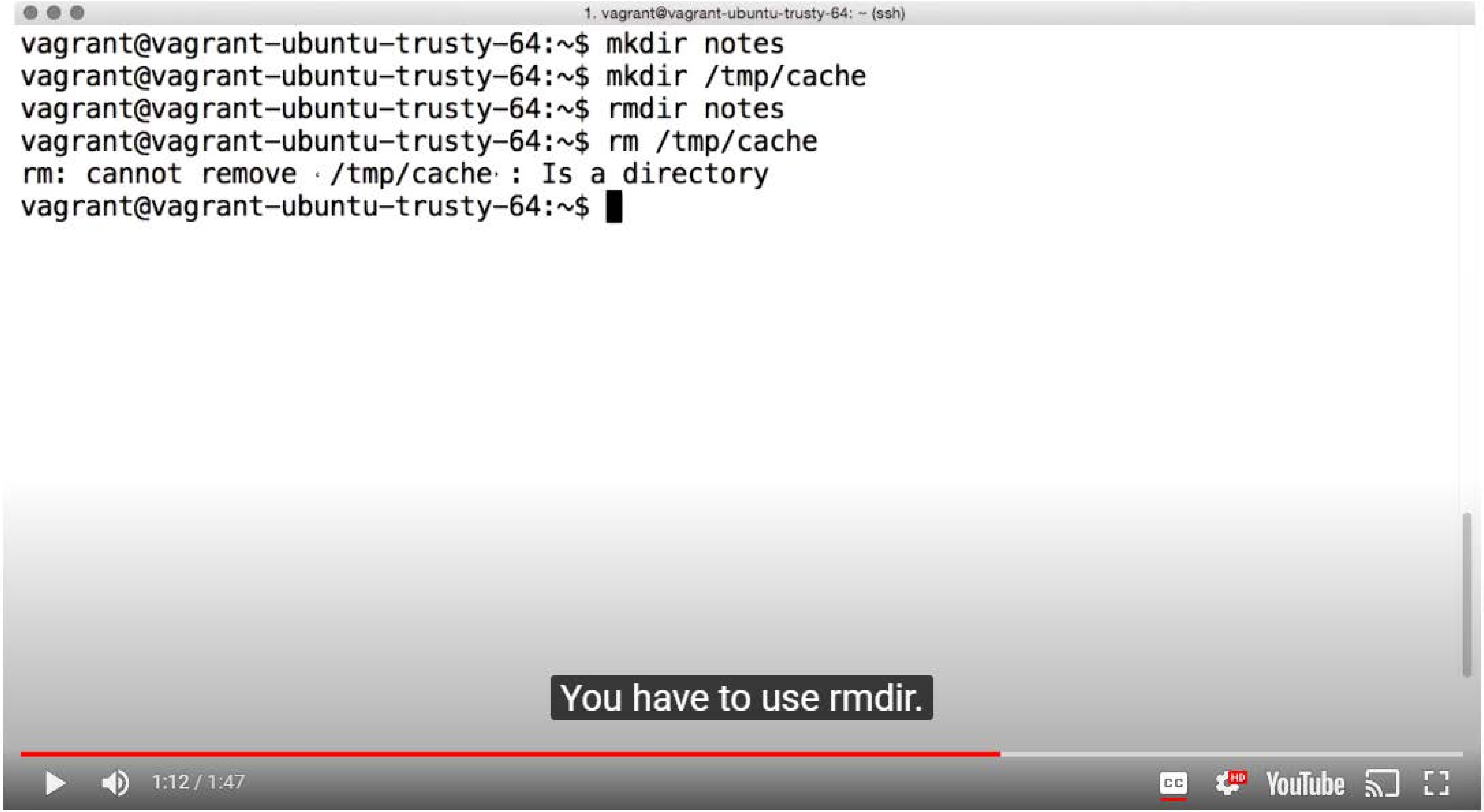
to make a directory called
cache inside the tmp directory.



Making and removing directories

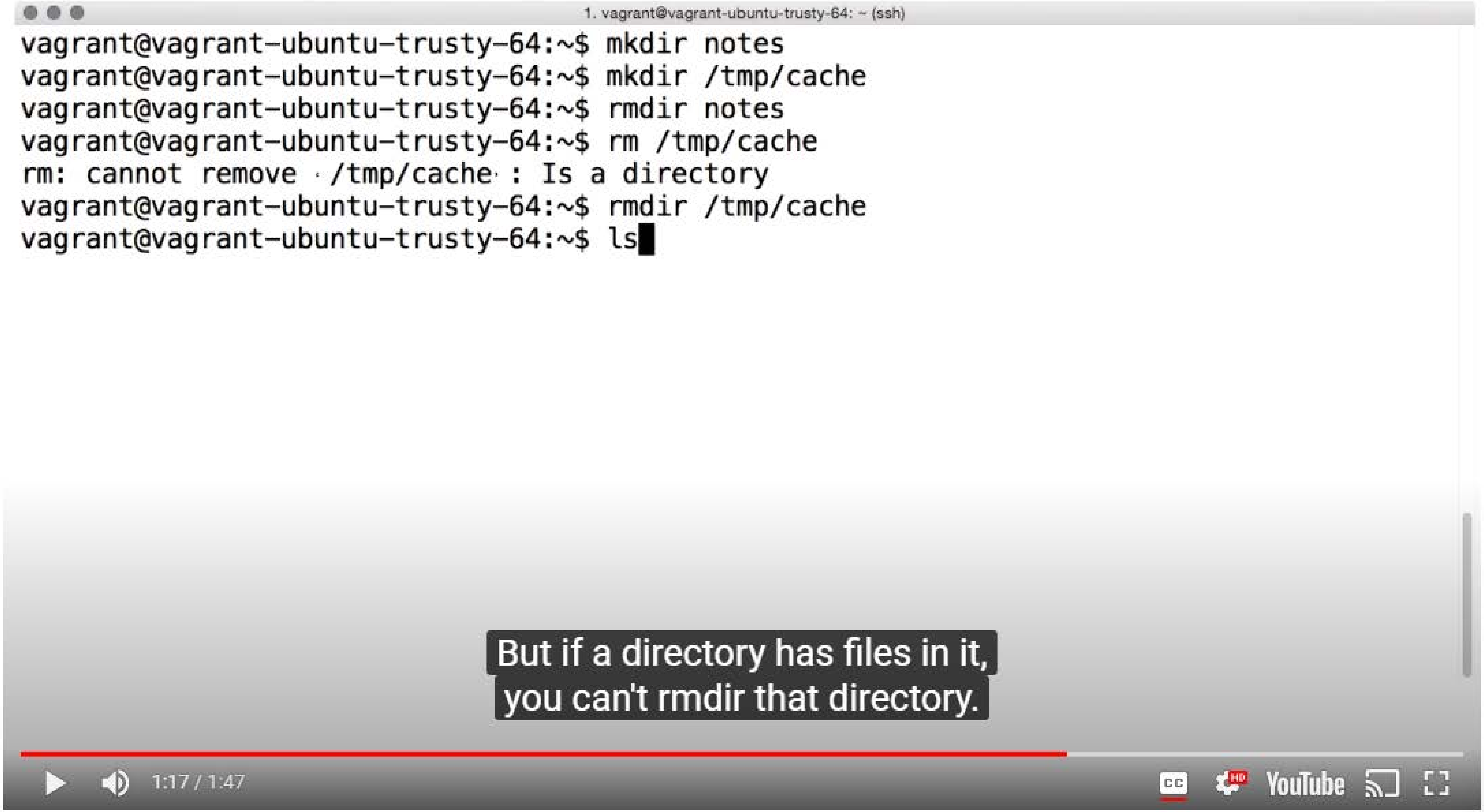
```
mkdir notes
mkdir /tmp/cache
```

or copy them with cp
like you've seen before.



