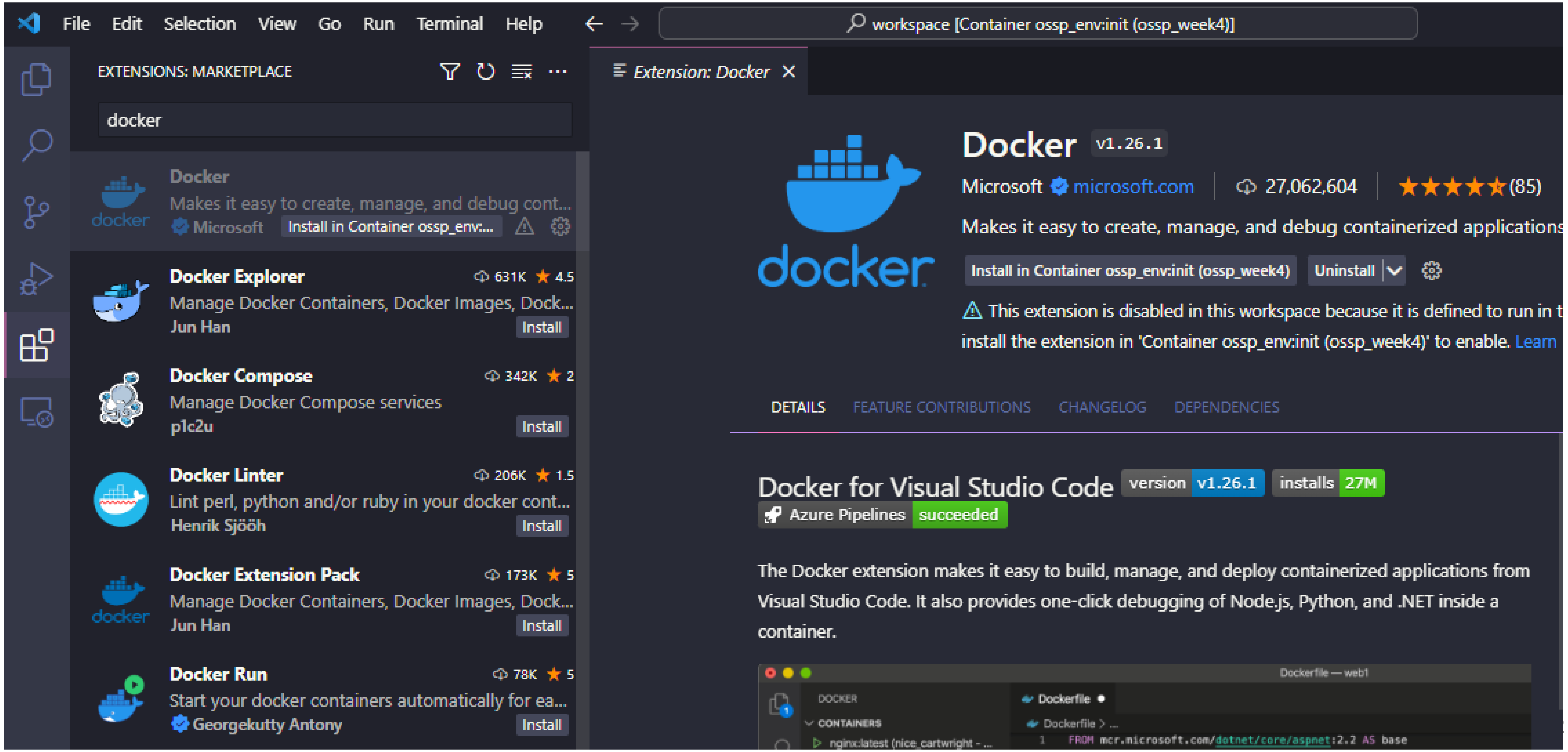
Bash Programming [SWE2021]

JinYeong Bak
jy.bak@skku.edu
Human Language Intelligence Lab, SKKU
Slide credit: TaekHyun Kim (treecko.kth@g.skku.edu)
YeongJun Hwang (hmtyj2@g.skku.edu)

Install Docker Extension



File Edit Selection View Go Run Terminal Help

workspace [Container ossp_env:init (ossp_week4)]

EXTENSIONS: MARKETPLACE

docker

Docker
Makes it easy to create, manage, and debug cont...
Microsoft — Install in Container ossp_env:...

Docker Explorer 631K ★ 4.5
Manage Docker Containers, Docker Images, Dock...
Jun Han — Install

Docker Compose 342K ★ 2
Manage Docker Compose services
p1c2u — Install

Docker Linter 206K ★ 1.5
Lint perl, python and/or ruby in your docker cont...
Henrik Sjööh — Install

Docker Extension Pack 173K ★ 5
Manage Docker Containers, Docker Images, Dock...
Jun Han — Install

Docker Run 78K ★ 5
Start your docker containers automatically for ea...
Georgekutty Antony — Install

Extension: Docker

Docker v1.26.1
Microsoft microsoft.com | 27,062,604 | ★★★★☆ (85)
Makes it easy to create, manage, and debug containerized applications

Install in Container ossp_env:init (ossp_week4) | Uninstall

⚠ This extension is disabled in this workspace because it is defined to run in t
install the extension in 'Container ossp_env:init (ossp_week4)' to enable. Learn

DETAILS FEATURE CONTRIBUTIONS CHANGELOG DEPENDENCIES

Docker for Visual Studio Code version v1.26.1 installs 27M
Azure Pipelines succeeded

The Docker extension makes it easy to build, manage, and deploy containerized applications from Visual Studio Code. It also provides one-click debugging of Node.js, Python, and .NET inside a container.

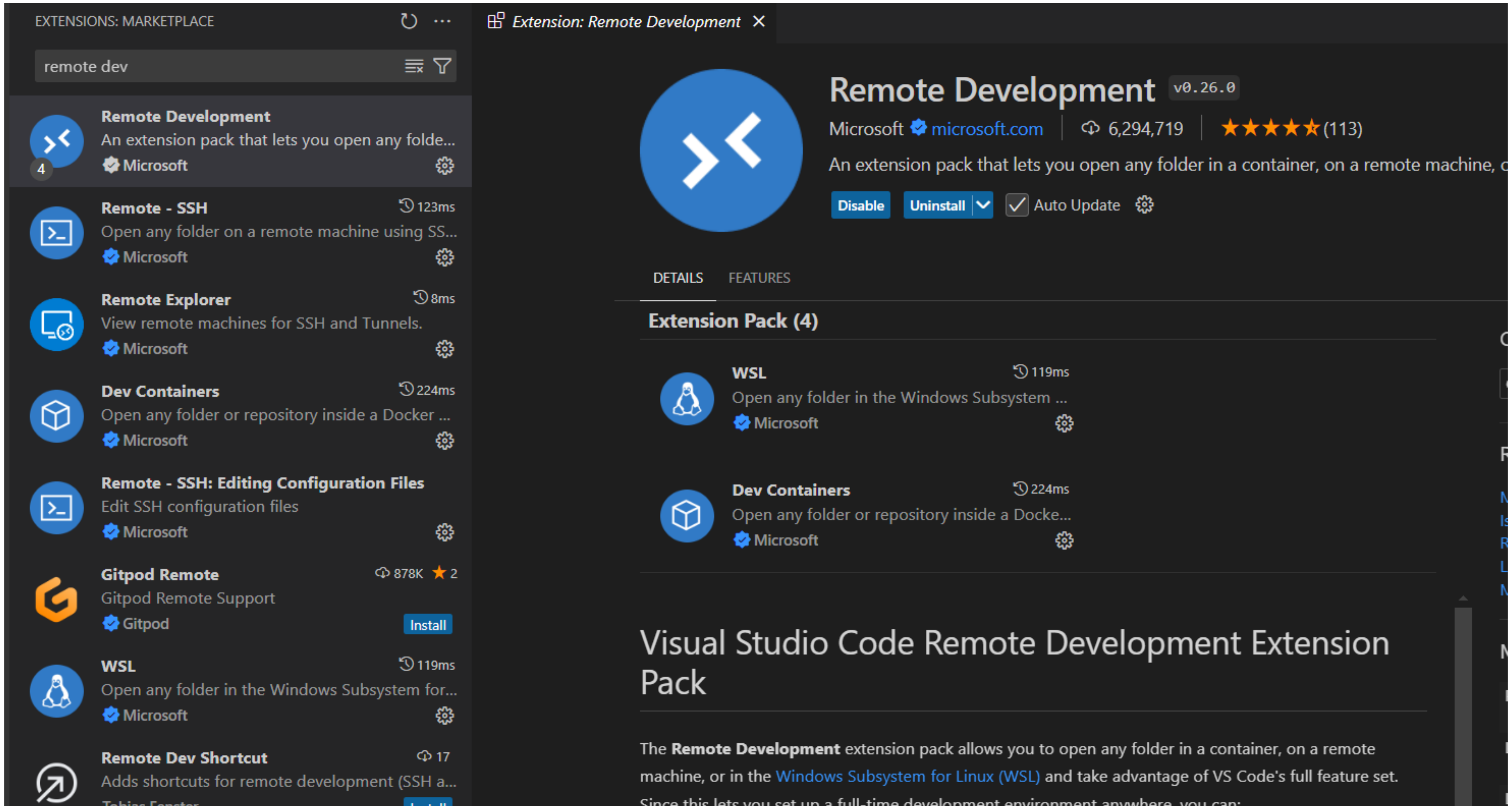
Dockerfile — web1

DOCKER
CONTAINERS
nginx:latest (nice_cartwright - ...

Dockerfile
Dockerfile > ...

```
FROM mcr.microsoft.com/dotnet/core/aspnet:2.2 AS base
```

Install Remote Development Extension



EXTENSIONS: MARKETPLACE

remote dev

Remote Development
An extension pack that lets you open any folde...
Microsoft

Remote - SSH 123ms
Open any folder on a remote machine using SS...
Microsoft

Remote Explorer 8ms
View remote machines for SSH and Tunnels.
Microsoft

Dev Containers 224ms
Open any folder or repository inside a Docker ...
Microsoft

Remote - SSH: Editing Configuration Files
Edit SSH configuration files
Microsoft

Gitpod Remote 878K ★ 2
Gitpod Remote Support
Gitpod — Install

WSL 119ms
Open any folder in the Windows Subsystem for...
Microsoft

Remote Dev Shortcut 17
Adds shortcuts for remote development (SSH a...

Extension: Remote Development

Remote Development v0.26.0
Microsoft microsoft.com | 6,294,719 | ★★★★★ (113)
An extension pack that lets you open any folder in a container, on a remote machine,

Disable | Uninstall | Auto Update

DETAILS | FEATURES

Extension Pack (4)

WSL 119ms
Open any folder in the Windows Subsystem ...
Microsoft

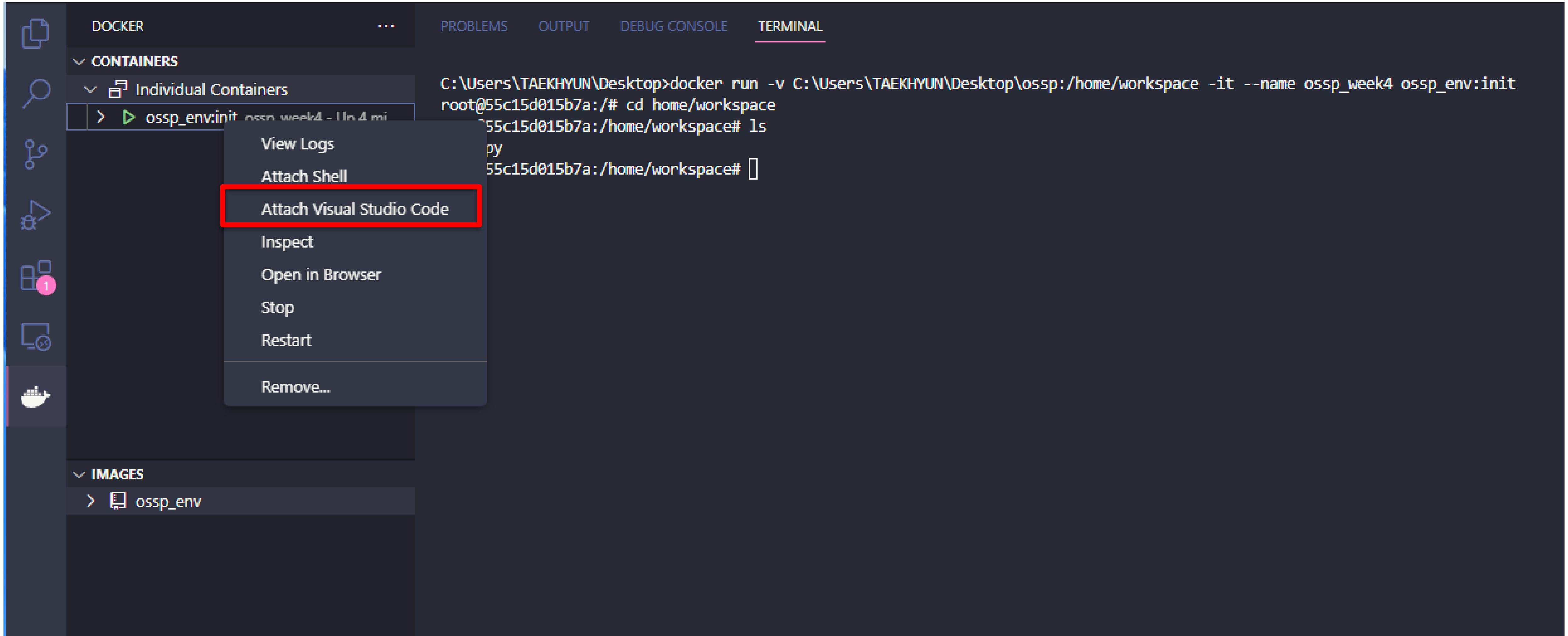
Dev Containers 224ms
Open any folder or repository inside a Docke...
Microsoft

Visual Studio Code Remote Development Extension Pack

The **Remote Development** extension pack allows you to open any folder in a container, on a remote machine, or in the Windows Subsystem for Linux (WSL) and take advantage of VS Code's full feature set.

