BASH
THE BOURNE-AGAIN SHELL

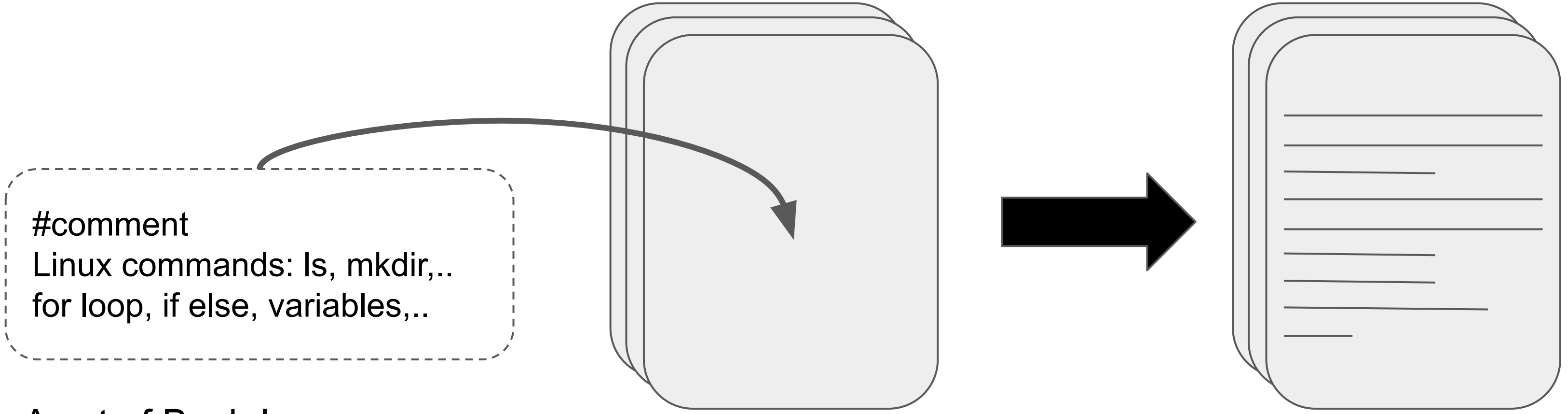
Introduction to basic Shell/Bash script

Nguyen Quang Khai
05/01/2025

Content

1. Overview

What is a Bash script file?



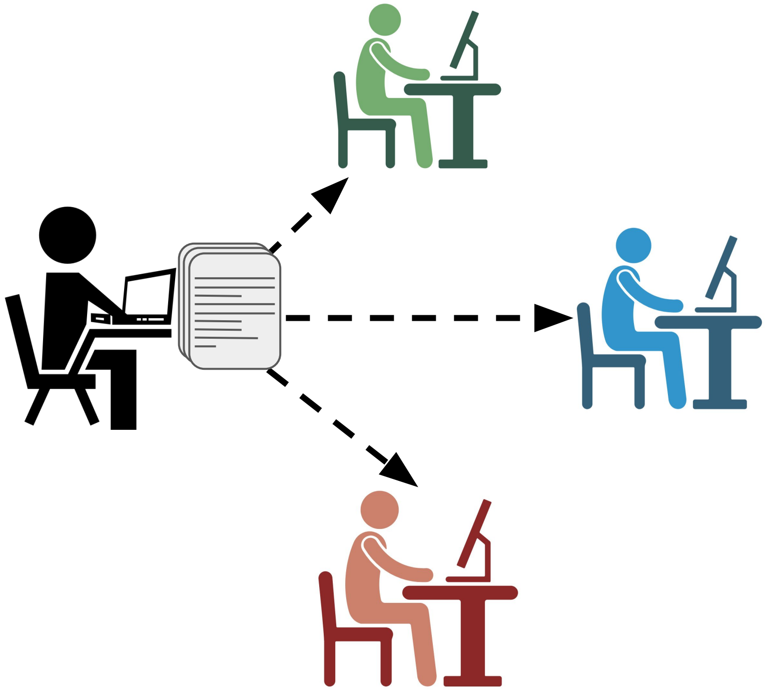
A set of Bash language commands arranged to be executed in order

a blank text file

a Bash script file
(set of commands)

Why need Bash script?