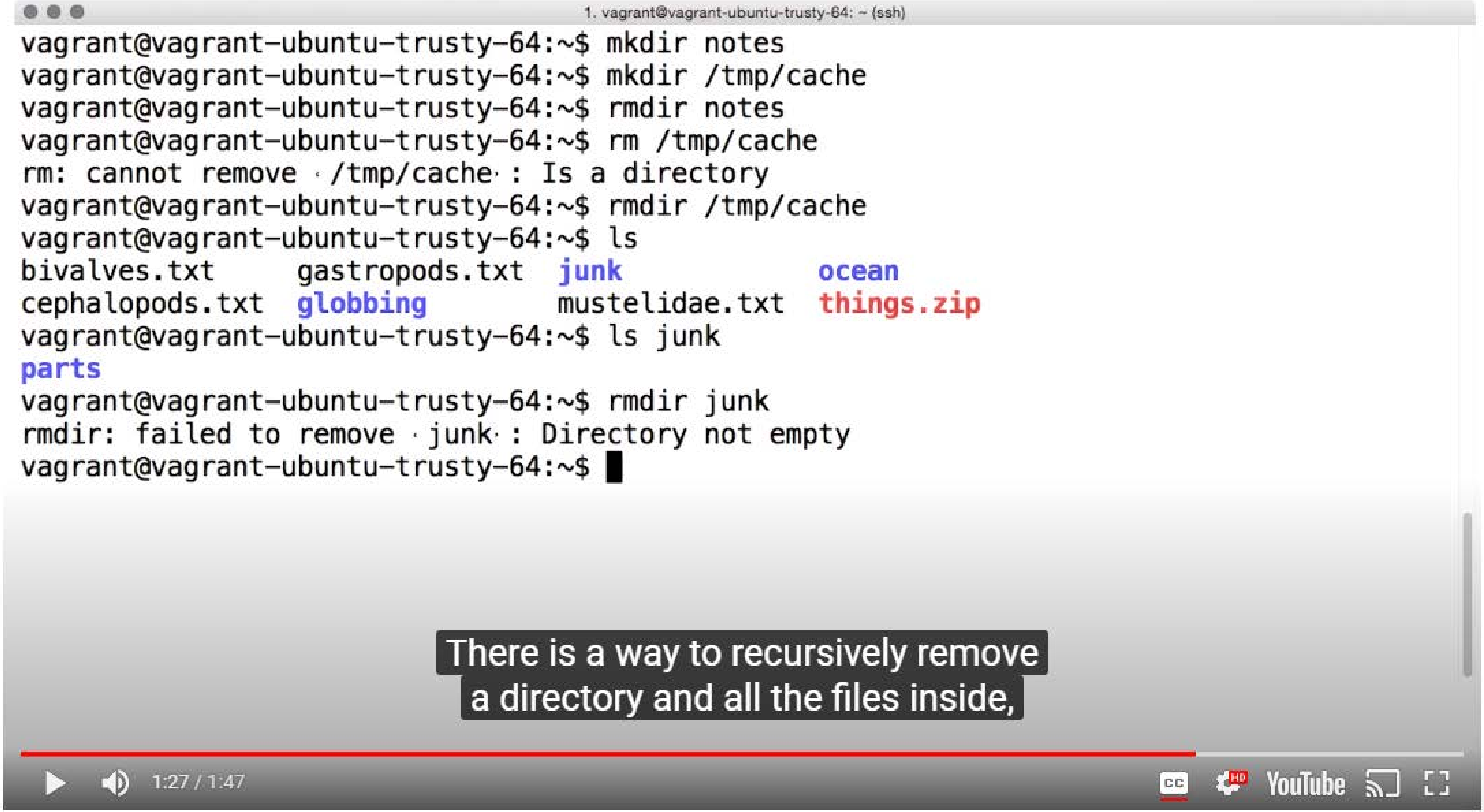
```
vagrant@vagrant-ubuntu-trusty-64:~$ mkdir notes
vagrant@vagrant-ubuntu-trusty-64:~$ mkdir /tmp/cache
vagrant@vagrant-ubuntu-trusty-64:~$ rmdir notes
vagrant@vagrant-ubuntu-trusty-64:~$ rm /tmp/cache
rm: cannot remove ‘/tmp/cache’: Is a directory
vagrant@vagrant-ubuntu-trusty-64:~$
```

You have to use rmdir.



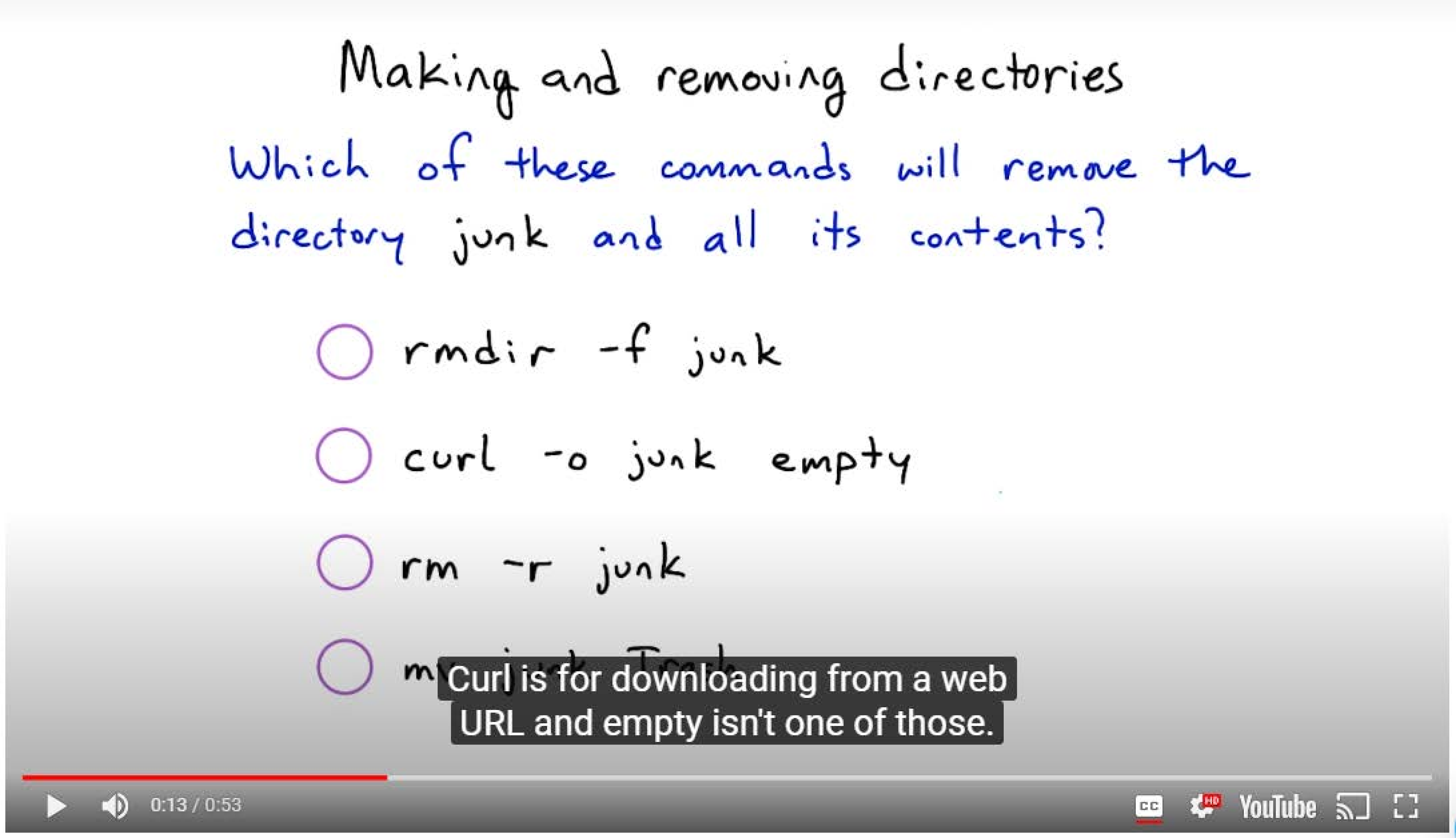
```
vagrant@vagrant-ubuntu-trusty-64:~$ mkdir notes
vagrant@vagrant-ubuntu-trusty-64:~$ mkdir /tmp/cache
vagrant@vagrant-ubuntu-trusty-64:~$ rmdir notes
vagrant@vagrant-ubuntu-trusty-64:~$ rm /tmp/cache
rm: cannot remove ‘/tmp/cache’: Is a directory
vagrant@vagrant-ubuntu-trusty-64:~$ rmdir /tmp/cache
vagrant@vagrant-ubuntu-trusty-64:~$ ls
```