How to open a new window with docker container

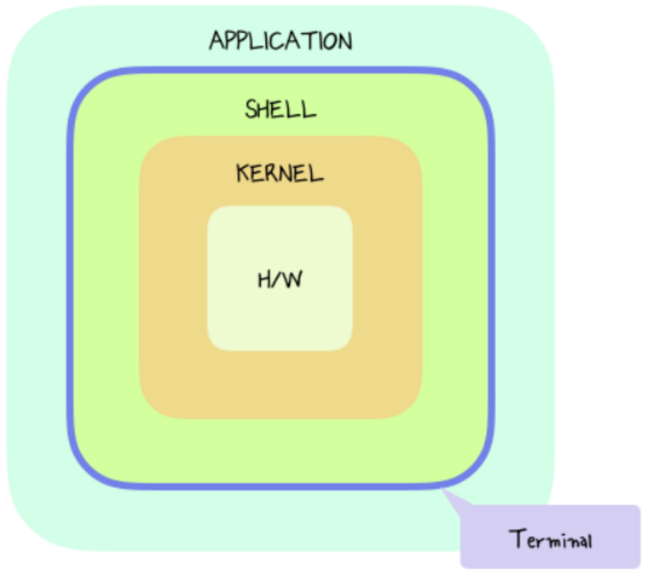
- When you click ‘Attach Visual Studio Code’, it will open new window

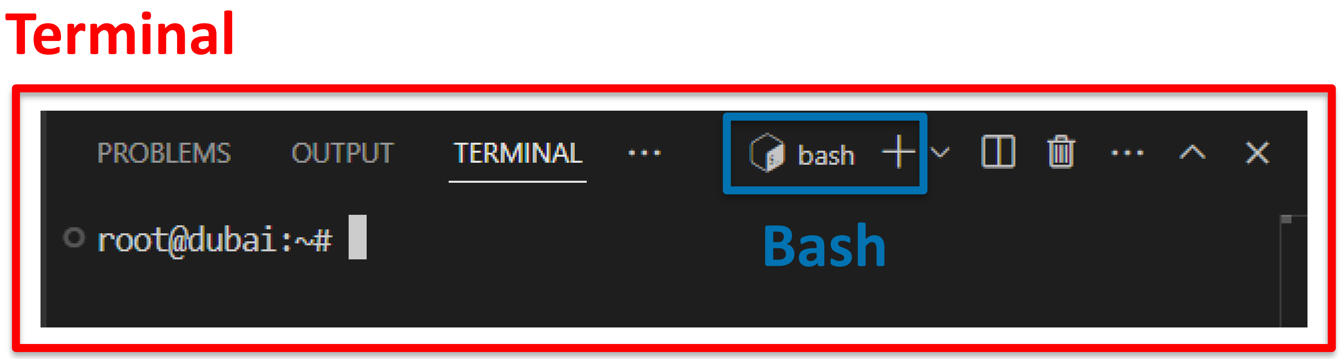


```
C:\Users\TAEKHYUN\Desktop>docker run -v C:\Users\TAEKHYUN\Desktop\ossp:/home/workspace -it --name ossp_week4 ossp_env:init
root@55c15d015b7a:/# cd home/workspace
     55c15d015b7a:/home/workspace# ls
     py
     55c15d015b7a:/home/workspace#
```

Definition of Bash

- **Shell** is the software that interprets and executes the various commands that we type in the terminal
- **Terminal** is the GUI window that you see on the screen
  - Takes commands and shows output
- **Bash** is a particular shell





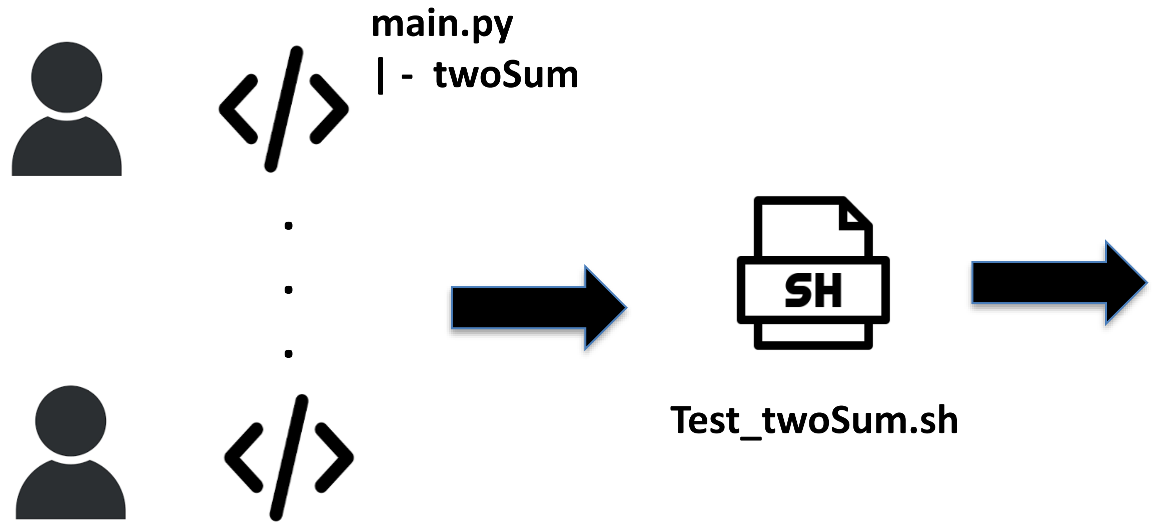
```
root@dubai:~#
```

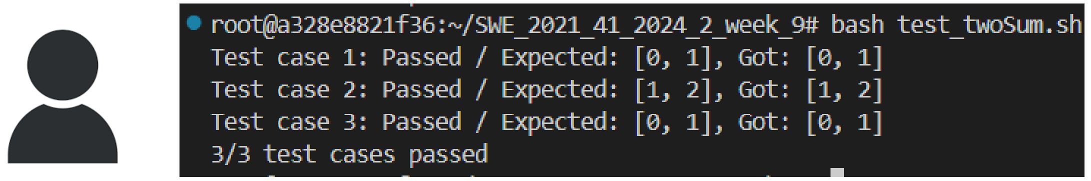
Definition of Bash

- BASH is acronym for ***B**ourne **A**gain **Sh**ell*
- If you use bash…
  - Give commands to the operating system interactively, or to execute batches of commands quickly (Controll OS)
  - Perform basic math, run basic tests and execute applications
  - Combine these operations and connect applications to each other to perform complex and automated tasks

Using Bash

- Bash example
  - Task automation





```
root@a328e8821f36:~/SWE_2021_41_2024_2_week_9# bash test_twoSum.sh
  Test case 1: Passed / Expected: [0, 1], Got: [0, 1]
  Test case 2: Passed / Expected: [1, 2], Got: [1, 2]
  Test case 3: Passed / Expected: [0, 1], Got: [0, 1]
  3/3 test cases passed
```

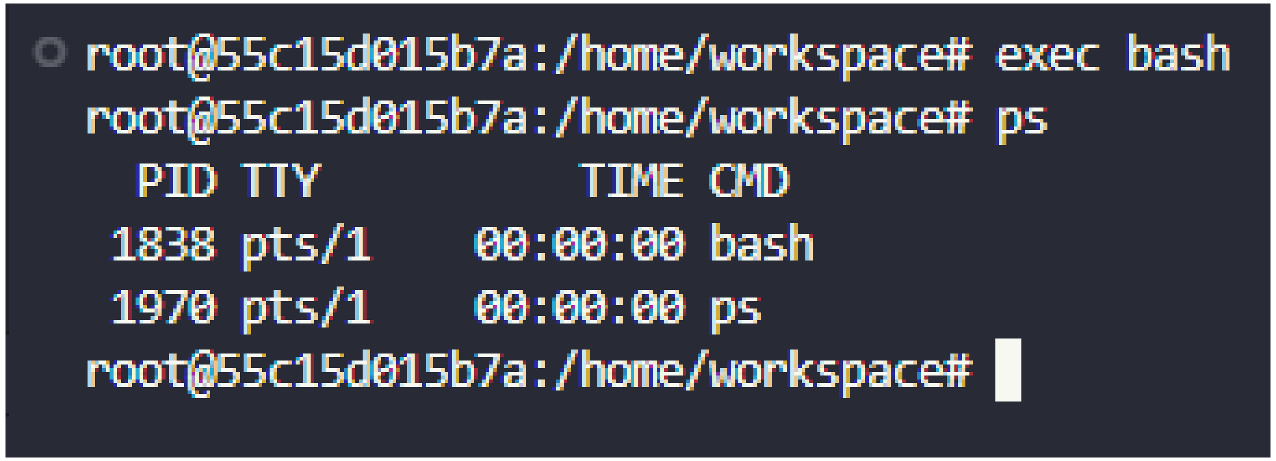
Using Bash

- Interactive mode vs non-interactive mode
  - (Interactive mode) a prompt and a command line
  - (non-interactive mode) executing scripts that are basically lists of commands, but stored in a file

- When a script is executed in non-interactive mode, all these commands are executed sequentially, one after another

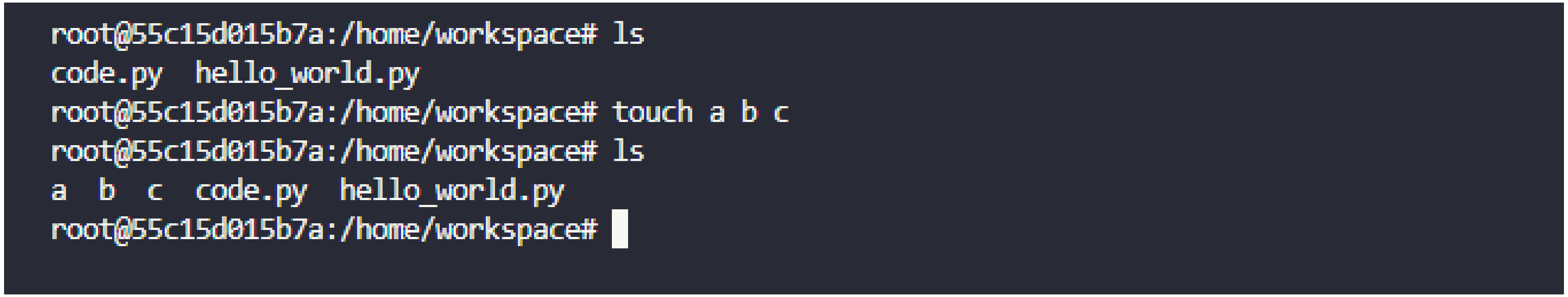
Commands and Arguments

- **ps:** Display information about a selection of the active processes
  - PID: process ID
  - TTY: terminal associated with the process
  - TIME: cumulated CPU time in [DD-]hh:mm:ss format
  - CMD: executable name



```
root@55c15d015b7a:/home/workspace# exec bash
root@55c15d015b7a:/home/workspace# ps
  PID TTY          TIME CMD
 1838 pts/1    00:00:00 bash
 1970 pts/1    00:00:00 ps
root@55c15d015b7a:/home/workspace#
```

