# Linux Workshop - Session 4

OSC - Linux Committee

Mar 01, 2022

Created in / & with ❤

## Linux Workshop - Session 3

1. Recap

## Let's Recap

1. How to view every file in the directory (including hidden) and view their permissions?
2. How to create the following directories all at once `~/Docs/FCIS/Date-Structure`?
3. How to view the current running shell?
4. What does `^l` does?
5. How to create and alias that does `ls -t` and map it to `lt`?
6. How to login into a remote client with the following date:

```
# Username: root
# Host_IP: 153.223.14.4
```

## Let's Recap

7. Head, Cat and Tail?
8. What are the three data streams in Linux?
9. How to send output of a command to a file?
10. How to send error of a command to a file?
11. How to send both output and error of a command to a file?
12. How to use the output of a command as an input to another command?
13. How to make a file excutable?
14. What is `less`?

## Let's Recap

15. How to search for a specific word in a file?
16. How to sort a file?
17. How to create a new user?
18. How to delete a user?
19. How to become a root?
20. How to install `telegram`?
21. How to kill a process?

## What is Shell Scripting?

Now, we know the some Linux commands, but we don't know how to use them proporly within a shell script, right? **WRONG!**

Shell scripts are nothing but plain text files that contains a series of commands that will be excuted on line at a time when the user run the script.

Although Linux is extensionless, it's a convention to give a shell script the **(.sh)** extension. You can give it whatever you want.

## First Shell Scripts

### Print “Hello, World!”

```
#!/bin/bash

echo "Hello, World!"
```

### How to run a script?

1. Make the script excutable
2. Run the script by typing either the absoulte path or the relative path of it

## First Shell Scripts

### List the content of the current directory

```
#!/bin/bash

ls
```

## First Shell Scripts

**Print "Hello, World!" & List the content of the current directory**

```
#!/bin/bash

echo "Hello, World!"
ls
```

## Variables

Can you guess how to assign a variable in Bash?

## Variables

### Assigning values to Variables

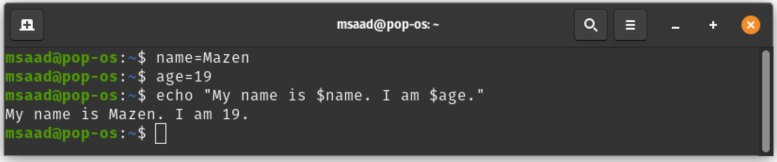
```
varname="text with spaces"
varname='text with spaces without any processing'
varname=textwithoutspaces
varname=20
```

## Reading a Variable

To read a variable, we place **$** before its name to tell bash to process it as a variable not a normal word

```
name=Muhammed
echo "My name is name"
echo "My name is $name"
echo 'My name is $name'
echo $name
```

## Dealing with Variables

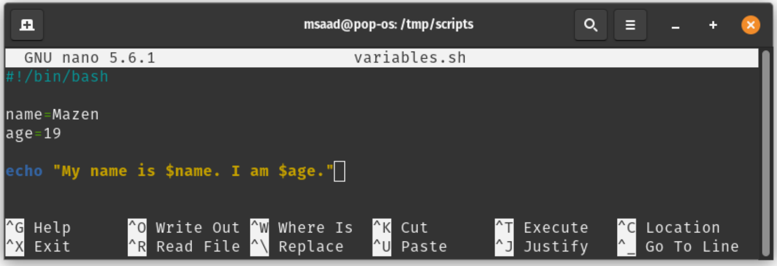


```
msaad@pop-os:~$ name=Mazen
msaad@pop-os:~$ age=19
msaad@pop-os:~$ echo "My name is $name. I am $age."
My name is Mazen. I am 19.
msaad@pop-os:~$ 
```

Figure 1: Variables in on Shell Prompt

Can you do the same thing but in a shell script?

## Dealing with variables

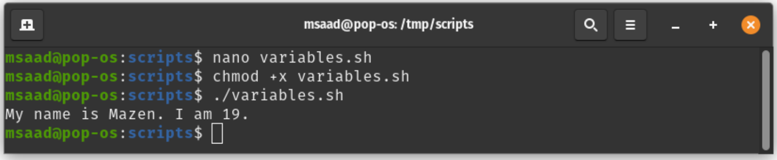


msaad@pop-os: /tmp/scripts

```
  GNU nano 5.6.1                          variables.sh
#!/bin/bash

name=Mazen
age=19

echo "My name is $name. I am $age."


^G Help        ^O Write Out  ^W Where Is  ^K Cut         ^T Execute   ^C Location
^X Exit        ^R Read File  ^\ Replace   ^U Paste       ^J Justify   ^_ Go To Line
```

Figure 2: Variables in a Shell Script

## Dealing with variables



```
msaad@pop-os:scripts$ nano variables.sh
msaad@pop-os:scripts$ chmod +x variables.sh
msaad@pop-os:scripts$ ./variables.sh
My name is Mazen. I am 19.
msaad@pop-os:scripts$ 
```

Figure 3: Running the script

## Dealing with variables

### Guess the output of each of the following lines

```
x=5
echo "$x"
echo $x
echo x
echo "x"
echo 'x'
echo '$x'
```

## Taking input from users

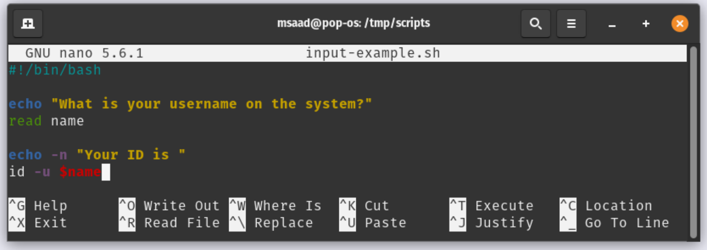
**New Command Alert!**

```
read varname
```