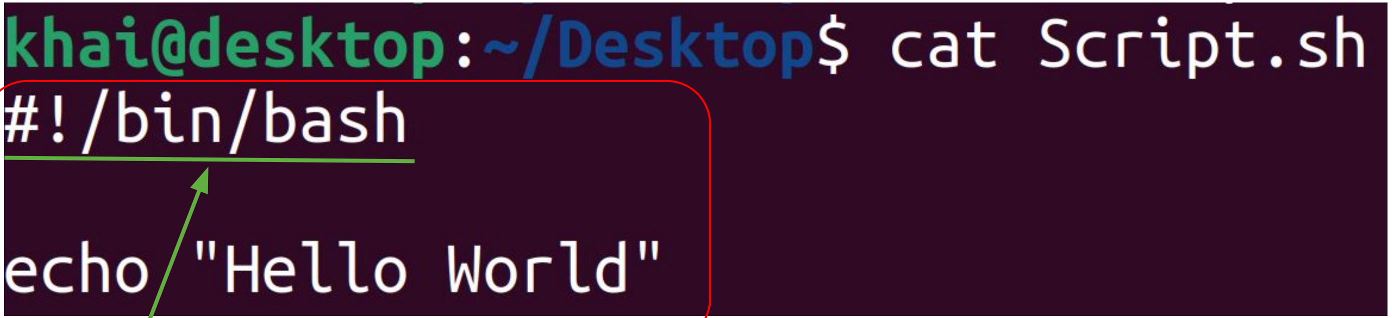
Run the same set of commands many times.



Share “that set of commands” with others.

Run a Bash script file in the terminal

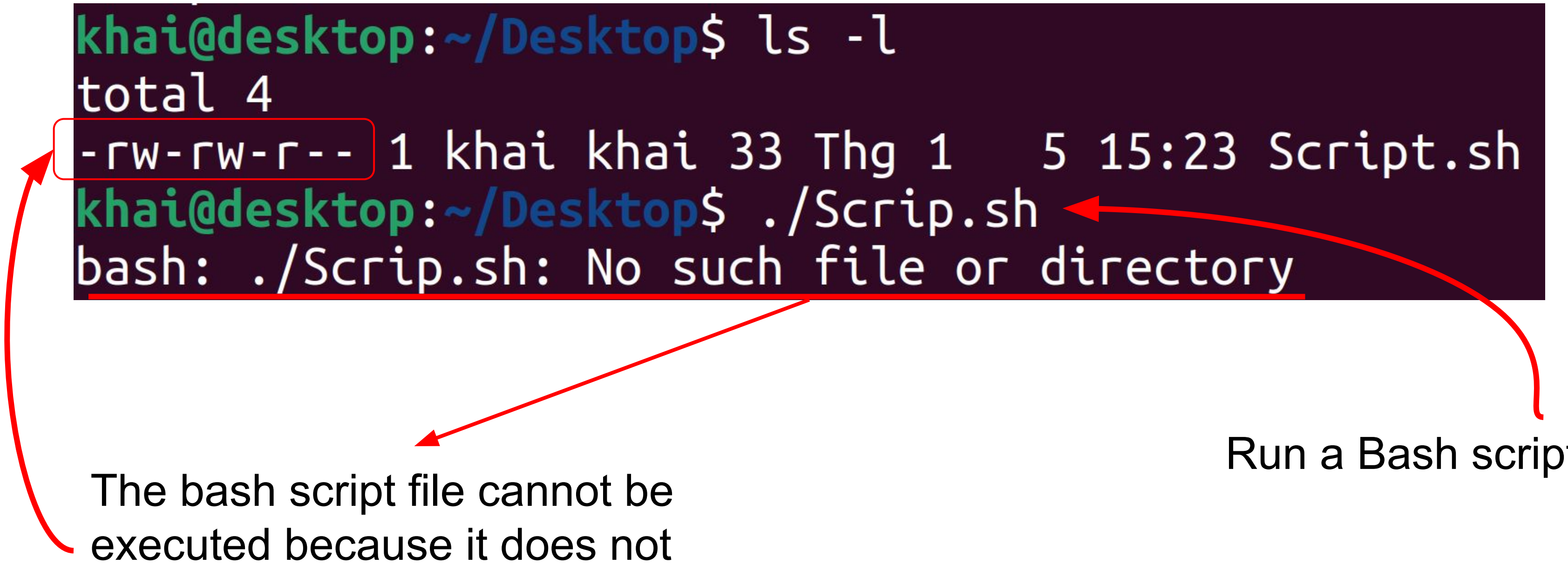
I have a Bash script file with content:



```
khai@desktop:~/Desktop$ cat Script.sh
#!/bin/bash

echo "Hello World"
```

#! called Shebang (This line means that this script file will be executed by Bash.