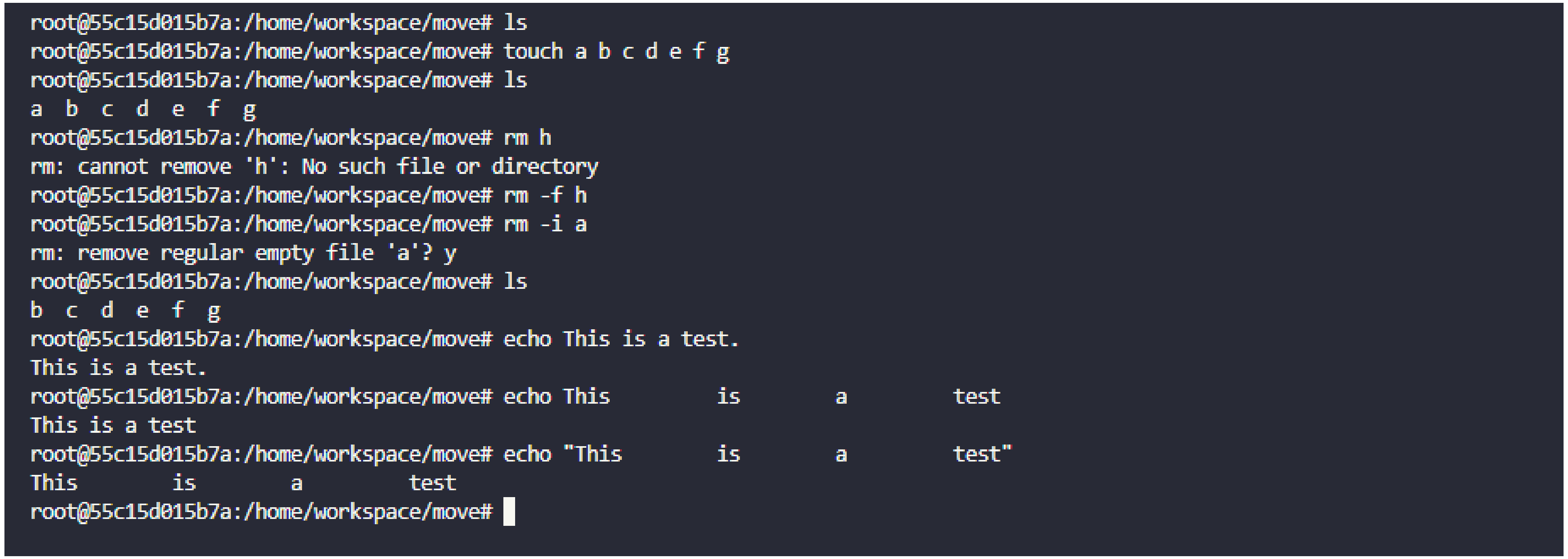
Commands and Arguments



```
root@55c15d015b7a:/home/workspace# ls
code.py  hello_world.py
root@55c15d015b7a:/home/workspace# touch a b c
root@55c15d015b7a:/home/workspace# ls
a  b  c  code.py  hello_world.py
root@55c15d015b7a:/home/workspace# 
```

- **ls**: List files in the current directory
- **touch a b c**: Create files ‘a’, ‘b’, and ‘c’
- **touch:** Changes the Last Modified time of a file. If the filename that it is given does not exist yet, it creates a file of that name as a new and empty file

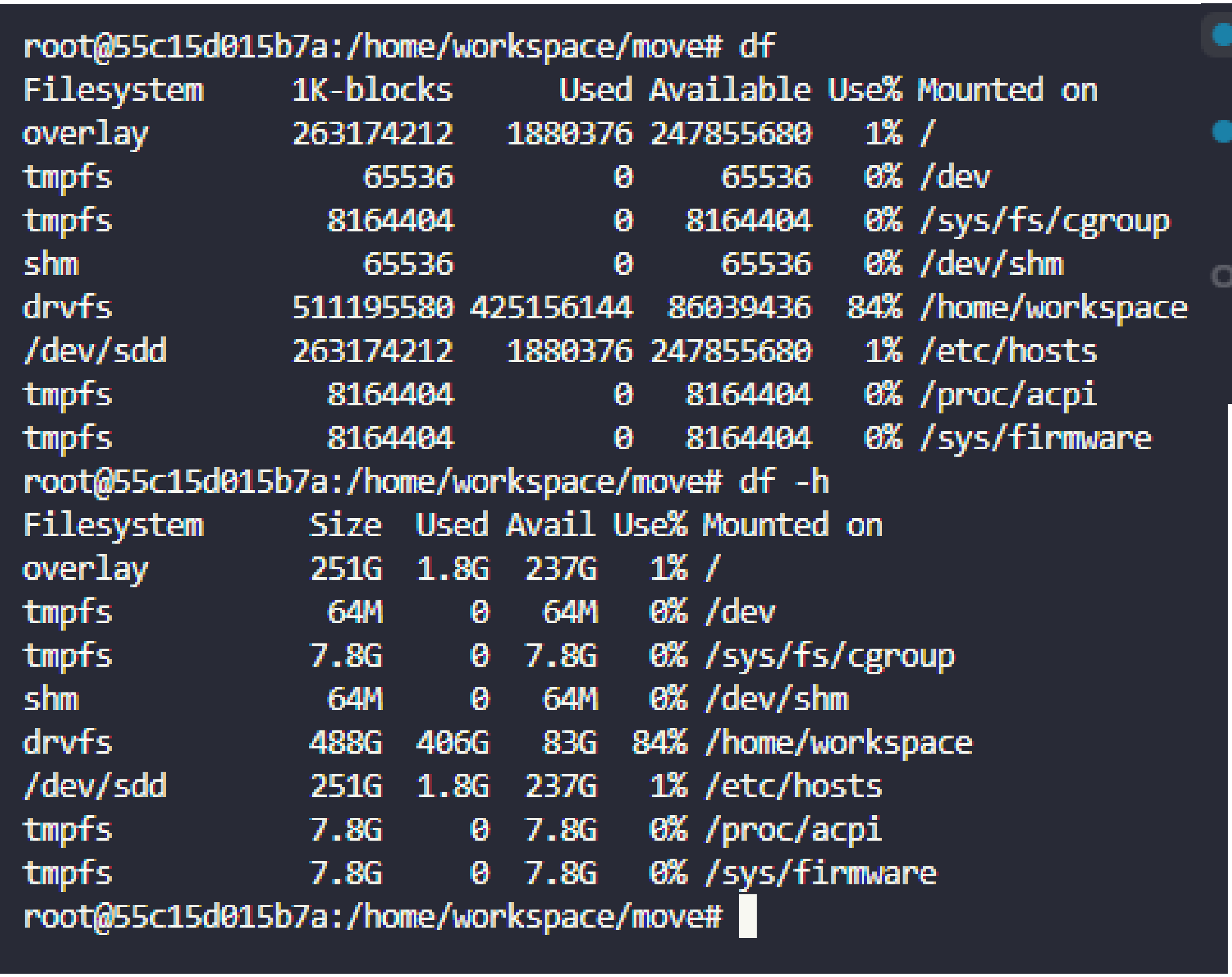
Commands and Arguments



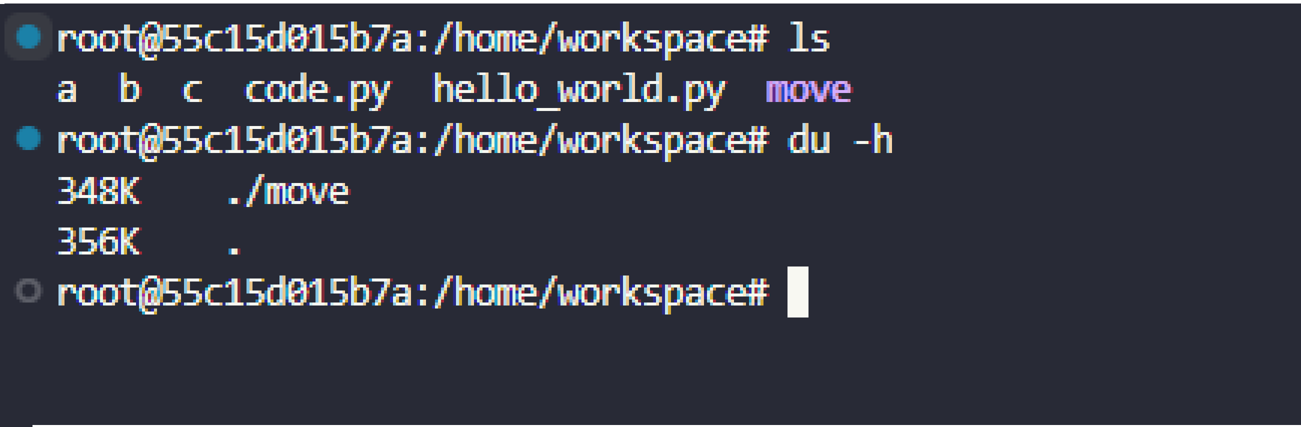
```
root@55c15d015b7a:/home/workspace/move# ls
root@55c15d015b7a:/home/workspace/move# touch a b c d e f g
root@55c15d015b7a:/home/workspace/move# ls
a  b  c  d  e  f  g
root@55c15d015b7a:/home/workspace/move# rm h
rm: cannot remove 'h': No such file or directory
root@55c15d015b7a:/home/workspace/move# rm -f h
root@55c15d015b7a:/home/workspace/move# rm -i a
rm: remove regular empty file 'a'? y
root@55c15d015b7a:/home/workspace/move# ls
b  c  d  e  f  g
root@55c15d015b7a:/home/workspace/move# echo This is a test.
This is a test.
root@55c15d015b7a:/home/workspace/move# echo This         is         a         test
This is a test
root@55c15d015b7a:/home/workspace/move# echo "This        is         a         test"
This        is         a         test
root@55c15d015b7a:/home/workspace/move# 
```

- **rm [options] file:** remove a file in the current directory
  - -f option: Ignore nonexistent files, never prompt
  - -i option: Prompt before every removal
  - -r option: Remove directories and their contents recursively
- **echo:** command that prints its arguments to standard output

Commands and Arguments



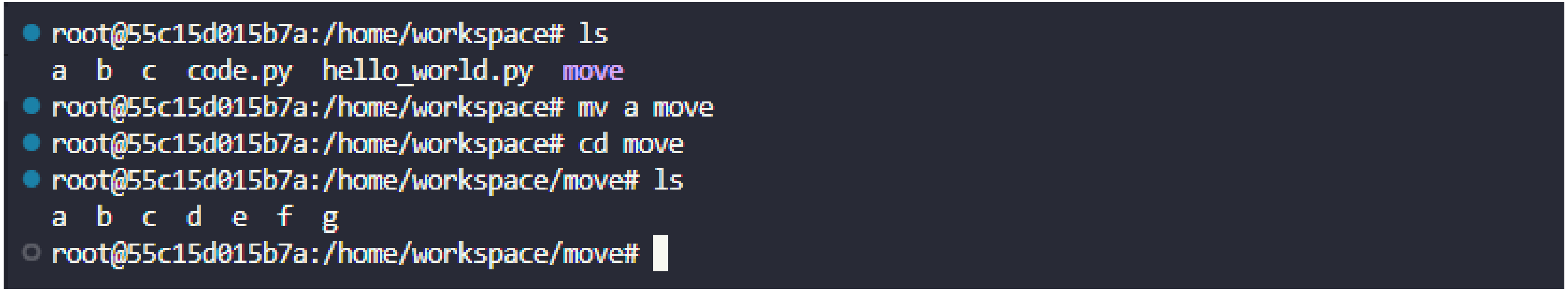
```
root@55c15d015b7a:/home/workspace/move# df
Filesystem     1K-blocks      Used Available Use% Mounted on
overlay        263174212   1880376 247855680   1% /
tmpfs              65536         0     65536   0% /dev
tmpfs            8164404         0   8164404   0% /sys/fs/cgroup
shm                65536         0     65536   0% /dev/shm
drvfs          511195580 425156144  86039436  84% /home/workspace
/dev/sdd       263174212   1880376 247855680   1% /etc/hosts
tmpfs            8164404         0   8164404   0% /proc/acpi
tmpfs            8164404         0   8164404   0% /sys/firmware
root@55c15d015b7a:/home/workspace/move# df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay         251G  1.8G  237G   1% /
tmpfs            64M     0   64M   0% /dev
tmpfs           7.8G     0  7.8G   0% /sys/fs/cgroup
shm              64M     0   64M   0% /dev/shm
drvfs           488G  406G   83G  84% /home/workspace
/dev/sdd        251G  1.8G  237G   1% /etc/hosts
tmpfs           7.8G     0  7.8G   0% /proc/acpi
tmpfs           7.8G     0  7.8G   0% /sys/firmware
root@55c15d015b7a:/home/workspace/move#
```



```
root@55c15d015b7a:/home/workspace# ls
a  b  c  code.py  hello_world.py  move
root@55c15d015b7a:/home/workspace# du -h
348K    ./move
356K    .
root@55c15d015b7a:/home/workspace#
```

