

Design Shell scripts

starts at 9:05pm

Agenda

1. Quotes
2. Brackets
3. Functions
4. Variables and their scope
5. Input Output Redirection

→ Quotes

①

'

②

"

③

`

④

\$ ()

⑤

\

echo 'ench "\$welcome" > .bashrc

①

'

'

→ no variable expansion.

②

"

"

Double quotes

```
#!/bin/bash
```

```
variable=50
```

```
echo 'Variable value is $variable'
```

Variable value is \$variable → output

if we use " "

output → variable value is 50

③ Backticks and \$()
Command expansion.

```
cat quotes_demo.sh
```

```
#!/bin/bash
```

```
variable=$(date)
```

```
echo "Variable value is $variable"
```

} Program.

```
ubuntu@ip-172-31-39-147:~$ ./quotes_demo.sh
```

Variable value is Mon Oct 14 15:52:19 UTC 2024 → output

```
ubuntu@ip-172-31-39-147:~$
```

⑤ Escape character (\)

```
ubuntu@ip-172-31-39-147:~$ echo "Price is \$10"
```

```
Price is $10
```

→ Brackets

① [] → test condition

if [condition]; then
 commands

fi

Common Tests:

Key Differences:

| Option | Purpose | When True |
|--------|-----------------------------------------------|---------------------------------------------------------|
| -e | Checks if any file or directory exists | True if the path points to any file or directory |
| -f | Checks if a regular file exists | True only if the path points to a regular file |

- **`-eq`**: Equal (for integers)

- **`-ne`**: Not equal (for integers)

- **`-lt`**, **`-le`**: Less than, less than or equal (for integers)

- **`-gt`**, **`-ge`**: Greater than, greater than or equal (for integers)

- **`-e`**: Checks if the file exists.

→ any type (file, directories, symbolic links)

- **`-f`**: Checks if the file is a regular file (not a directory).

→ only files

- **`-d`**: Checks if the file is a directory.

- **`-r`**: Checks if the file is readable.

- **`-w`**: Checks if the file is writable.

- **`-x`**: Checks if the file is executable.

- **-s****: Checks if the file is not empty (has a size greater than zero).

```
if [ "$var" -eq 1 ] ; then  
    commands  
fi
```

```
if [ -e "path" ] ; then  
    echo "File exists"  
fi
```

② Double Square Brackets `[[]]`

`&&` → AND

`||` → OR

→ logical operators

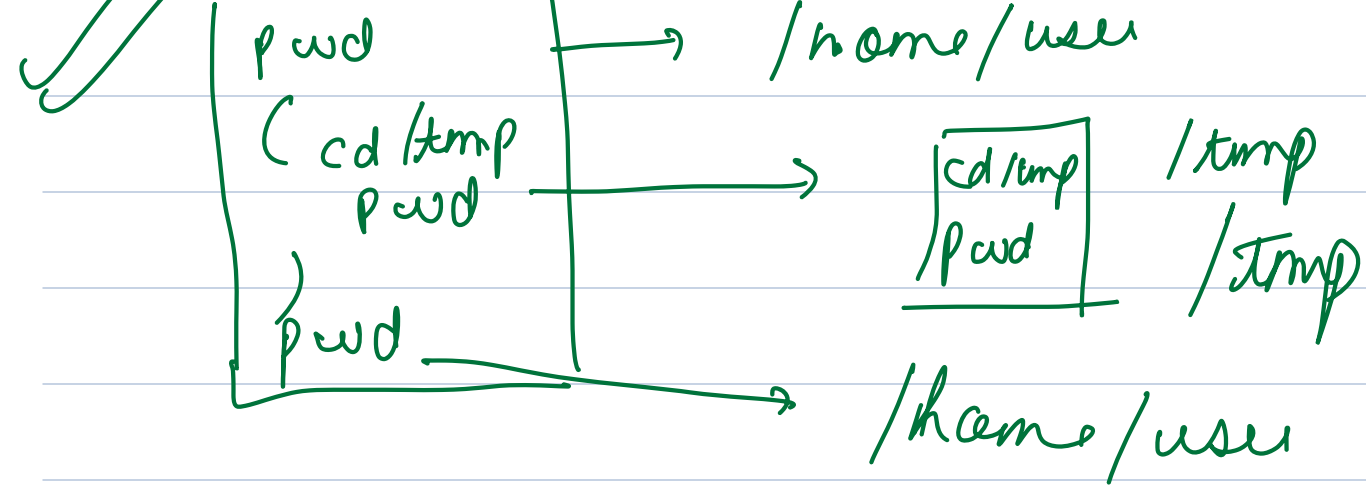
→ String Pattern matching. (*)

→ Regular expressions.