But if a directory has files in it,
you can't rmdir that directory.



```
vagrant@vagrant-ubuntu-trusty-64:~$ mkdir notes
vagrant@vagrant-ubuntu-trusty-64:~$ mkdir /tmp/cache
vagrant@vagrant-ubuntu-trusty-64:~$ rmdir notes
vagrant@vagrant-ubuntu-trusty-64:~$ rm /tmp/cache
rm: cannot remove ‘/tmp/cache’: Is a directory
vagrant@vagrant-ubuntu-trusty-64:~$ rmdir /tmp/cache
vagrant@vagrant-ubuntu-trusty-64:~$ ls
bivalves.txt     gastropods.txt  junk            ocean
cephalopods.txt  globbing        mustelidae.txt  things.zip
vagrant@vagrant-ubuntu-trusty-64:~$ ls junk
parts
vagrant@vagrant-ubuntu-trusty-64:~$ rmdir junk
rmdir: failed to remove ‘junk’: Directory not empty
vagrant@vagrant-ubuntu-trusty-64:~$ █
```

There is a way to recursively remove
a directory and all the files inside,



Making and removing directories

Which of these commands will remove the directory junk and all its contents?

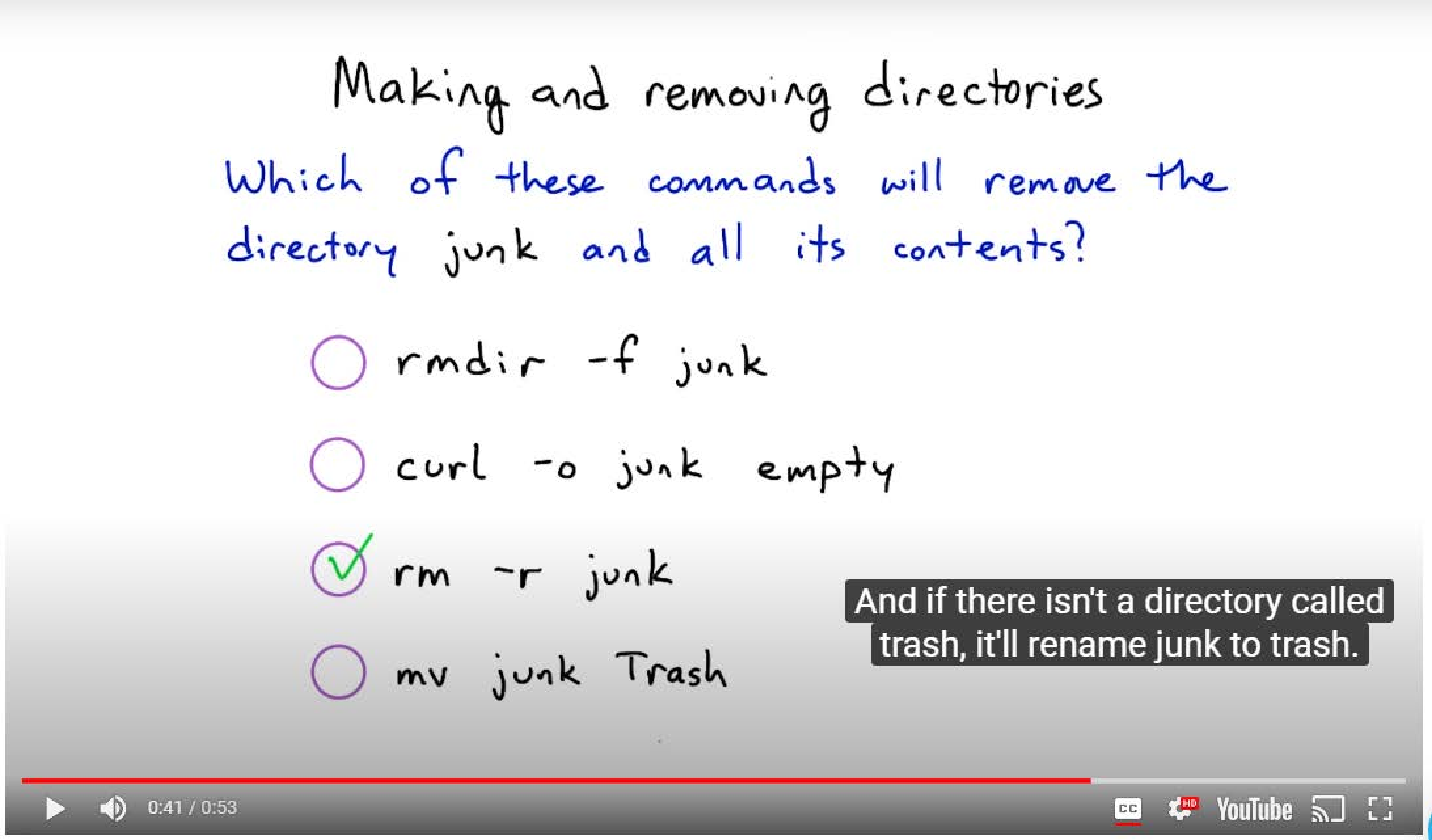
○ rmdir -f junk

○ curl -o junk empty

○ rm -r junk

○ mv junk Trash

Curl is for downloading from a web URL and empty isn't one of those.



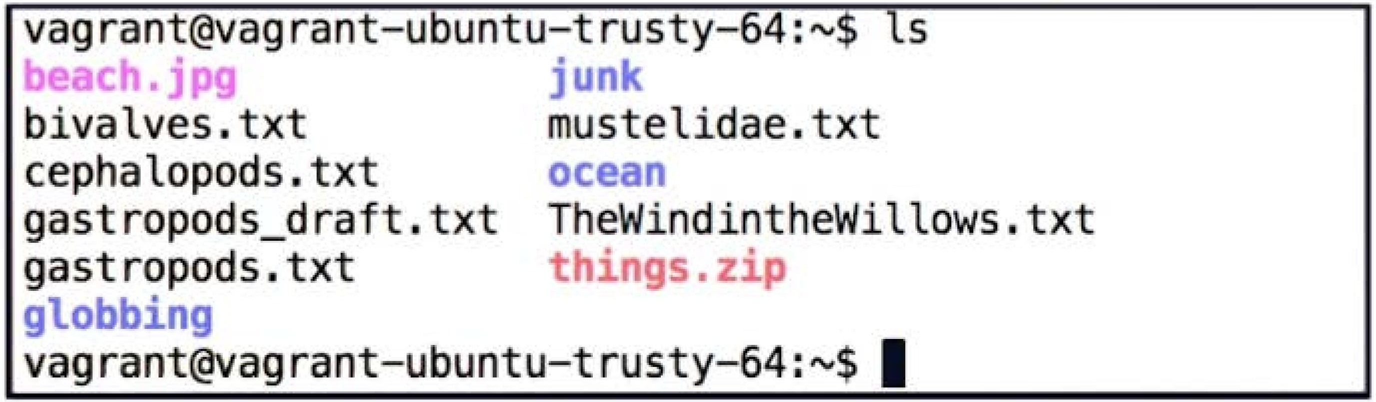
Making and removing directories

Which of these commands will remove the directory junk and all its contents?