- **df:** Display the amount of disk space available
  - -h option: human-readable
- **du**: Disk usage, estimate file space usage

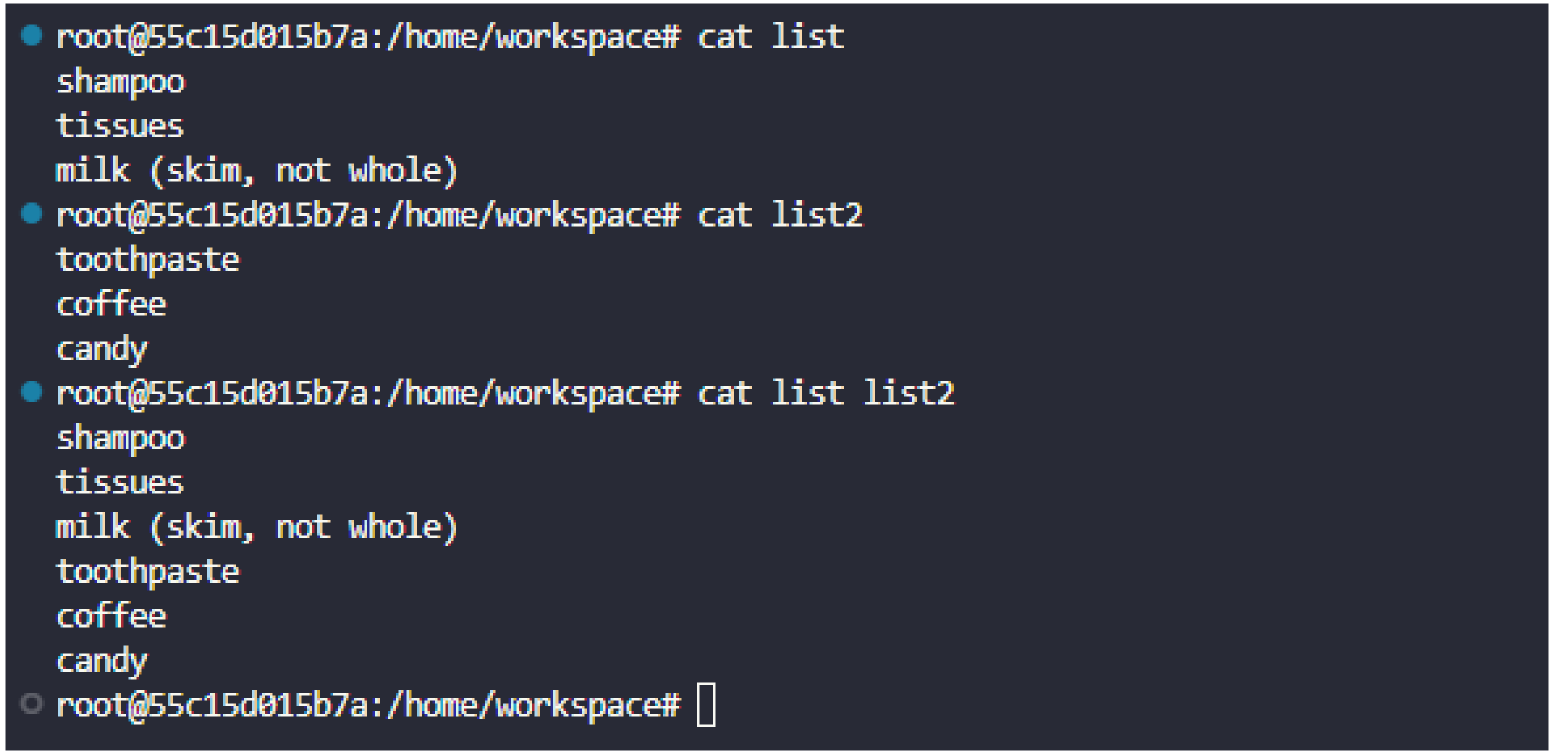
Commands and Arguments



```
root@55c15d015b7a:/home/workspace# ls
a  b  c  code.py  hello_world.py  move
root@55c15d015b7a:/home/workspace# mv a move
root@55c15d015b7a:/home/workspace# cd move
root@55c15d015b7a:/home/workspace/move# ls
a  b  c  d  e  f  g
root@55c15d015b7a:/home/workspace/move# 
```

- **mv [options] source dest :** move files and directories

Commands and Arguments - Strings

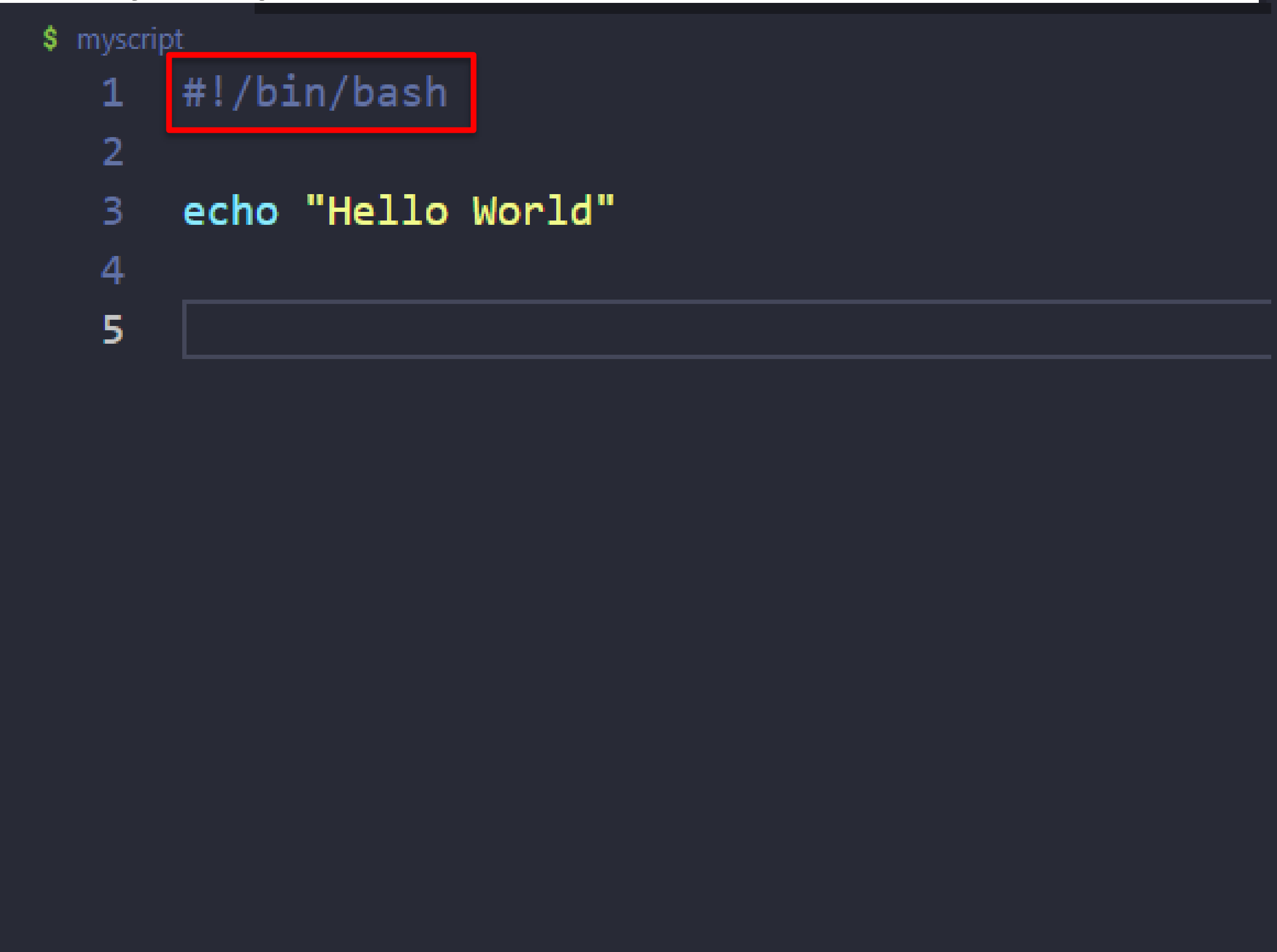


```
root@55c15d015b7a:/home/workspace# cat list
shampoo
tissues
milk (skim, not whole)
root@55c15d015b7a:/home/workspace# cat list2
toothpaste
coffee
candy
root@55c15d015b7a:/home/workspace# cat list list2
shampoo
tissues
milk (skim, not whole)
toothpaste
coffee
candy
root@55c15d015b7a:/home/workspace# 
```

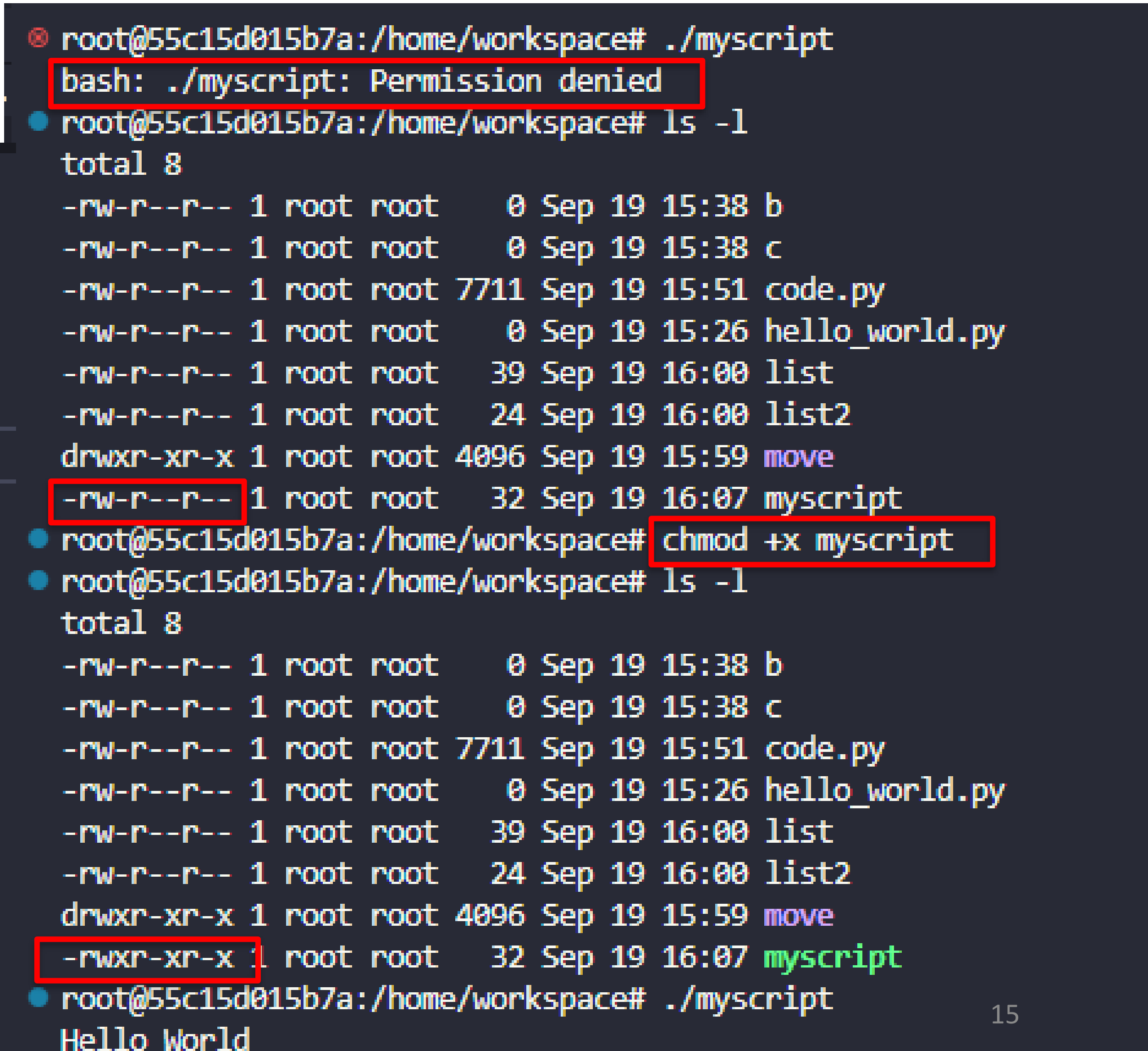
- **cat:** Concatenate and print the contents of a file