Run a Bash script file in the terminal



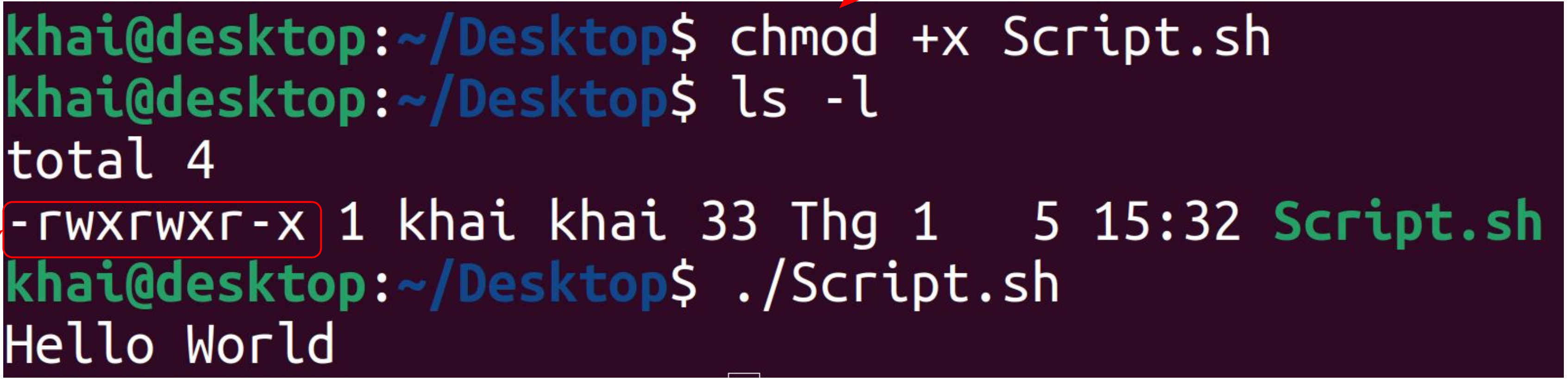
```
khai@desktop:~/Desktop$ ls -l
total 4
-rw-rw-r-- 1 khai khai 33 Thg 1   5 15:23 Script.sh
khai@desktop:~/Desktop$ ./Scrip.sh
bash: ./Scrip.sh: No such file or directory
```

Run a Bash script file

The bash script file cannot be executed because it does not have the required permissions.

Run a Bash script file in the terminal

Grant execution permission to Bash script file using chmod command



```
khai@desktop:~/Desktop$ chmod +x Script.sh
khai@desktop:~/Desktop$ ls -l
total 4
-rwxrwxr-x 1 khai khai 33 Thg 1   5 15:32 Script.sh
khai@desktop:~/Desktop$ ./Script.sh
Hello World
```

✓

Execution permission granted

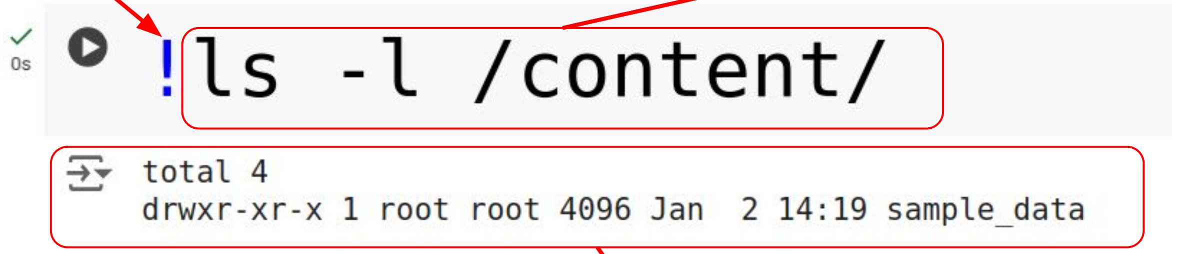
Run Bash script (or Linux commands) in Google Colab

1. Single line Bash script:

Place the ! right before command line

Bash script/Linux command line

Example 1:



```
!ls -l /content/
```

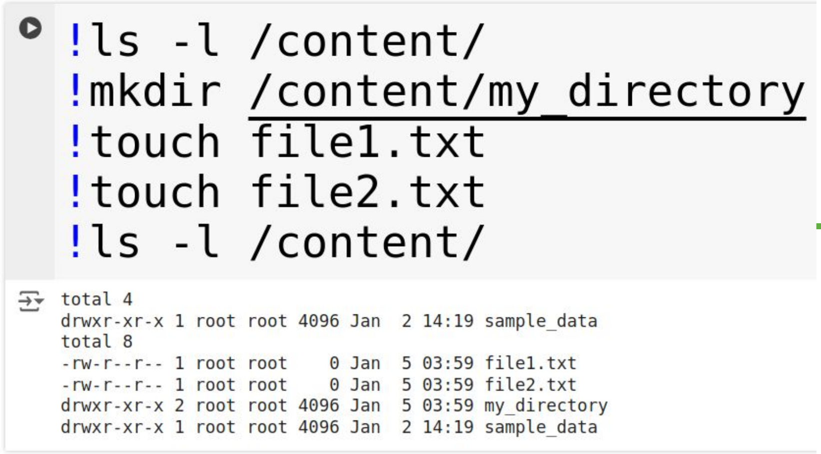
```
total 4
drwxr-xr-x 1 root root 4096 Jan  2 14:19 sample_data
```

Output

Run Bash script (or Linux commands) in Google Colab

2. Multiple line Bash script:

Example 2:

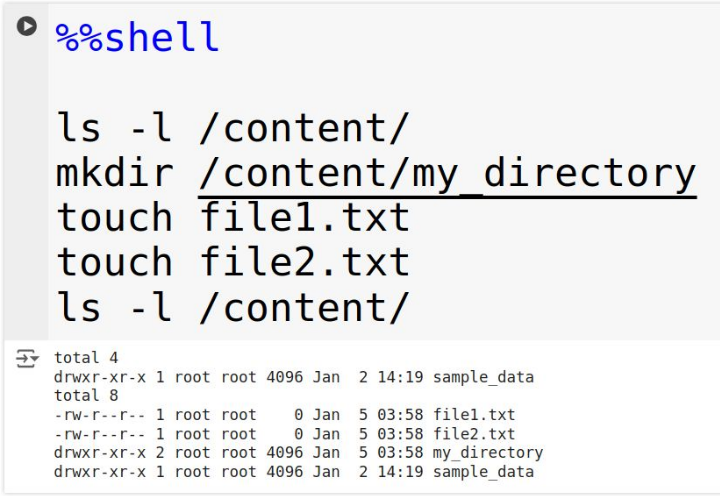


```
!ls -l /content/
!mkdir /content/my_directory
!touch file1.txt
!touch file2.txt
!ls -l /content/
```

```
total 4
drwxr-xr-x 1 root root 4096 Jan  2 14:19 sample_data
total 8
-rw-r--r-- 1 root root    0 Jan  5 03:59 file1.txt
-rw-r--r-- 1 root root    0 Jan  5 03:59 file2.txt
drwxr-xr-x 2 root root 4096 Jan  5 03:59 my_directory
drwxr-xr-x 1 root root 4096 Jan  2 14:19 sample_data
```

Should be like this →

Example 3:



```
%%shell

ls -l /content/
mkdir /content/my_directory
touch file1.txt
touch file2.txt
ls -l /content/
```

```
total 4
drwxr-xr-x 1 root root 4096 Jan  2 14:19 sample_data
total 8
-rw-r--r-- 1 root root    0 Jan  5 03:58 file1.txt
-rw-r--r-- 1 root root    0 Jan  5 03:58 file2.txt
drwxr-xr-x 2 root root 4096 Jan  5 03:58 my_directory
drwxr-xr-x 1 root root 4096 Jan  2 14:19 sample_data
```

- Save time when coding
- Make code easy to read

2. Variable

2. Variables

Basic Syntax: variable_name=value

Some rules:

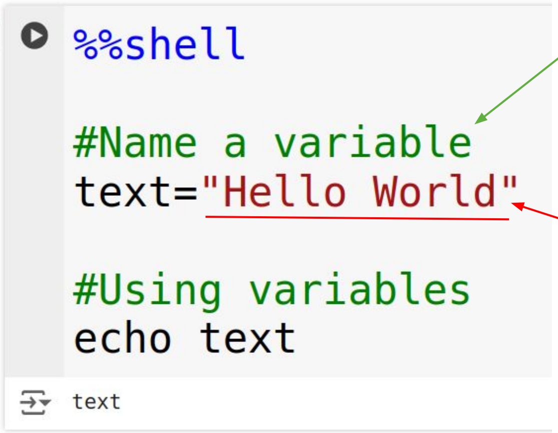
- If you want to name a variable with 2 or more words, there should be an underscore between the words.
- Should contain letters, numbers, and underscores.
- Cannot have spaces around **=**
  (e.g., var = value is invalid).
- Bash variables are case-sensitive (VAR and var are different).

Some rules:

- String:
  - a word
  - multiple words with spaces between words: Place that string in quotes " "
- A number
- Output of a command: put that command in $( ):
  variable_name=$(a command)
- a directory path

2. Variables

Example 4:



```
%%shell

#Name a variable
text="Hello World"

#Using variables
echo text
```

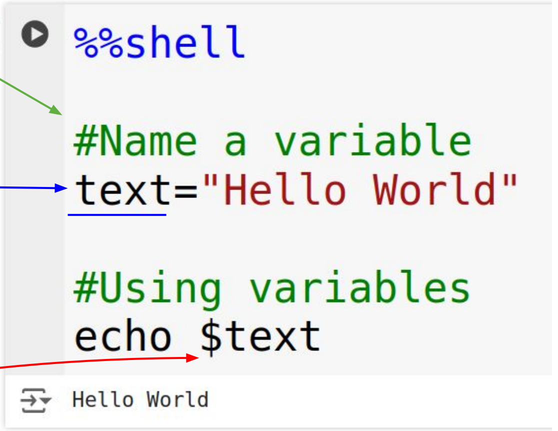
```
text
```

a **comment** (option): Put # at the beginning of the line to tell Bash to ignore this line

variable name

value

Example 5:



```
%%shell

#Name a variable
text="Hello World"

#Using variables
echo $text
```

```
Hello World
```

How to use a variable: Place the $ right before the variable name.