- ○ rmdir -f junk
- ○ curl -o junk empty
- ✓ rm -r junk
- ○ mv junk Trash

And if there isn't a directory called trash, it'll rename junk to trash.

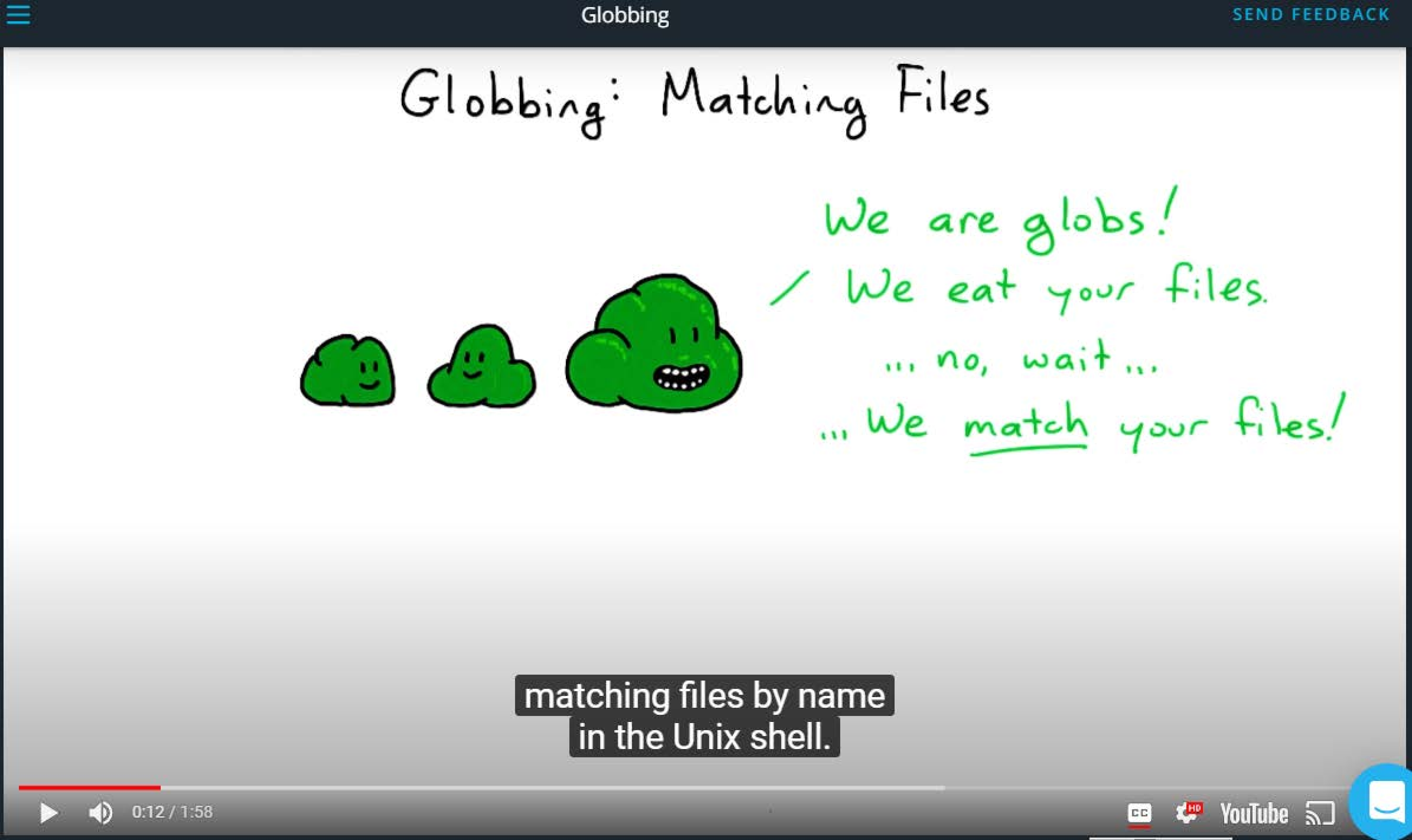
How would you make a new directory called Photos and move beach.jpg into it?



```
vagrant@vagrant-ubuntu-trusty-64:~$ ls
beach.jpg                junk
bivalves.txt             mustelidae.txt
cephalopods.txt          ocean
gastropods_draft.txt     TheWindintheWillows.txt
gastropods.txt           things.zip
globbing
vagrant@vagrant-ubuntu-trusty-64:~$
```

Enter the commands here:

```
mkdir Photos
mv beach.jpg Photos
```



Globbing: Matching Files

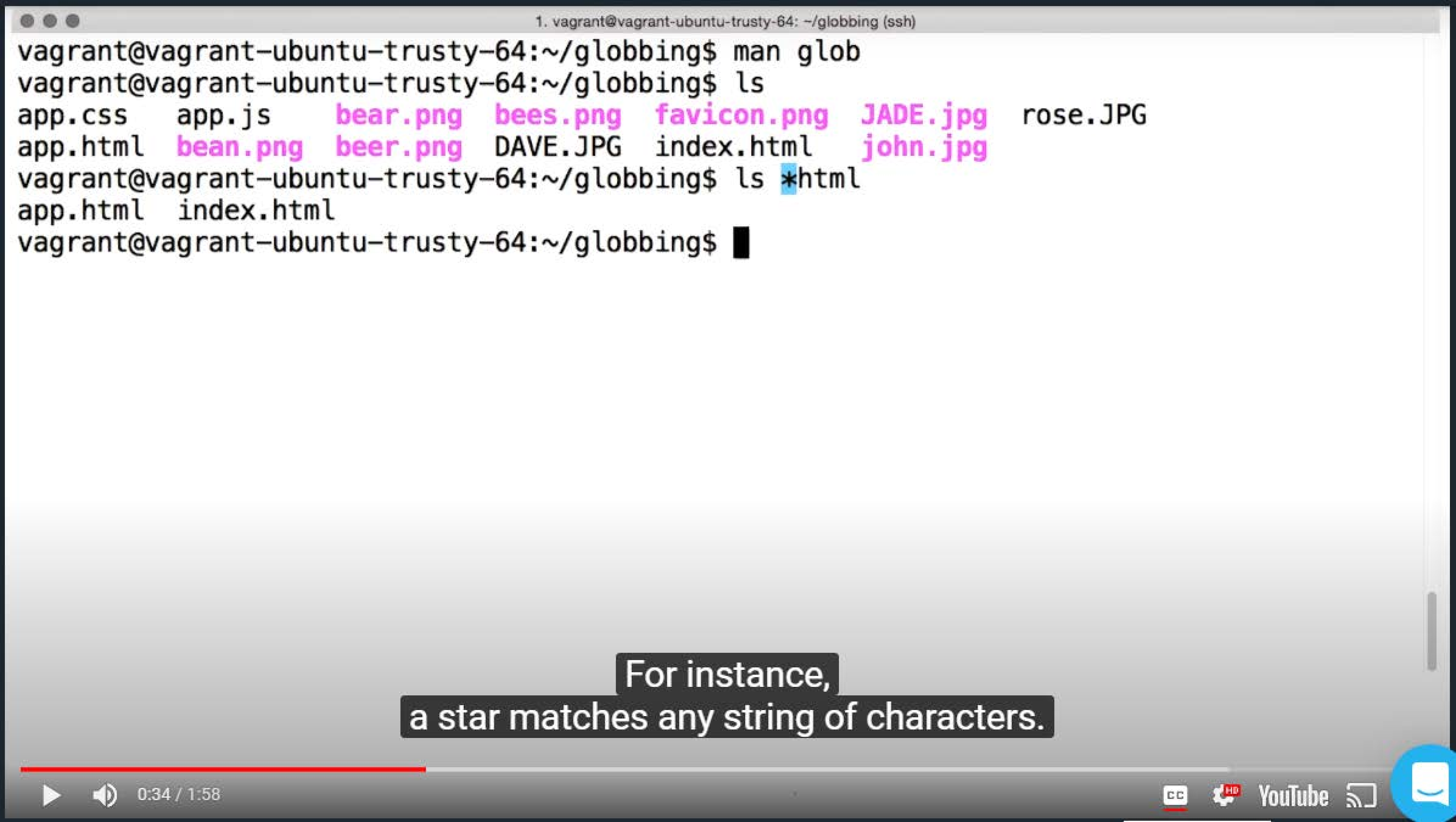
We are globs!