Commands and Arguments - Scripts

$ myscript



```
$ myscript
#!/bin/bash

echo "Hello World"

```



```
root@55c15d015b7a:/home/workspace# ./myscript
bash: ./myscript: Permission denied
root@55c15d015b7a:/home/workspace# ls -l
total 8
-rw-r--r-- 1 root root    0 Sep 19 15:38 b
-rw-r--r-- 1 root root    0 Sep 19 15:38 c
-rw-r--r-- 1 root root 7711 Sep 19 15:51 code.py
-rw-r--r-- 1 root root    0 Sep 19 15:26 hello_world.py
-rw-r--r-- 1 root root   39 Sep 19 16:00 list
-rw-r--r-- 1 root root   24 Sep 19 16:00 list2
drwxr-xr-x 1 root root 4096 Sep 19 15:59 move
-rw-r--r-- 1 root root   32 Sep 19 16:07 myscript
root@55c15d015b7a:/home/workspace# chmod +x myscript
root@55c15d015b7a:/home/workspace# ls -l
total 8
-rw-r--r-- 1 root root    0 Sep 19 15:38 b
-rw-r--r-- 1 root root    0 Sep 19 15:38 c
-rw-r--r-- 1 root root 7711 Sep 19 15:51 code.py
-rw-r--r-- 1 root root    0 Sep 19 15:26 hello_world.py
-rw-r--r-- 1 root root   39 Sep 19 16:00 list
-rw-r--r-- 1 root root   24 Sep 19 16:00 list2
drwxr-xr-x 1 root root 4096 Sep 19 15:59 move
-rwxr-xr-x 1 root root   32 Sep 19 16:07 myscript
root@55c15d015b7a:/home/workspace# ./myscript
Hello World
```

Special Characters

Char	Description
$	Expansion – introduces various types of expansion: parameter expansion
‘ ‘	Signle quotes – protect the text inside them so that it has a literal meaning
“ “	Double quotes – protect the text inside them from being split into multiple words or arguments
\	Escape – (backslash) prevents the next character from being interpreted as a special character
[[ ]]	Test – an evaluation of a conditional expression to determine whether it is “true” or “false”
!	Negate – used to negate or reverse a test or exit status
>, >>, <	Redirection – redirect a command’s output or input to a file
|	Pipe – send the output from one command to the input of another command
*, ?	Globs – “wildcard” characters which match parts of filenames (e.g. ls *.txt)

Variables and Special Parameters

- Variables vs Special parameters
  - variables: parameters that you can create and update yourself
  - special parameters: parameters that read-only, pre-set by BASH, and used to communicate some type of internal status.

- Variable naming conventions
  - Variable names should start with a letter or an underscore(**_**)
  - Variable names can contain letters, numbers, and underscores(**_**)
  - Variable names are case-sensitive
  - Variable names should not contain spaces or special characters
  - Use descriptive names that reflect the purpose of the variable
  - Avoid using reserved keywords, such as ***if, then, else, fi,*** and so on as variable names

Variables

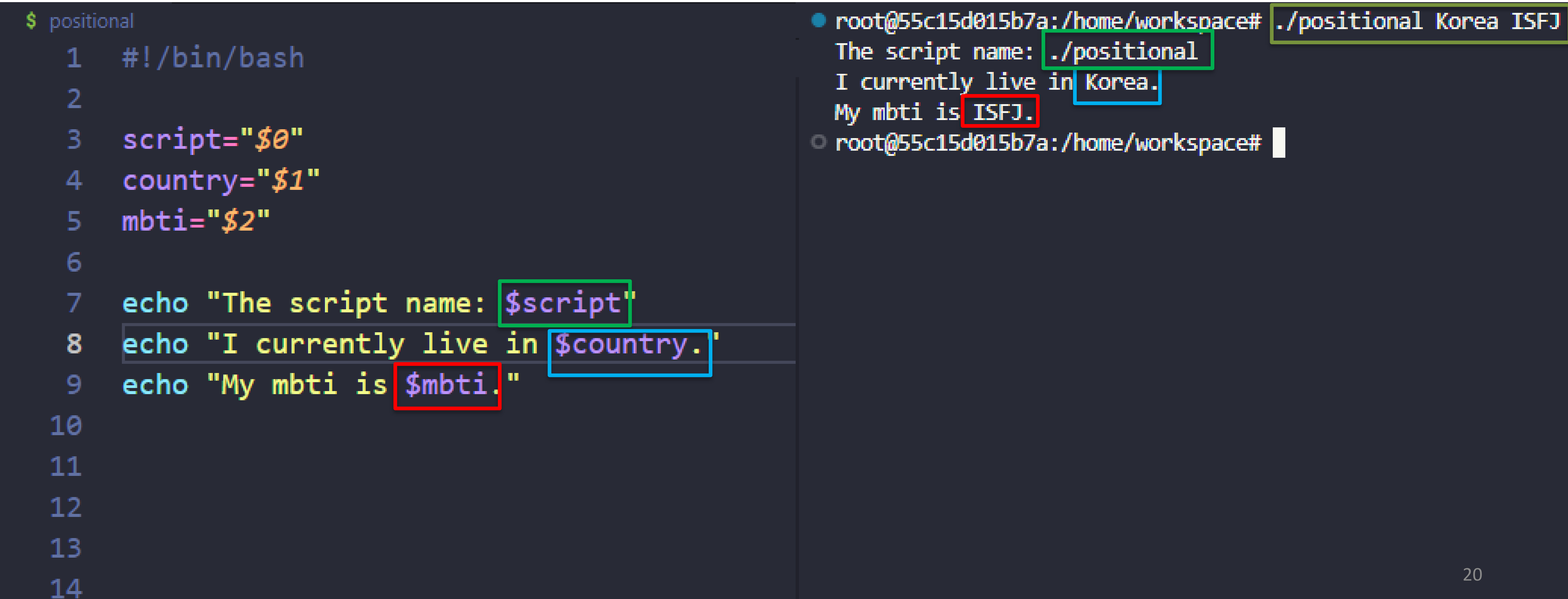
- Valid variable names in Bash:
  - name
  - count
  - _var
  - myVar
  - MY_VAR
- Invalid variable names:
  - 2ndvar (variable name starts with a number)
  - my var (variable name contains a space)
  - my-var (variable name contains a hyphen)

Special Parameters

Parameter Name	Usage	Description
0	“$0”	Contains the name, or the path, of the script. This is not always reliable
1 2 etc.	“$1” etc.	Positional Parameters contain the arguments that were passed to the current script or function
*****	“$*”	Expands to all the words of all the positional parameters. Double quoted, it expands to a single string containing them all, separated by the first character of the IFS variable
@	“$@”	Expands to all the words of all the positional parameters. Double quoted, it expands to a list of them all as individual words
#	$#	Expands to the number of positional parameters
?	$?	Expands to the exit code of the most recently completed foreground command
$	$$	Expands to the PID (process ID number) of the current shell
!	$!	Expands to the PID of the command most recently executed in the background
_	“$_”	Expands to the last argument of the last command that was executed

Special Parameters

$ positional



```
$ positional
#!/bin/bash

script="$0"
country="$1"
mbti="$2"

echo "The script name: $script"
echo "I currently live in $country."
echo "My mbti is $mbti."
```

```
root@55c15d015b7a:/home/workspace# ./positional Korea ISFJ
The script name: ./positional
I currently live in Korea.
My mbti is ISFJ.
root@55c15d015b7a:/home/workspace#
```

Patterns

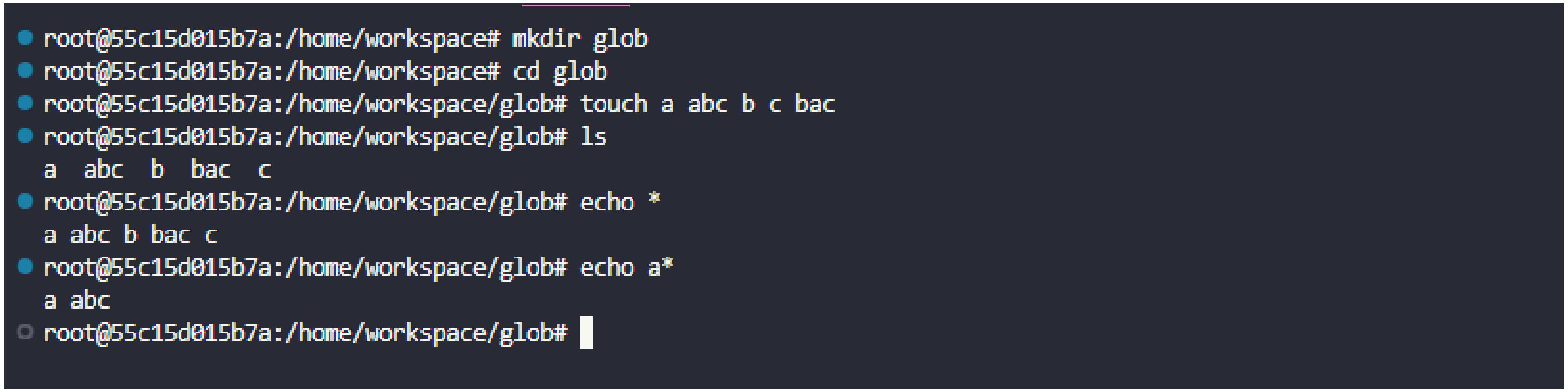
- A pattern is a string with a special format designed to match filenames, or to check, classify or validate data strings

- Bash offers three different kinds of *pattern matching*
  - *Glob Patterns*
  - *Extended Globs*
  - *Regular Expression*

Glob Patterns

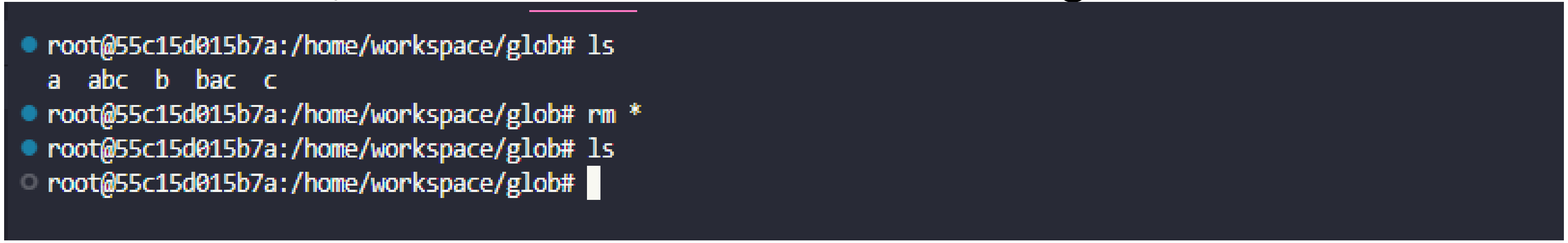
- Globs are basically patterns that can be used to match filenames or other strings
- Globs are composed of normal characters and metacharacters; these are the metacharacters that can be used in globs:
  - *: Matches any string, including the null string
  - ?: Matches any single character
  - […]: Matches any one of the enclosed characters

Glob Patterns



```
● root@55c15d015b7a:/home/workspace# mkdir glob
● root@55c15d015b7a:/home/workspace# cd glob
● root@55c15d015b7a:/home/workspace/glob# touch a abc b c bac
● root@55c15d015b7a:/home/workspace/glob# ls
  a  abc  b  bac  c
● root@55c15d015b7a:/home/workspace/glob# echo *
  a abc b bac c
● root@55c15d015b7a:/home/workspace/glob# echo a*
  a abc
○ root@55c15d015b7a:/home/workspace/glob#
```

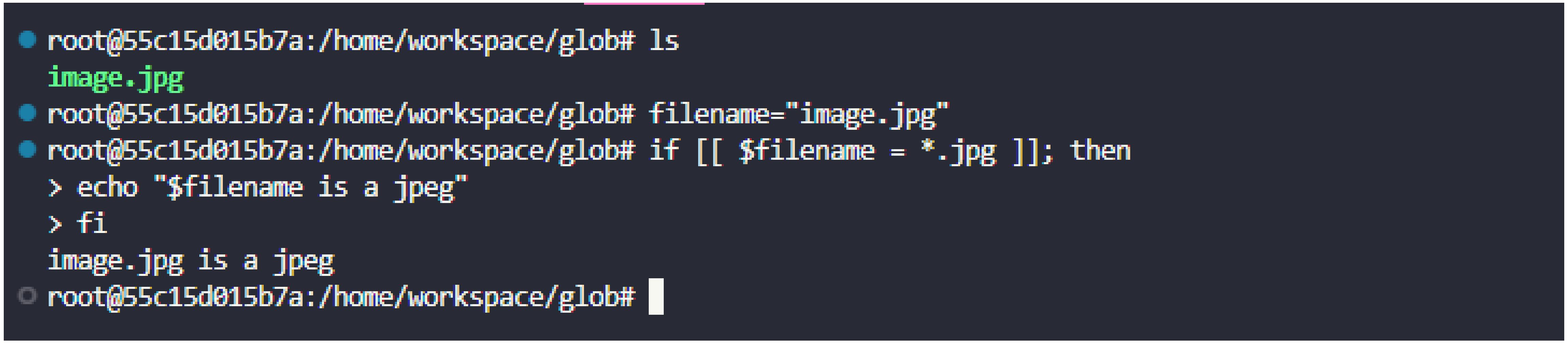
- For instance, ‘echo a*’ has the same meaning with ‘echo a abc’



```
● root@55c15d015b7a:/home/workspace/glob# ls
  a  abc  b  bac  c
● root@55c15d015b7a:/home/workspace/glob# rm *
● root@55c15d015b7a:/home/workspace/glob# ls
○ root@55c15d015b7a:/home/workspace/glob#
```