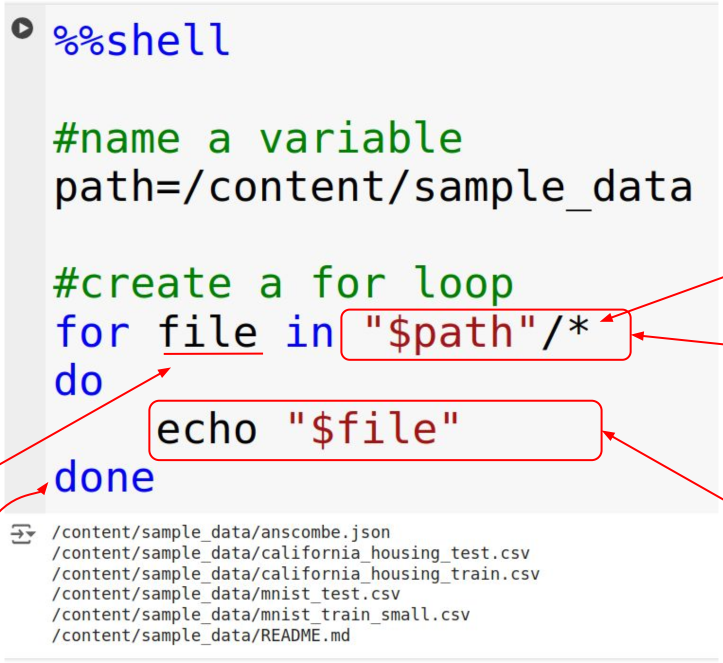
3. For loop

3. For loop

Example 6: Combine variable naming with a for loop

Syntax

```
for     in
do


done
```



```
%%shell

#name a variable
path=/content/sample_data

#create a for loop
for file in "$path"/*
do
    echo "$file"
done
```

```
/content/sample_data/anscombe.json
/content/sample_data/california_housing_test.csv
/content/sample_data/california_housing_train.csv
/content/sample_data/mnist_test.csv
/content/sample_data/mnist_train_small.csv
/content/sample_data/README.md
```

* means anything

a set

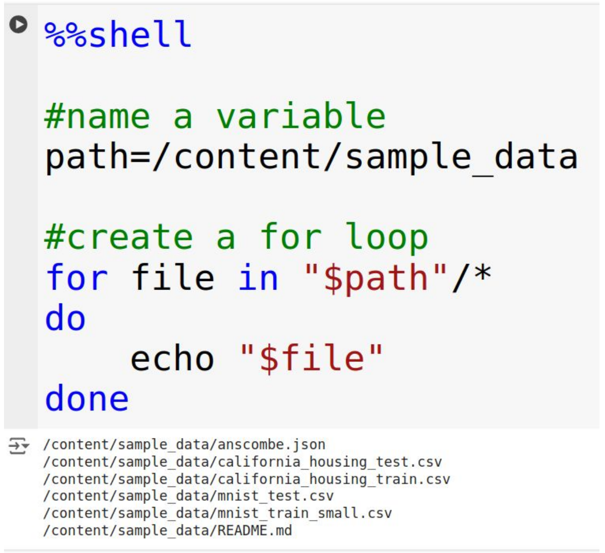
Action to perform for each element

variable name for each element in the set (Can be changed)

When all elements in the set are scanned, the "for loop" is stopped.

3. For loop

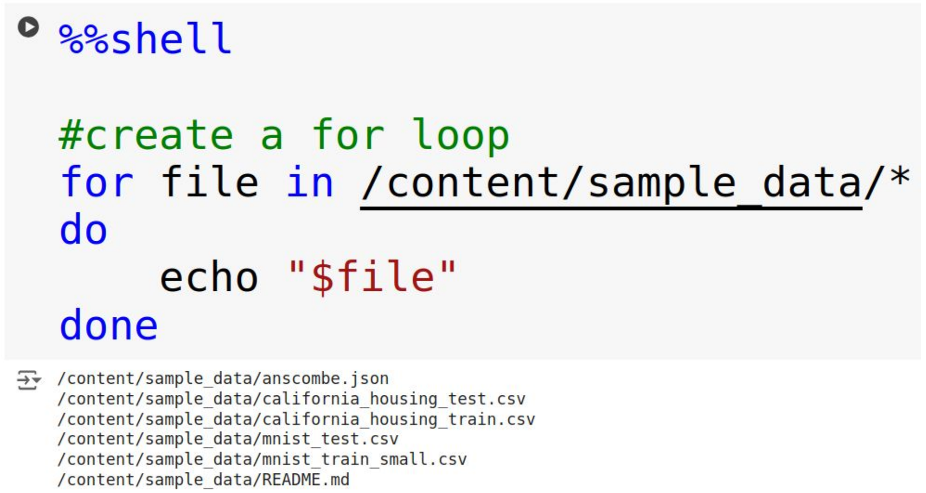
Example 6:



```
%%shell

#name a variable
path=/content/sample_data

#create a for loop
for file in "$path"/*
do
    echo "$file"
done
```

```
/content/sample_data/anscombe.json
/content/sample_data/california_housing_test.csv
/content/sample_data/california_housing_train.csv
/content/sample_data/mnist_test.csv
/content/sample_data/mnist_train_small.csv
/content/sample_data/README.md
```