~~We eat your files.~~

... no, wait...

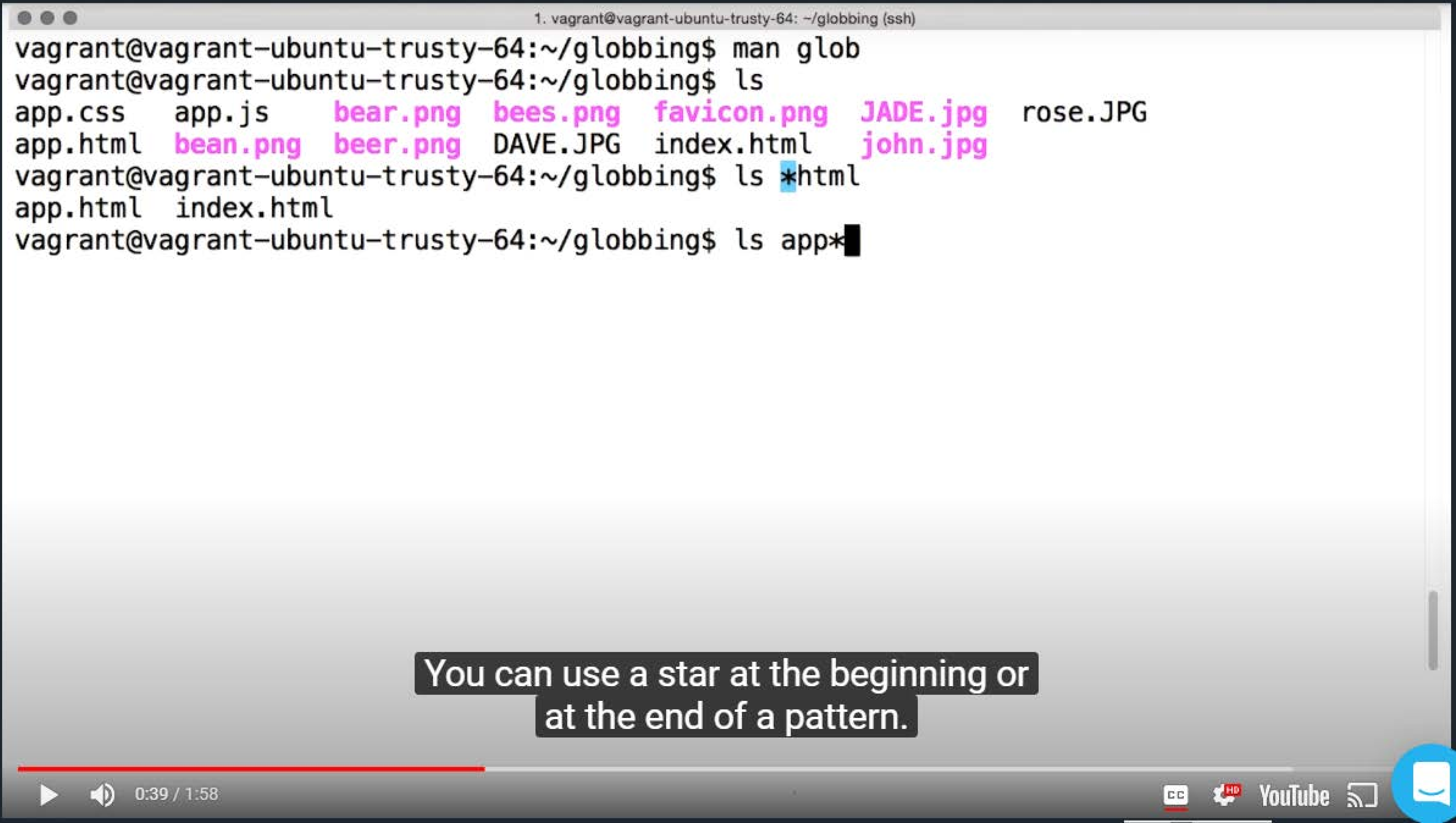
... We match your files!

matching files by name
in the Unix shell.



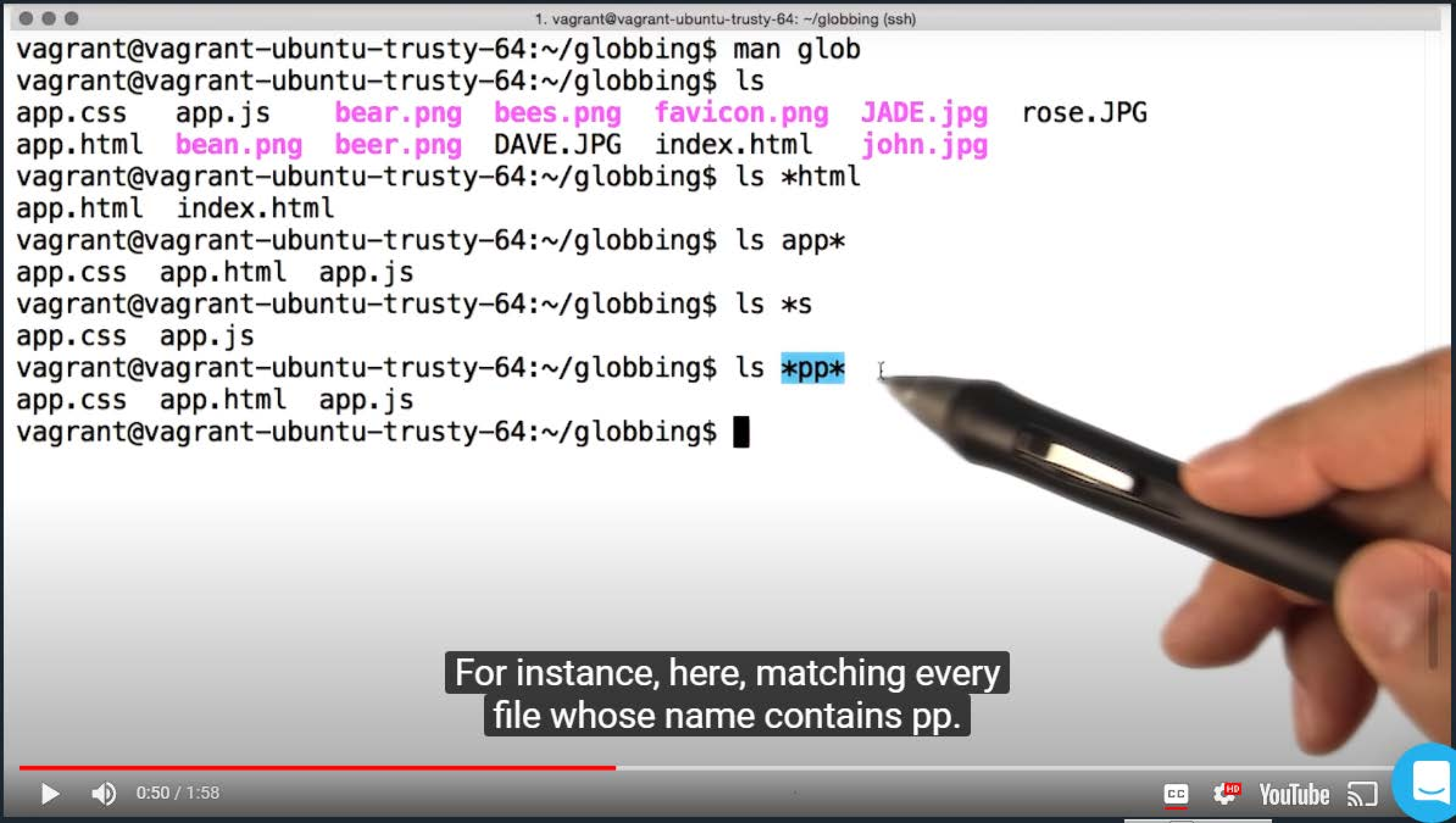
```
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ man glob
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls
app.css   app.js    bear.png  bees.png  favicon.png  JADE.jpg  rose.JPG
app.html  bean.png  beer.png  DAVE.JPG  index.html   john.jpg
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls *html
app.html  index.html
vagrant@vagrant-ubuntu-trusty-64:~/globbing$
```

For instance,
a star matches any string of characters.