- Here, the filenames will be passed as a single argument to **rm**

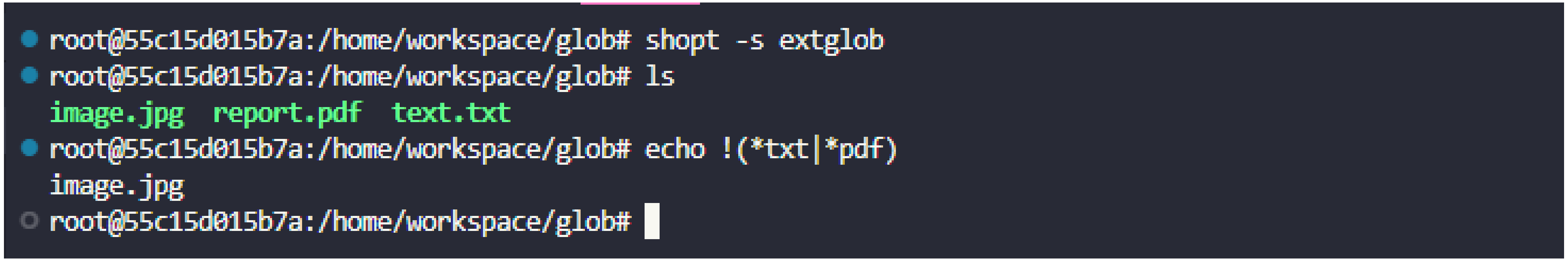
Glob Patterns



```
root@55c15d015b7a:/home/workspace/glob# ls
image.jpg
root@55c15d015b7a:/home/workspace/glob# filename="image.jpg"
root@55c15d015b7a:/home/workspace/glob# if [[ $filename = *.jpg ]]; then
> echo "$filename is a jpeg"
> fi
image.jpg is a jpeg
root@55c15d015b7a:/home/workspace/glob# 
```

- Globs may also be used to check whether data matches a specific format
- The **[[** keyword and the **case** keyword both offer the opportunity to check a string against a glob, either regular globs, or extended globs, if the latter have been enabled

Extended Globs (Optional)



```
● root@55c15d015b7a:/home/workspace/glob# shopt -s extglob
● root@55c15d015b7a:/home/workspace/glob# ls
  image.jpg  report.pdf  text.txt
● root@55c15d015b7a:/home/workspace/glob# echo !(*txt|*pdf)
  image.jpg
○ root@55c15d015b7a:/home/workspace/glob# 
```

- Extended Globs are more powerful in nature; they are equivalent to regular expression
- To use this feature, command ‘shopt –s extglob’

?(list)	Matches zero or one occurrence of the given patterns
*(list)	Matches zero or more occurrences of the given patterns
+(list)	Matches one or more occurrences of the vien patterns
@(list)	Matches one of the given patterns
!(list)	Matches anything but the given patterns

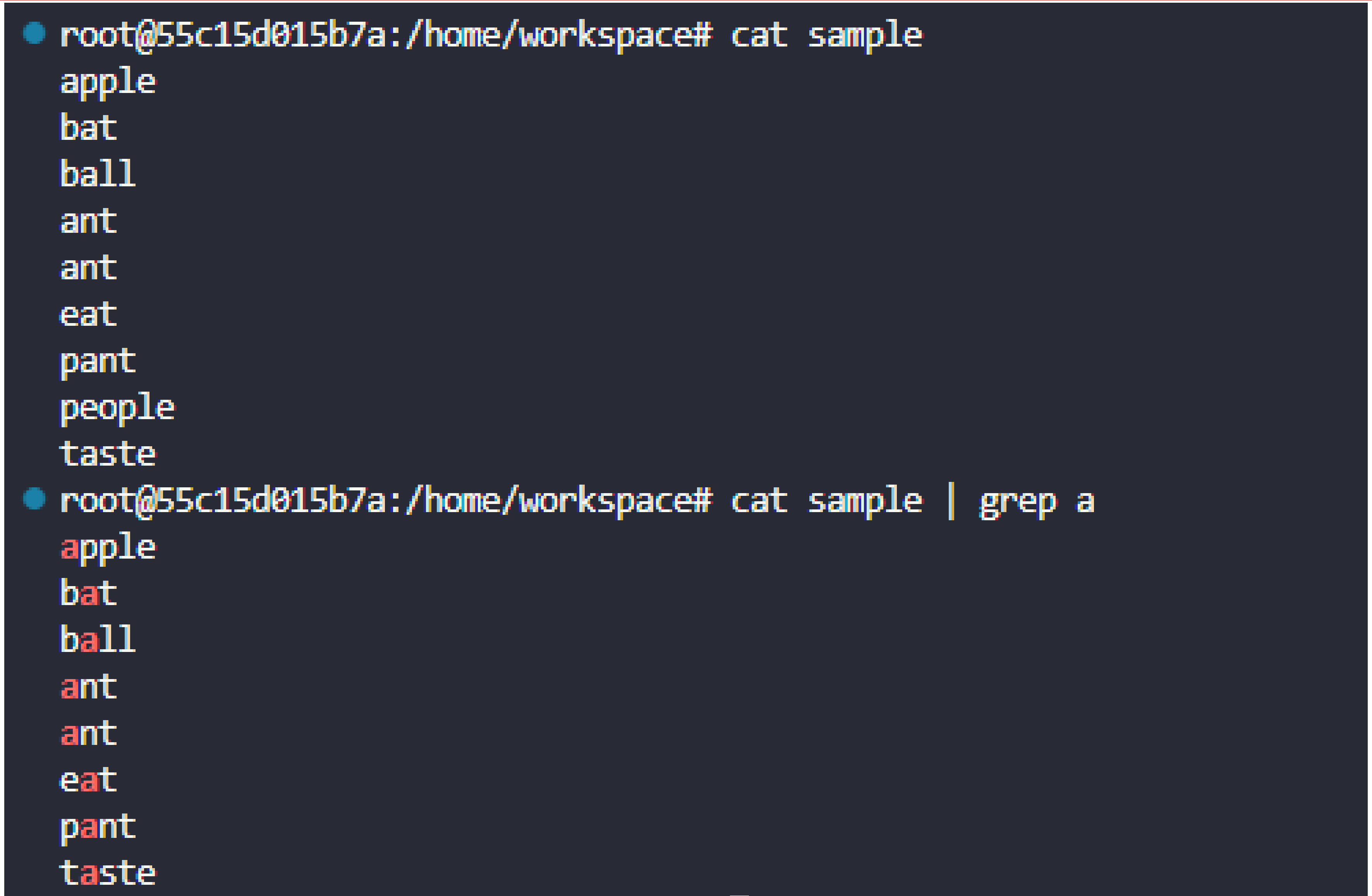
Regular Expressions (Optional)

- Regular expression (regex) are similar to *Glob Patterns*, but they can only be used for pattern matching, not for filename matching

[Reading material]

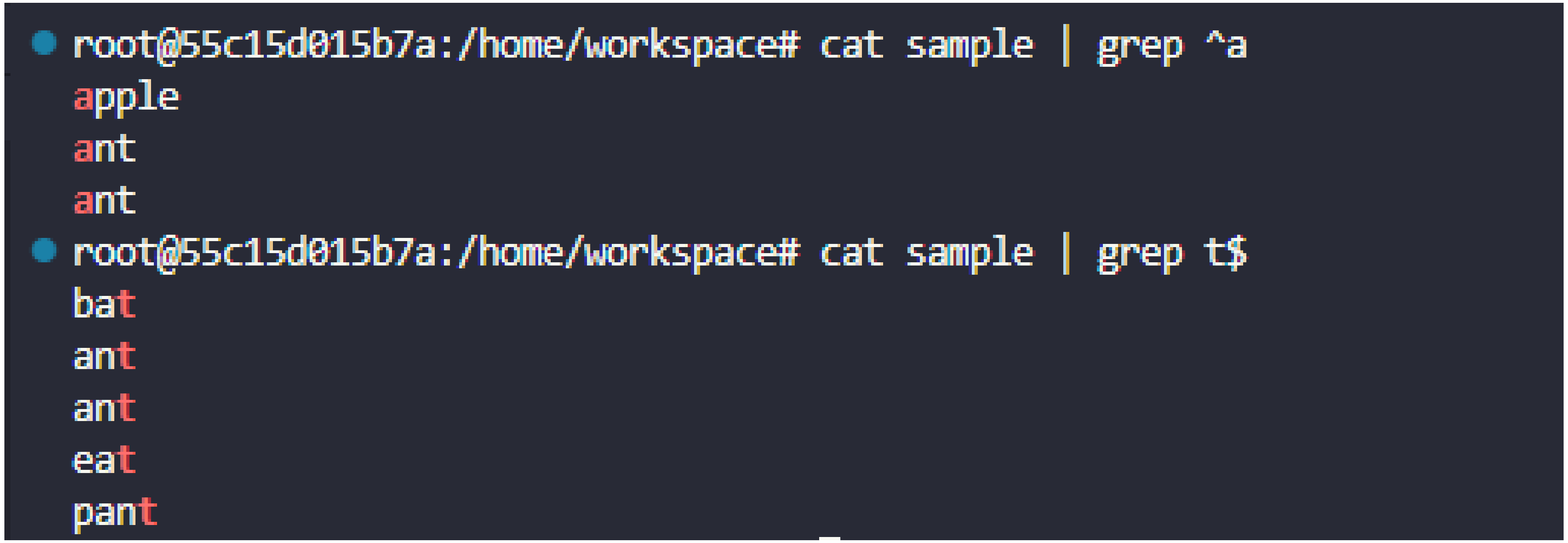
http://mywiki.wooledge.org/RegularExpression

Regular Expressions (Optional)



```
root@55c15d015b7a:/home/workspace# cat sample
apple
bat
ball
ant
ant
eat
pant
people
taste
root@55c15d015b7a:/home/workspace# cat sample | grep a
apple
bat
ball
ant
ant
eat
pant
taste
```

Regular Expressions (Optional)



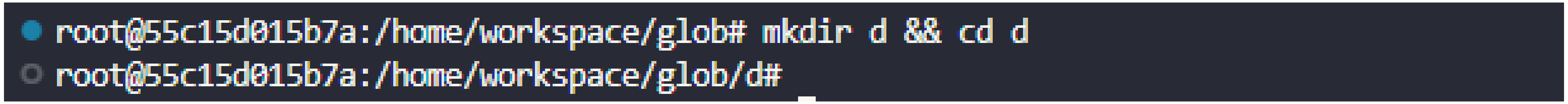
```
● root@55c15d015b7a:/home/workspace# cat sample | grep ^a
  apple
  ant
  ant
● root@55c15d015b7a:/home/workspace# cat sample | grep t$
  bat
  ant
  ant
  eat
  pant
```

Tests and Conditionals – Exit Status

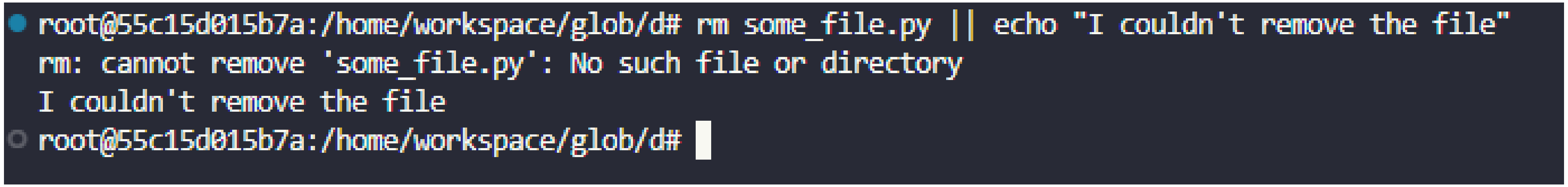
- Every command results in an exit code whenever it terminates
- The exit code is like a return value from functions (0 – 255)
- Convention dictates that we use 0 to denote success, and any other number to denote failure of some sort
- The specific number is entirely application-specific, and is used to hint as to what exactly went wrong