Example 7:



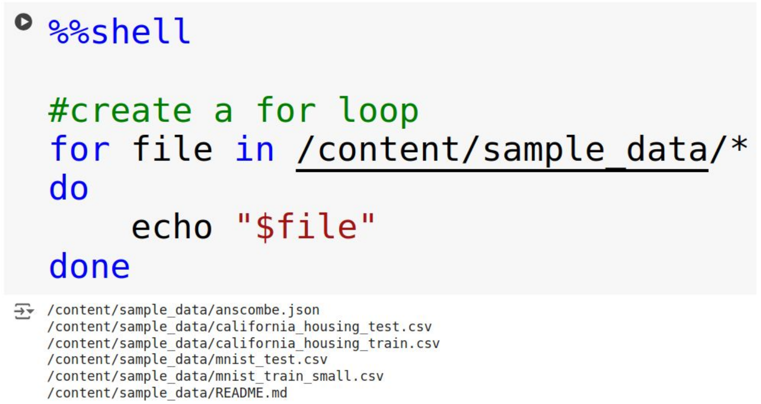
```
%%shell

#create a for loop
for file in /content/sample_data/*
do
    echo "$file"
done
```

```
/content/sample_data/anscombe.json
/content/sample_data/california_housing_test.csv
/content/sample_data/california_housing_train.csv
/content/sample_data/mnist_test.csv
/content/sample_data/mnist_train_small.csv
/content/sample_data/README.md
```

If the "path" variable is only used once in a Bash script, it is not necessary to name the variable.

3. For loop

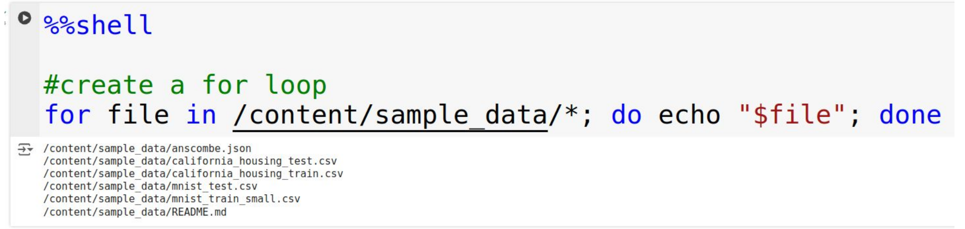
Example 7:



```
%%shell

#create a for loop
for file in /content/sample_data/*
do
    echo "$file"
done
```

```
/content/sample_data/anscombe.json
/content/sample_data/california_housing_test.csv
/content/sample_data/california_housing_train.csv
/content/sample_data/mnist_test.csv
/content/sample_data/mnist_train_small.csv
/content/sample_data/README.md
```

VS



```
%%shell

#create a for loop
for file in /content/sample_data/*; do echo "$file"; done
```

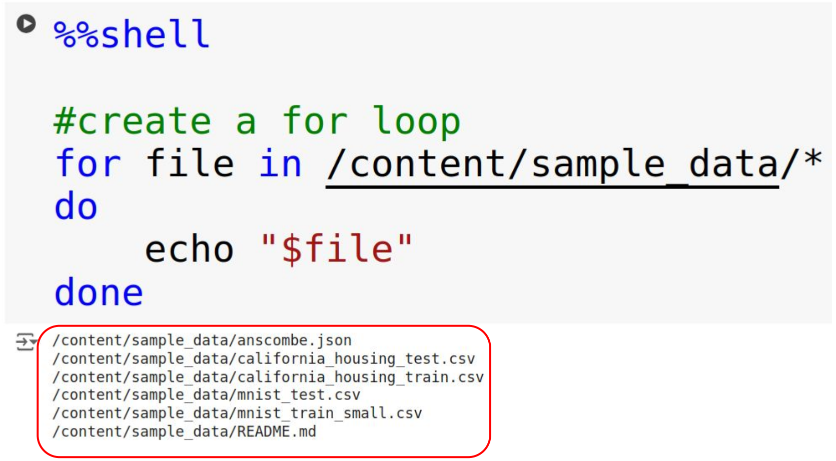
```
/content/sample_data/anscombe.json
/content/sample_data/california_housing_test.csv
/content/sample_data/california_housing_train.csv
/content/sample_data/mnist_test.csv
/content/sample_data/mnist_train_small.csv
/content/sample_data/README.md
```

makes it easy to read quickly when there are many commands

When there are many commands, it will become a very long line of text → difficult to read