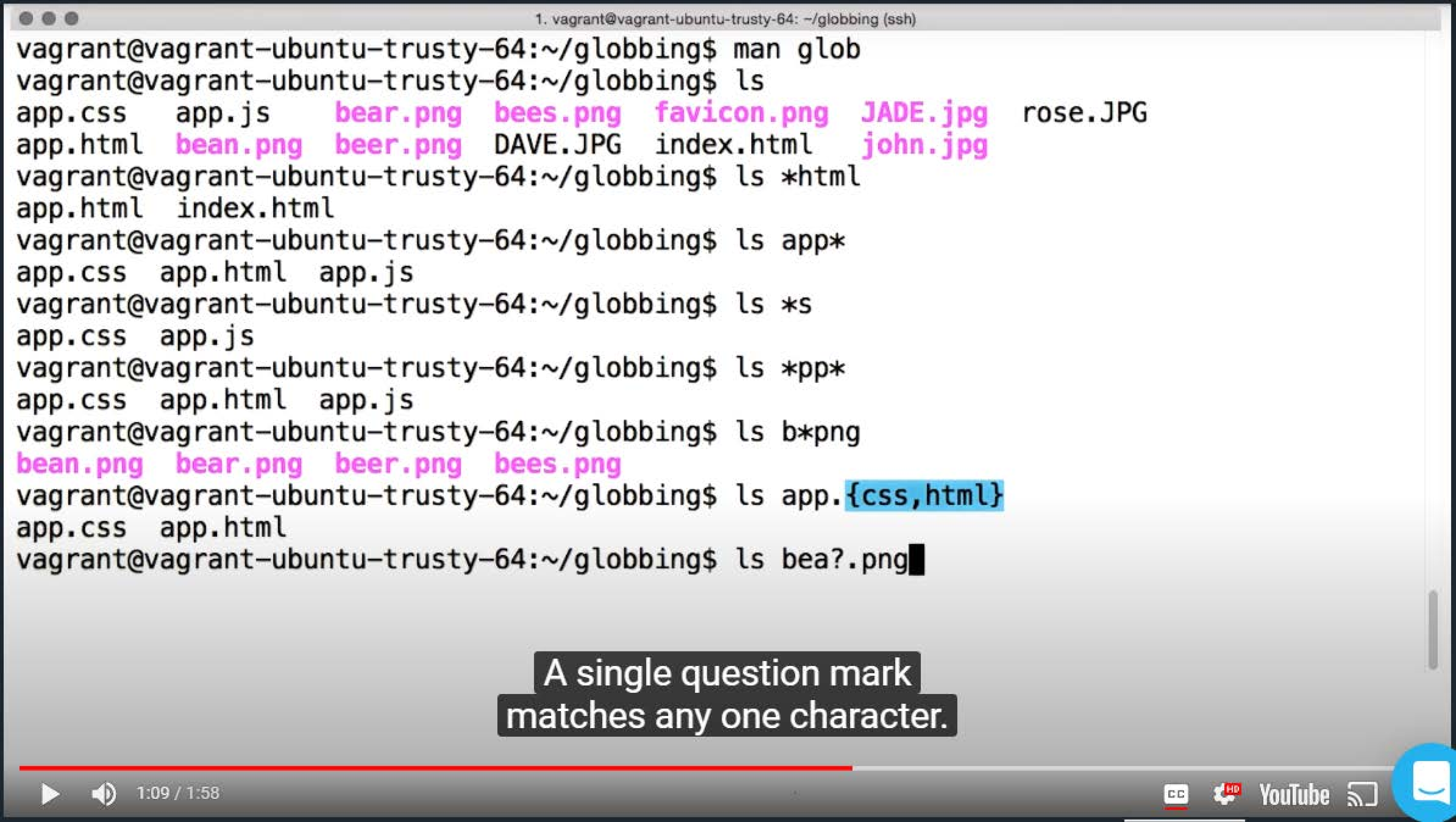
```
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ man glob
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls
app.css   app.js    bear.png  bees.png  favicon.png  JADE.jpg  rose.JPG
app.html  bean.png  beer.png  DAVE.JPG  index.html   john.jpg
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls *html
app.html  index.html
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls app*
```

You can use a star at the beginning or
at the end of a pattern.



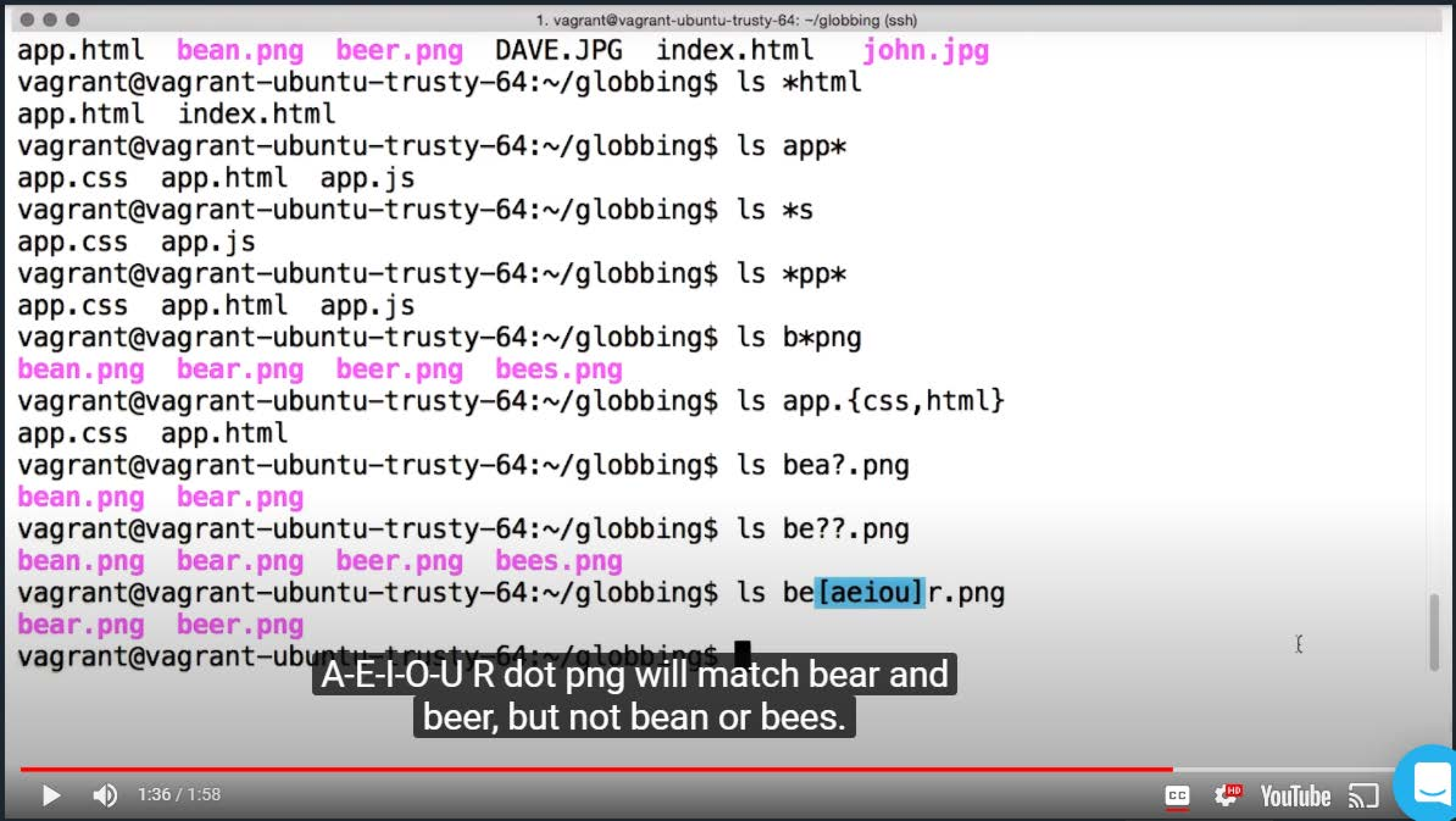
```
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ man glob
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls
app.css   app.js    bear.png  bees.png  favicon.png  JADE.jpg  rose.JPG
app.html  bean.png  beer.png  DAVE.JPG  index.html   john.jpg
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls *html
app.html  index.html
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls app*
app.css  app.html  app.js
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls *s
app.css  app.js
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls *pp*
app.css  app.html  app.js
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ 
```