Tests and Conditionals - Control Operators (&& and ||)

- Control Operators are '&&' and '||', which respectively represent a logical AND and a logical OR
- They are used to control whether the second command should be executed depending on the success of the first *(conditional execution)*



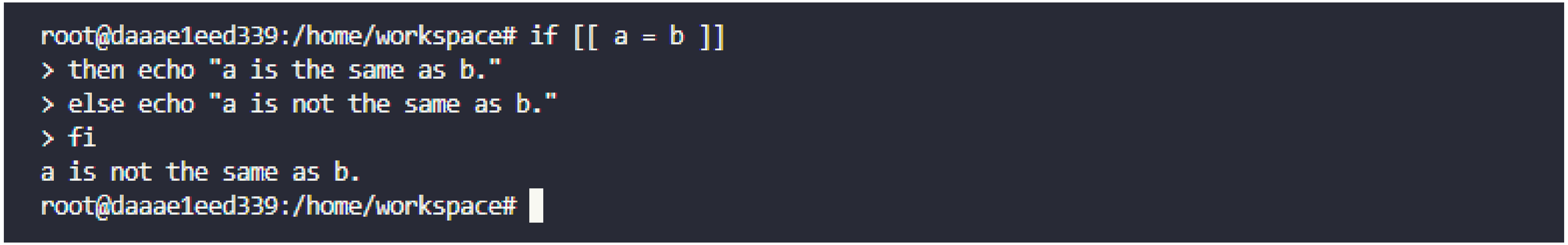
```
root@55c15d015b7a:/home/workspace/glob# mkdir d && cd d
root@55c15d015b7a:/home/workspace/glob/d#
```



```
root@55c15d015b7a:/home/workspace/glob/d# rm some_file.py || echo "I couldn't remove the file"
rm: cannot remove 'some_file.py': No such file or directory
I couldn't remove the file
root@55c15d015b7a:/home/workspace/glob/d#
```

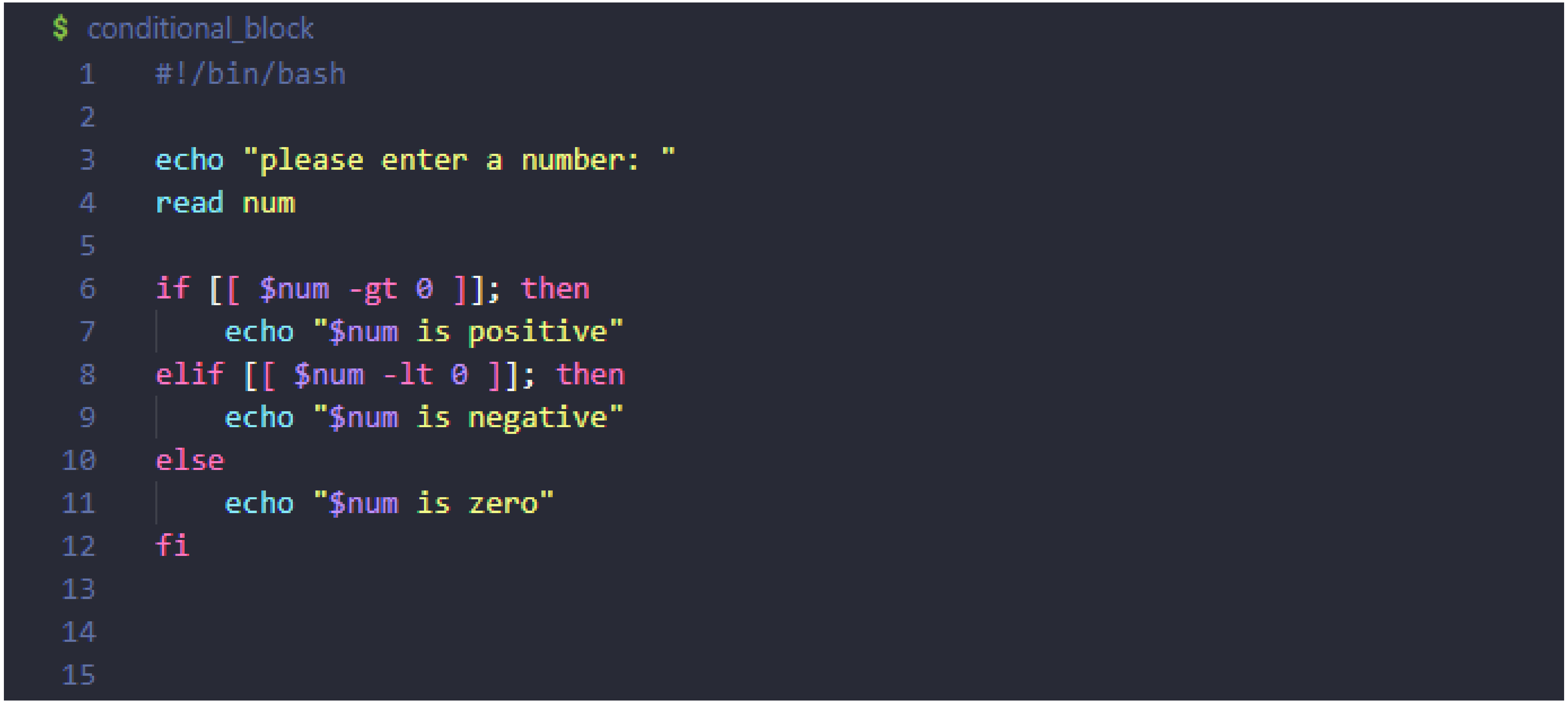
Tests and Conditionals – Conditional Blocks

- **if** is a shell keyword that executes a command, and checks that command’s exit code to see whether it was successful



```
root@daaae1eed339:/home/workspace# if [[ a = b ]]
> then echo "a is the same as b."
> else echo "a is not the same as b."
> fi
a is not the same as b.
root@daaae1eed339:/home/workspace# 
```

Tests and Conditionals – Conditional Blocks



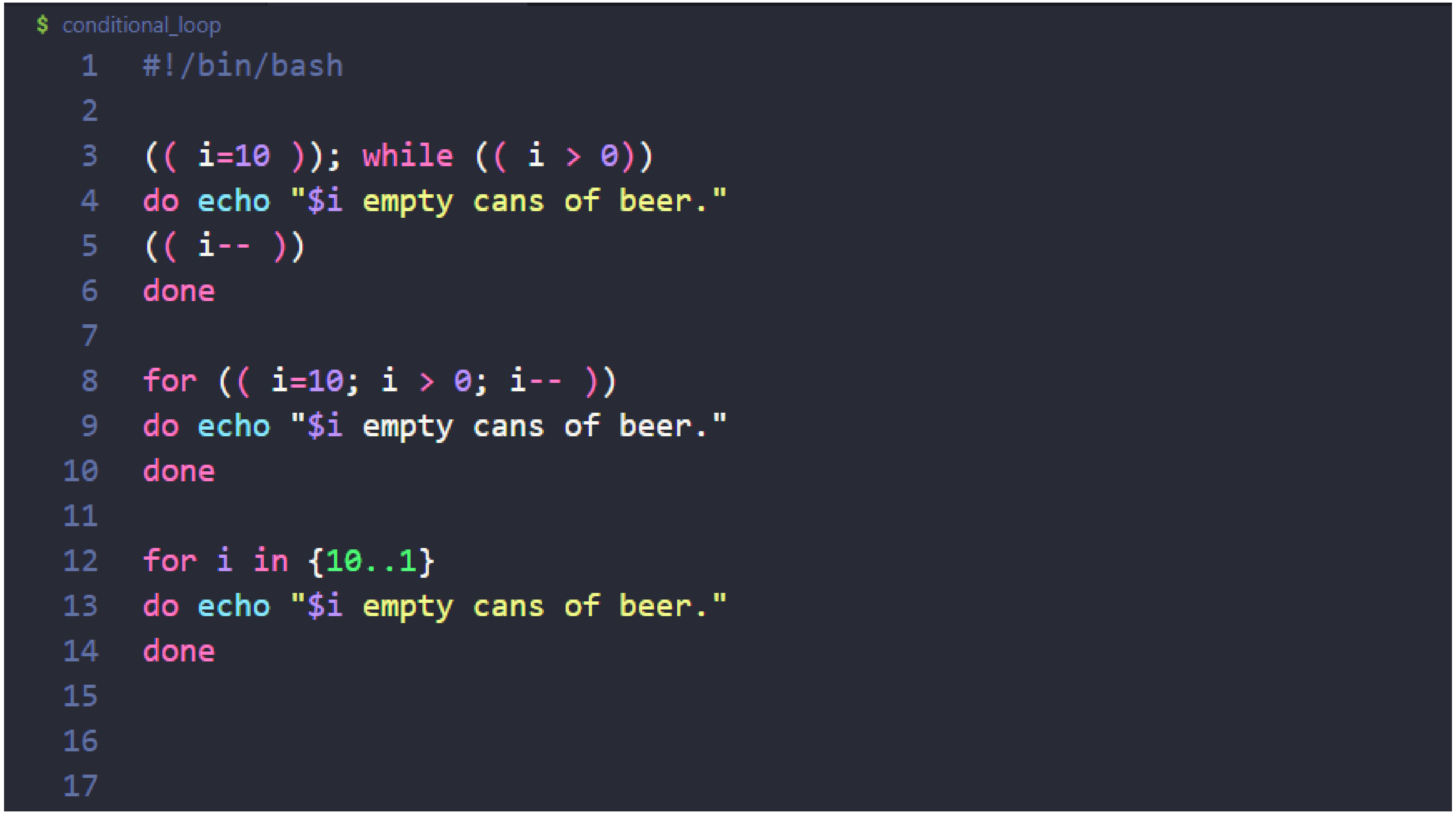
$ conditional_block

```bash
#!/bin/bash

echo "please enter a number: "
read num

if [[ $num -gt 0 ]]; then
    echo "$num is positive"
elif [[ $num -lt 0 ]]; then
    echo "$num is negative"
else
    echo "$num is zero"
fi



```

Tests and Conditionals – Conditional Loops

- **while** *command*: Repeat so long as command is executed successfully
- **until** *command*: Repeat so long as command is executed unsuccessfully
- **for** *variable* **in** *words*: Repeat the loop for each word, setting variable to each word in turn
- **for** ((*expression*; *expression*; *expression*)): Starts by evaluating the first arithmetic expression; repeats the loop so long as the second arithmetic expression is successful

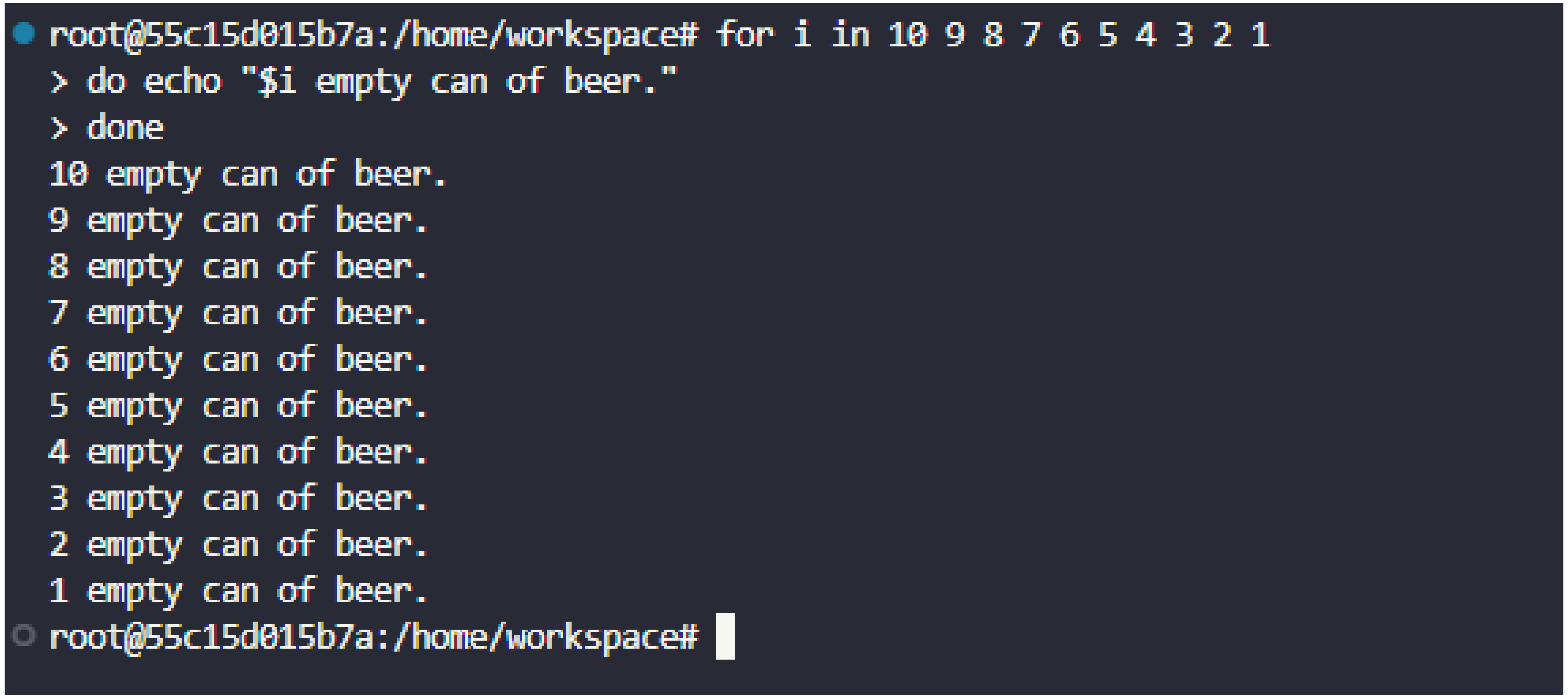
Tests and Conditionals – Conditional Loops