③ Parentheses `()`

→ runs commands in a subshell

✓ ✓ shell



`()` → also used for arrays.

```
cat p_demo.sh
```

```
#!/bin/bash
```

```
echo "Current working directory is $(pwd)"
```

```
(
```

```
    cd /tmp
```

```
    echo "Current working directory is $(pwd)"
```

```
)
```

```
echo "Current working directory is $(pwd)"
```

Output

```
./p_demo.sh
```

```
Current working directory is /home/ubuntu
```

Current working directory is /tmp

Current working directory is /home/ubuntu

Double Parentheses (())
arithmetic evaluation.

#!/bin/bash

x=5

((x++))

→ 6

echo "\$x"

Curly Braces { }

touch file {1..5}

cat curly_braces.sh

#!/bin/bash

```
name="Computer"
```

```
echo "Hello $nameWorld"
```

```
echo "Hello ${name}World"
```

```
./curly_braces.sh
```

```
Hello
```

} Output

Summary Table of Brackets:

| Bracket Type | Common Usage |
|--------------|--------------------------------------------------------------|
| [] | Conditional testing (e.g., ["\$var" -eq 10]) |
| [[]] | Extended conditional testing (e.g., [["\$name" == user*]]) |
| () | Subshell creation (e.g., (cd /tmp; ls)) |
| () | Array declaration (e.g., my_array=(apple banana)) |
| { } | Brace expansion (e.g., echo {A,B,C}) |
| { } | Parameter expansion (e.g., echo \${var}) |
| < > | Input/output redirection (e.g., sort < file.txt) |
| (()) | Arithmetic evaluation (e.g., ((x++))) |

```
Hello ComputerWorld
```

Break 10:18

→ Functions.

→ Syntax:

function_name () {

commands

OR

function function_name {

commands

→ Calling a function.

function_name

function_name "arg 1" "arg 2"

cat some_function.sh

#!/bin/bash

greet_user(){

}


```
echo "Hello $1, Welcome!"
```

```
}
```

function

```
greet_user "Apple" → case 1
```

```
greet_user Apple → case 2
```

```
greet_user "Apple Banana" → case 3
```

```
greet_user "Apple" "Banana" → case 4
```

```
ubuntu@ip-172-31-39-147:~$ ./some_function.sh
```

```
Hello Apple, Welcome!
```

```
Hello Apple, Welcome!
```

```
Hello Apple Banana, Welcome!
```

```
Hello Apple, Welcome!
```

```
ubuntu@ip-172-31-39-147:~$
```

output

→ Return Values.

Exit status.

\$? → used to fetch the exit status.

→ ls

0 → successful

Non zero → Not successful

130 ctrl+c

127 command not found

#!/bin/bash

ls /home

echo "Exit status is \$?"

→ 0

ls /non_existent_directory

→ directory not found.

echo "Exit status is \$?"

→ 2

Global & Local variables.

↓
default

local → keyword local.

local variable1 = 50

Program for flags.

kubectl get pods -n {namespace}

-o wide
free -h

function.

-d



deleting
data

echo " "

-s



saving the
data.

echo " "

```
cat variables.sh
```

```
#!/bin/bash
```

```
process_data() {
```

```
    local option="$1"
```

```
    case $option in
```

```
        -d) echo "Deleting data";;
```

```
        -s) echo "Saving data";;
```

```
        *) echo "Invalid Option";;
```

esac

}

process_data -d

ubuntu@ip-172-31-39-147:~\$./variables.sh

Deleting data

ubuntu@ip-172-31-39-147:~\$ vi variables.sh

ubuntu@ip-172-31-39-147:~\$./variables.sh (passing -n instead of -d)


Invalid Option

ubuntu@ip-172-31-39-147:~\$

→ I/O Redirection.

① Output Redirection.

② Appending
>
>>



echo "apple" > file.txt

③ Input redirection.

<

→ no-operation.

sort < user.txt

: < 'delimiter'

. line 1

: line 2

line 3

delimiter

(4)

2>

— output stream >

— error stream. 2 >

Black Hole → /dev/null

&> → output & error both.

tee command.

'
\$()

↓
print on terminal

↘
send it to a file.

-a → append.