For instance, here, matching every file whose name contains pp.



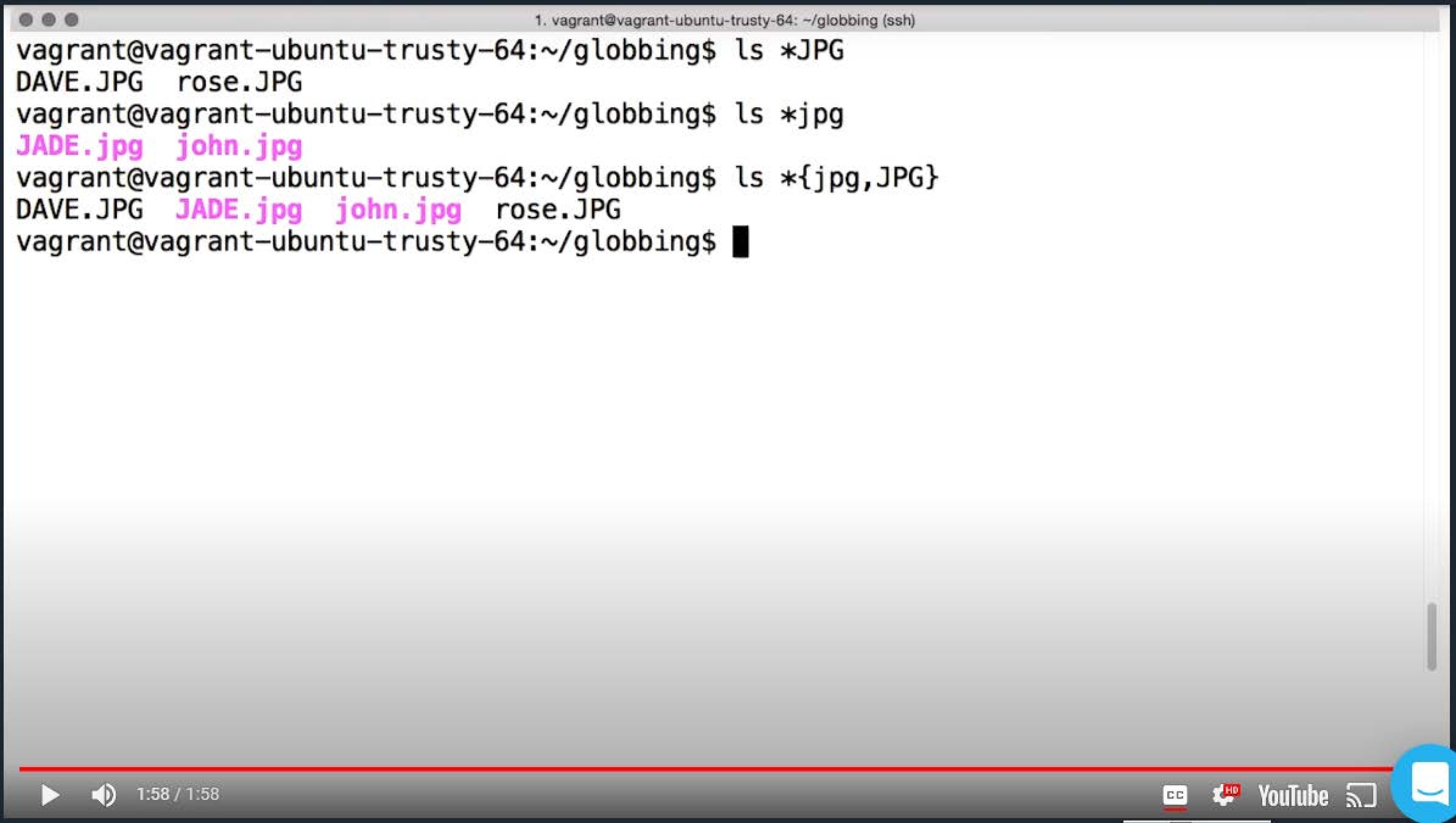
```
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ man glob
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls
app.css   app.js    bear.png  bees.png  favicon.png  JADE.jpg  rose.JPG
app.html  bean.png  beer.png  DAVE.JPG  index.html   john.jpg
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls *html
app.html  index.html
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls app*
app.css  app.html  app.js
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls *s
app.css  app.js
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls *pp*
app.css  app.html  app.js
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls b*png
bean.png  bear.png  beer.png  bees.png
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls app.{css,html}
app.css  app.html
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls bea?.png
```

A single question mark
matches any one character.