$ conditional_loop

```bash
#!/bin/bash

(( i=10 )); while (( i > 0))
do echo "$i empty cans of beer."
(( i-- ))
done

for (( i=10; i > 0; i-- ))
do echo "$i empty cans of beer."
done

for i in {10..1}
do echo "$i empty cans of beer."
done



```

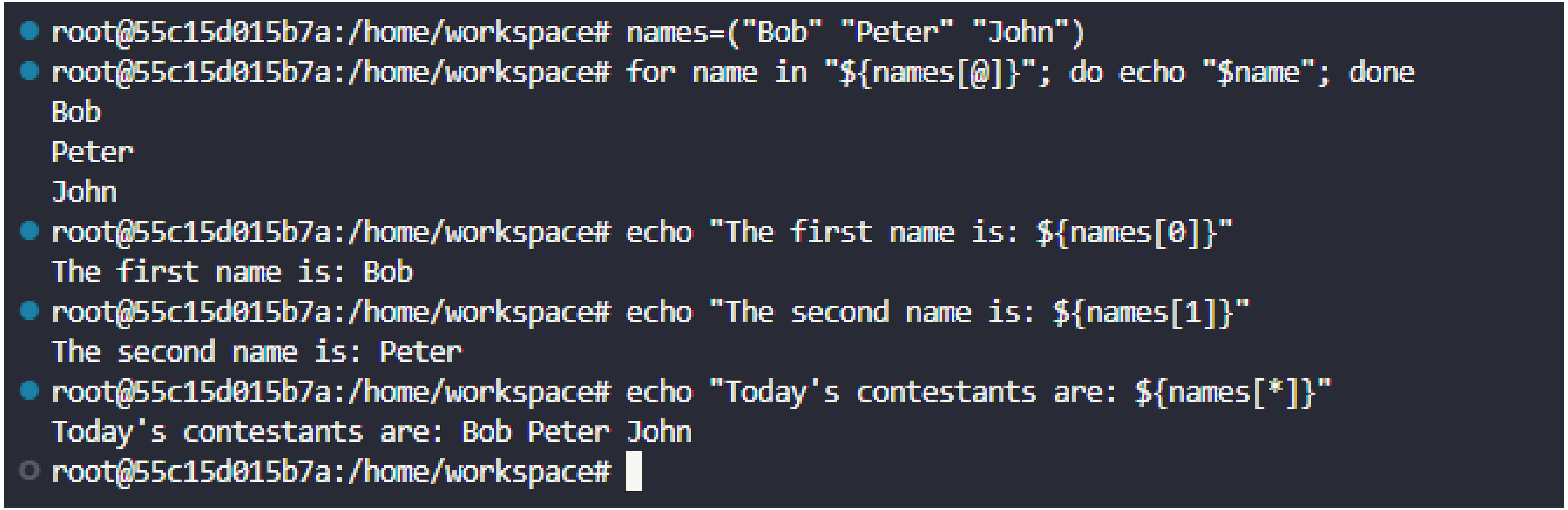
Tests and Conditionals – Conditional Loops



```
root@55c15d015b7a:/home/workspace# for i in 10 9 8 7 6 5 4 3 2 1
> do echo "$i empty can of beer."
> done
10 empty can of beer.
9 empty can of beer.
8 empty can of beer.
7 empty can of beer.
6 empty can of beer.
5 empty can of beer.
4 empty can of beer.
3 empty can of beer.
2 empty can of beer.
1 empty can of beer.
root@55c15d015b7a:/home/workspace#
```

- Bash takes the characters between **in** and the end of the line, and splits them up into words

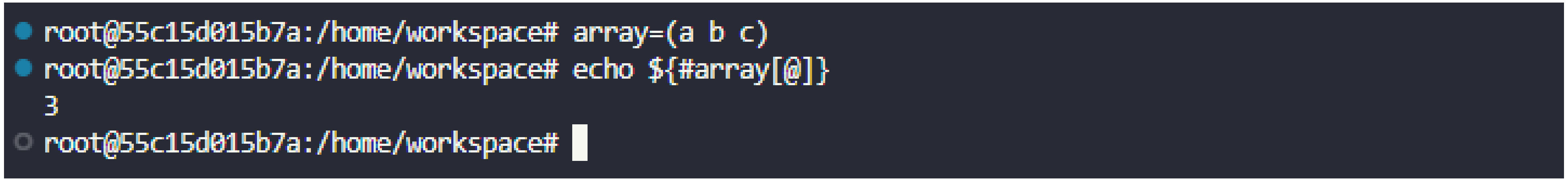
Arrays



```
root@55c15d015b7a:/home/workspace# names=("Bob" "Peter" "John")
root@55c15d015b7a:/home/workspace# for name in "${names[@]}"; do echo "$name"; done
Bob
Peter
John
root@55c15d015b7a:/home/workspace# echo "The first name is: ${names[0]}"
The first name is: Bob
root@55c15d015b7a:/home/workspace# echo "The second name is: ${names[1]}"
The second name is: Peter
root@55c15d015b7a:/home/workspace# echo "Today's contestants are: ${names[*]}"
Today's contestants are: Bob Peter John
root@55c15d015b7a:/home/workspace# 
```

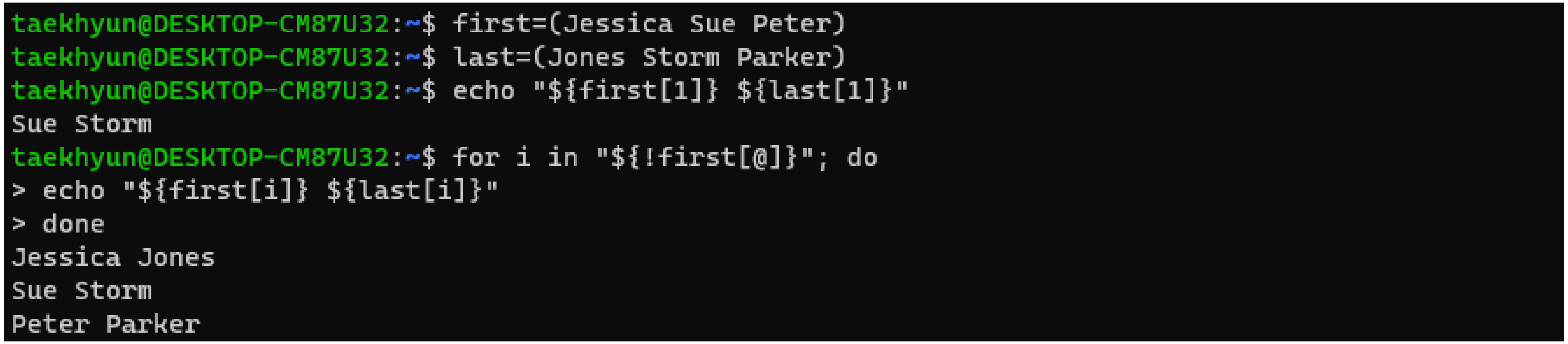
- Several ways you can create or fill your array with data
  - The easiest way to create a simple array with data is by using the **=()** syntax

Arrays



```
● root@55c15d015b7a:/home/workspace# array=(a b c)
● root@55c15d015b7a:/home/workspace# echo ${#array[@]}
  3
○ root@55c15d015b7a:/home/workspace# 
```

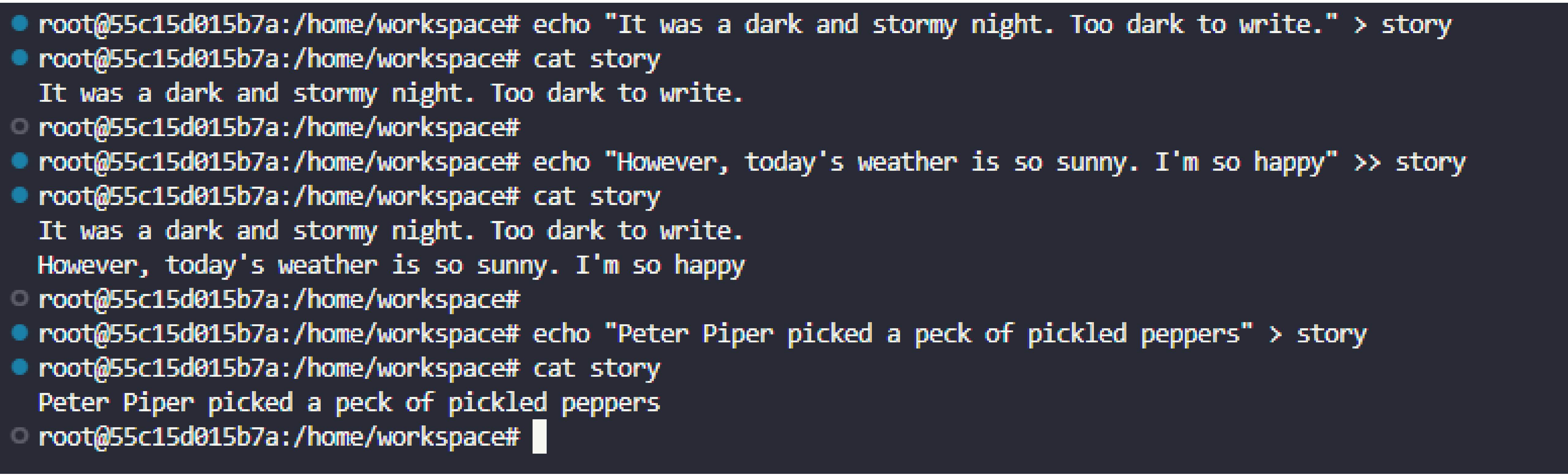
- You can get the number of elements of an array by using ${#array[@]}



```
taekhyun@DESKTOP-CM87U32:~$ first=(Jessica Sue Peter)
taekhyun@DESKTOP-CM87U32:~$ last=(Jones Storm Parker)
taekhyun@DESKTOP-CM87U32:~$ echo "${first[1]} ${last[1]}"
Sue Storm
taekhyun@DESKTOP-CM87U32:~$ for i in "${!first[@]}"; do
> echo "${first[i]} ${last[i]}"
> done
Jessica Jones
Sue Storm
Peter Parker
```

- You can loop over the indices of one of the arrays, and then use that same index in both arrays together

Input And Output – Redirection



```
root@55c15d015b7a:/home/workspace# echo "It was a dark and stormy night. Too dark to write." > story
root@55c15d015b7a:/home/workspace# cat story
It was a dark and stormy night. Too dark to write.
root@55c15d015b7a:/home/workspace#
root@55c15d015b7a:/home/workspace# echo "However, today's weather is so sunny. I'm so happy" >> story
root@55c15d015b7a:/home/workspace# cat story
It was a dark and stormy night. Too dark to write.
However, today's weather is so sunny. I'm so happy
root@55c15d015b7a:/home/workspace#
root@55c15d015b7a:/home/workspace# echo "Peter Piper picked a peck of pickled peppers" > story
root@55c15d015b7a:/home/workspace# cat story
Peter Piper picked a peck of pickled peppers
root@55c15d015b7a:/home/workspace#
```

- The most basic form of input/output manipulation in BASH
- You can send output to a file instead of the terminal, or have an application read from a file instead of from the keyboard