3. For loop

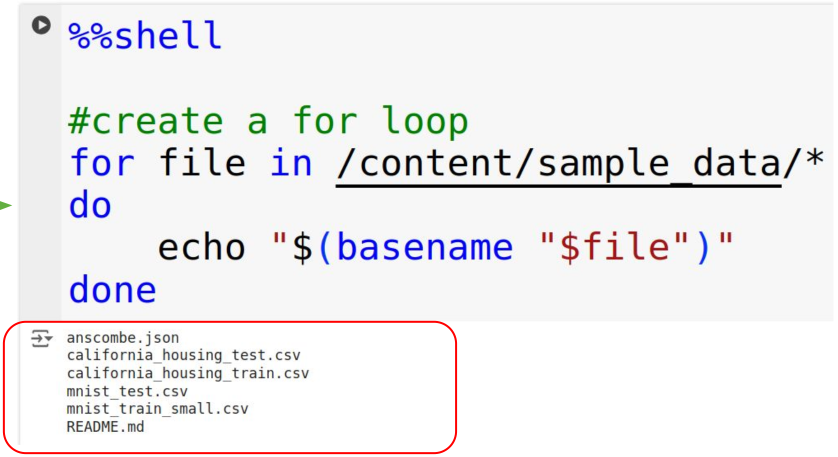
Example 7:



```
%%shell

#create a for loop
for file in /content/sample_data/*
do
    echo "$file"
done
```

```
/content/sample_data/anscombe.json
/content/sample_data/california_housing_test.csv
/content/sample_data/california_housing_train.csv
/content/sample_data/mnist_test.csv
/content/sample_data/mnist_train_small.csv
/content/sample_data/README.md
```

Example 8:



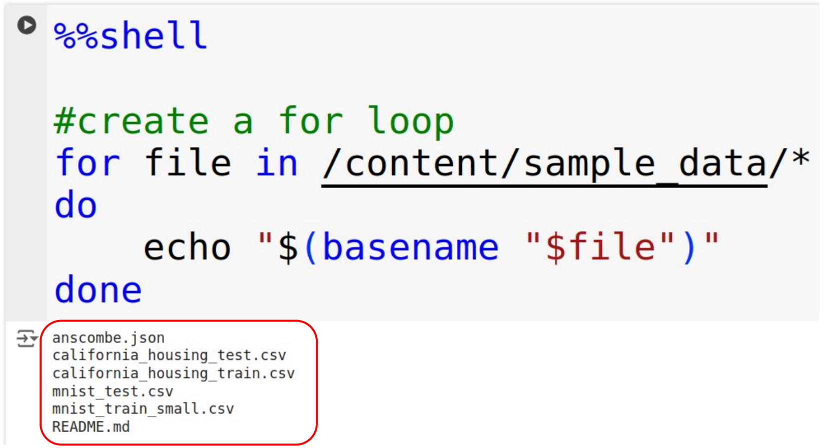
```
%%shell

#create a for loop
for file in /content/sample_data/*
do
    echo "$(basename "$file")"
done
```

```
anscombe.json
california_housing_test.csv
california_housing_train.csv
mnist_test.csv
mnist_train_small.csv
README.md
```

Use the **basename** command, if you want get just the file name without the path.

3. For loop

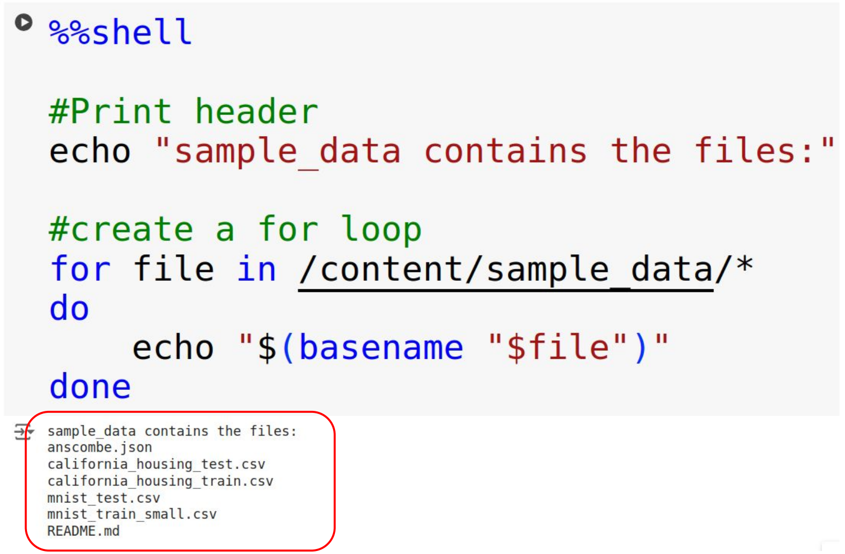
Example 8:



```
%%shell

#create a for loop
for file in /content/sample_data/*
do
    echo "$(basename "$file")"
done
```

```
anscombe.json
california_housing_test.csv
california_housing_train.csv
mnist_test.csv
mnist_train_small.csv
README.md
```

Example 9:



```
%%shell

#Print header
echo "sample_data contains the files:"

#create a for loop
for file in /content/sample_data/*
do
    echo "$(basename "$file")"
done
```

```
sample_data contains the files:
anscombe.json
california_housing_test.csv
california_housing_train.csv
mnist_test.csv
mnist_train_small.csv
README.md
```