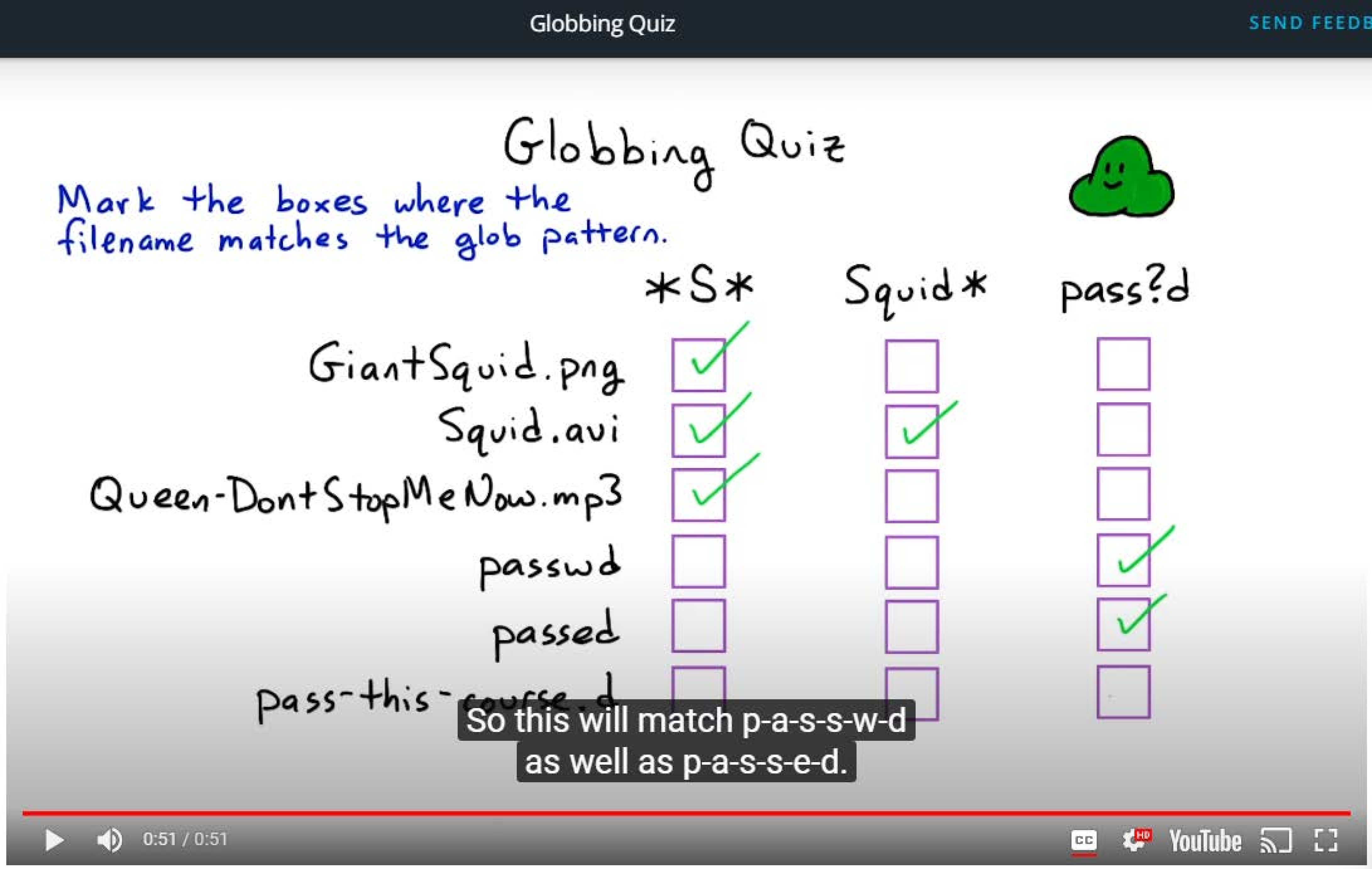


```
app.html  bean.png  beer.png  DAVE.JPG  index.html   john.jpg
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls *html
app.html  index.html
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls app*
app.css  app.html  app.js
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls *s
app.css  app.js
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls *pp*
app.css  app.html  app.js
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls b*png
bean.png  bear.png  beer.png  bees.png
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls app.{css,html}
app.css  app.html
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls bea?.png
bean.png  bear.png
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls be??.png
bean.png  bear.png  beer.png  bees.png
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls be[aeiou]r.png
bear.png  beer.png
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ 
```

A-E-I-O-U R dot png will match bear and beer, but not bean or bees.



```
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls *JPG
DAVE.JPG  rose.JPG
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls *jpg
JADE.jpg  john.jpg
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ ls *{jpg,JPG}
DAVE.JPG  JADE.jpg  john.jpg  rose.JPG
vagrant@vagrant-ubuntu-trusty-64:~/globbing$ 
```

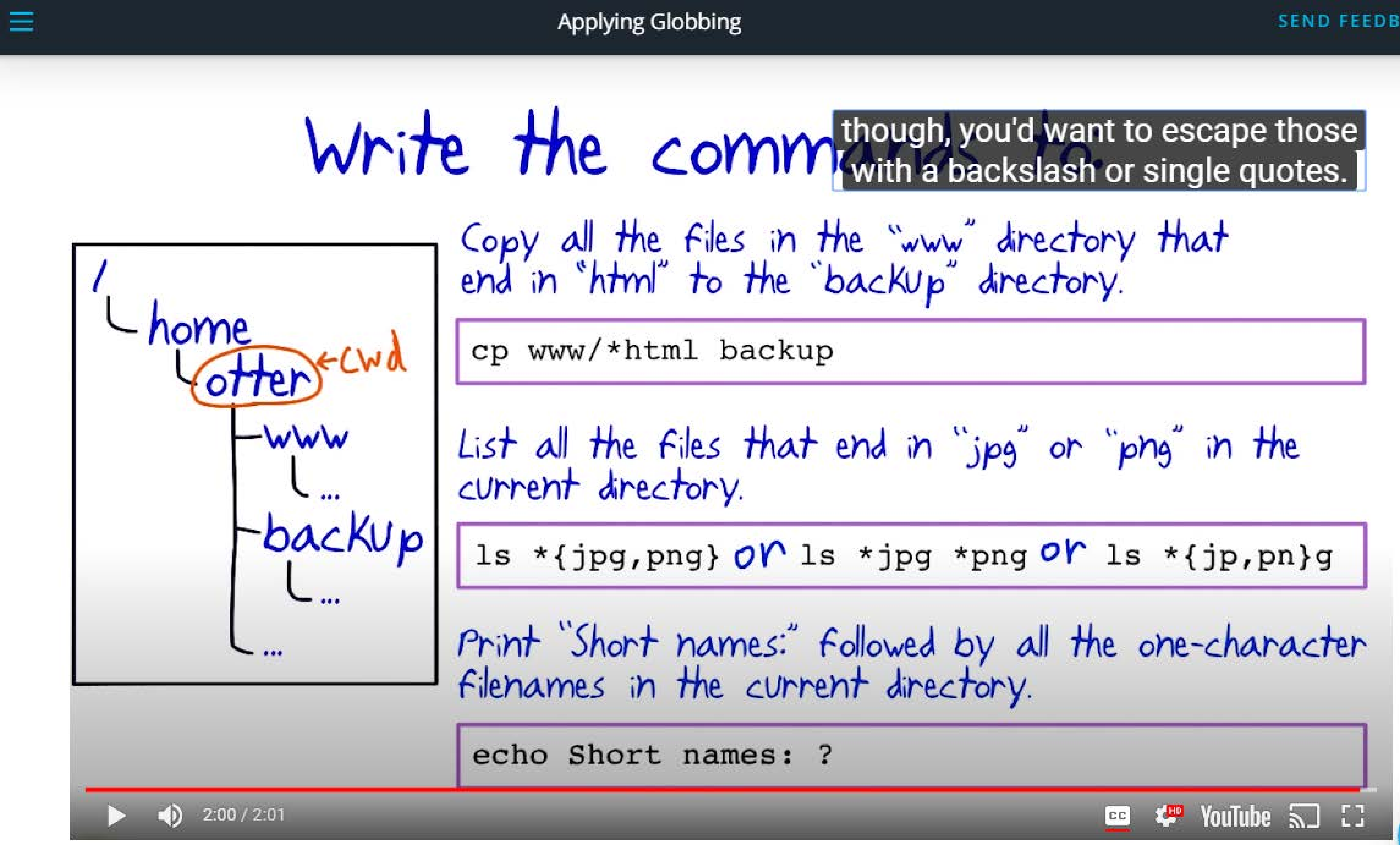


Globbing Quiz

Mark the boxes where the filename matches the glob pattern.

	S	Squid*	pass?d
GiantSquid.png	☑	□	□
Squid.avi	☑	☑	□
Queen-DontStopMeNow.mp3	☑	□	□
passwd	□	□	☑
passed	□	□	☑
pass-this-course.d	□	□	□

So this will match p-a-s-s-w-d as well as p-a-s-s-e-d.



Write the commands to:

though, you'd want to escape those with a backslash or single quotes.

Copy all the files in the "www" directory that end in "html" to the "backup" directory.

```
cp www/*html backup
```

List all the files that end in "jpg" or "png" in the current directory.

```
ls *{jpg,png} or ls *jpg *png or ls *{jp,pn}g
```

Print "Short names:" followed by all the one-character filenames in the current directory.

```
echo Short names: ?
```