Print 1 header

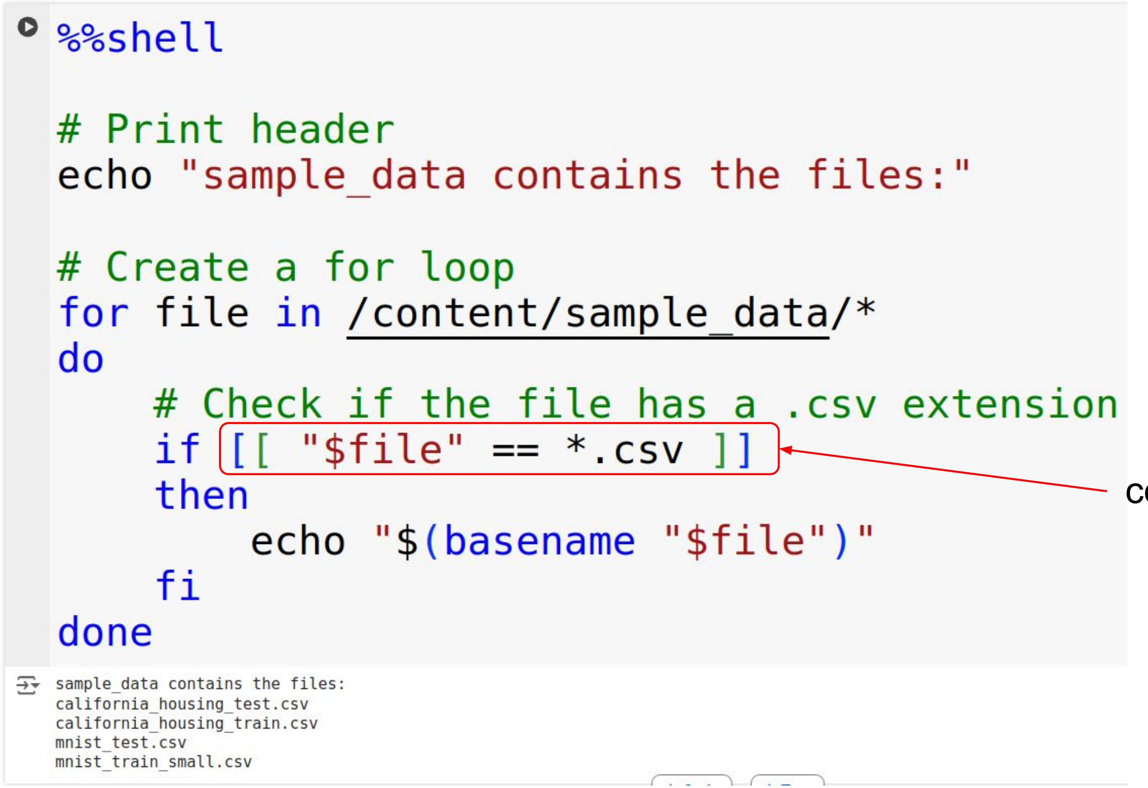
4. If else

4. If else

Example 10: Combine if else in a for loop

Syntax

```
if
then

fi
```



```
%%shell

# Print header
echo "sample_data contains the files:"

# Create a for loop
for file in /content/sample_data/*
do
    # Check if the file has a .csv extension
    if [[ "$file" == *.csv ]]
    then
        echo "$(basename "$file")"
    fi
done
```

```
sample_data contains the files:
california_housing_test.csv
california_housing_train.csv
mnist_test.csv
mnist_train_small.csv
```

condition

4. If else

[[-z STRING]]	empty string
[[-n STRING]]	not empty string
[[STRING == STRING]]	equal
[[STRING != STRING]]	not equal
[[NUM -eq NUM]]	equal
[[NUM -ne NUM]]	not equal
[[NUM -lt NUM]]	less than
[[NUM -le NUM]]	less than or equal
[[NUM -gt NUM]]	greater than
[[NUM -ge NUM]]	greater than or equal
[[STRING =~ STRING]]	regexp
((NUM < NUM))	numeric conditions

4. If else

[[-e FILE]]	exists
[[-r FILE]]	readable
[[-h FILE]]	symlink
[[-d FILE]]	directory
[[-w FILE]]	writable
[[-s FILE]]	size is > 0 byte
[[-f FILE]]	file
[[-x FILE]]	executable
[[FILE1 -nt FILE2]]	i is more recent than 2
[[FILE1 -ot FILE2]]	2 is more recent than 1
[[FILE1 -ge FILE2]]	same files

[[! EXPR]]	not
[[X && Y]]	and
[[X || Y]]	or

Examples used in this presentation, Google Colab:
https://colab.research.google.com/drive/1puSoJnupKsv64PhV8wasR2NTAPRK8LgU?usp=sharing
(For convenient code editing, in Google Colab: File → Save a copy in Drive)

For more about Bash script:

- lecture 2, 3 in MGMA 2024 course: https://github.com/UeenHuynh/MGMA_2024

Thank you!