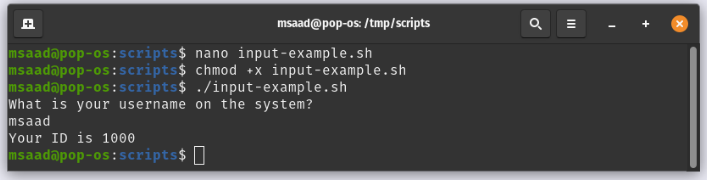
We can use this command on command line or in a shell script

# Taking input from users

## Taking input in a shell script



```
  GNU nano 5.6.1                    input-example.sh
#!/bin/bash

echo "What is your username on the system?"
read name

echo -n "Your ID is "
id -u $name

^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line
```



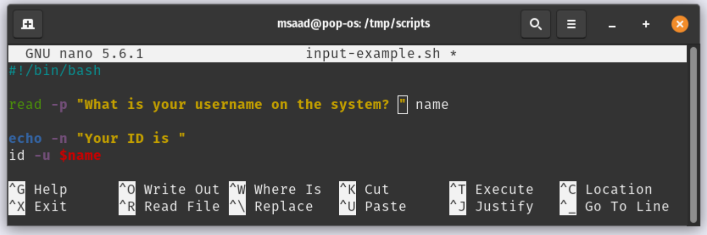
```
msaad@pop-os:scripts$ nano input-example.sh
msaad@pop-os:scripts$ chmod +x input-example.sh
msaad@pop-os:scripts$ ./input-example.sh
What is your username on the system?
msaad
Your ID is 1000
msaad@pop-os:scripts$ 
```

## Taking input from users

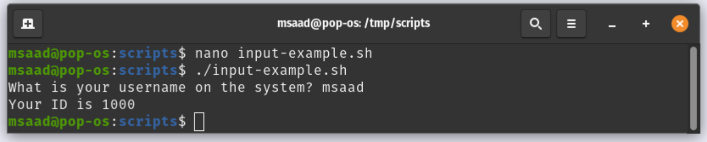
Now let's get the same job done more elegantly. You can use (read) with the (-p) flag to prompt the user with a question and take input at the same time.

## Taking input from users

### Elegent input



```
  GNU nano 5.6.1                    input-example.sh *
#!/bin/bash

read -p "What is your username on the system? " name

echo -n "Your ID is "
id -u $name

^G Help       ^O Write Out ^W Where Is  ^K Cut        ^T Execute    ^C Location
^X Exit       ^R Read File ^\ Replace   ^U Paste      ^J Justify    ^_ Go To Line
```



```
msaad@pop-os:scripts$ nano input-example.sh
msaad@pop-os:scripts$ ./input-example.sh
What is your username on the system? msaad
Your ID is 1000
msaad@pop-os:scripts$ 
```

## Taking input from users

### Command-line arguments

Command line arguments are nothing new to us. We introduced it in the second session when we explained the command line syntax. Let's recap:

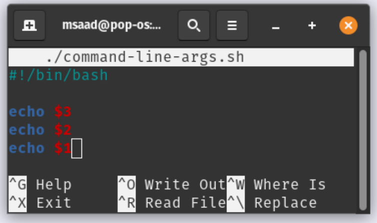
```
# Command     # Option/Flag    # Argument
   ls                -a          /var/log
```

### Command-line arguments with shell scripts

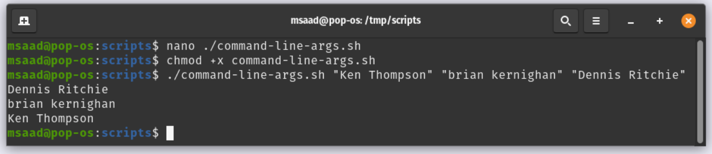
We can do the same thing with our bash scripts. To do this we use the variables from $1 to $n. These are automatically set by the system when we run our script so all we need to do is refer to them.

# Taking input from users

## Command-line arguments with shell scripts



```
    ./command-line-args.sh
#!/bin/bash

echo $3
echo $2
echo $1

^G Help       ^O Write Out^W Where Is
^X Exit       ^R Read File^\ Replace
```

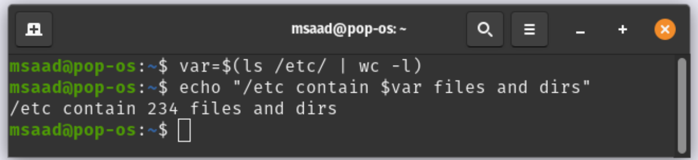


```
msaad@pop-os:scripts$ nano ./command-line-args.sh
msaad@pop-os:scripts$ chmod +x command-line-args.sh
msaad@pop-os:scripts$ ./command-line-args.sh "Ken Thompson" "brian kernighan" "Dennis Ritchie"
Dennis Ritchie
brian kernighan
Ken Thompson
msaad@pop-os:scripts$
```

## Command Substitution

What if you want to save the output of a command in a variable ?

### Command Substitution



```
msaad@pop-os:~$ var=$(ls /etc/ | wc -l)
msaad@pop-os:~$ echo "/etc contain $var files and dirs"
/etc contain 234 files and dirs
msaad@pop-os:~$ 
```

## Command Substitution

## Special Variables

### Environment Variables

- `BASH_VERSION` Bash version.
- `HOST_NAME` Host name.
- `HOME` Home directory.
- `PATH` Executable locations.
- `TERM` Default terminal.
- `SHELL` Default shell.
- `EDITOR` Default text editor.

## Special Variables

### Other Useful Variables

- $0 - The name of the Bash script.
- $1 - First argument to the Bash script.
- $# - How many arguments were passed to the Bash script.
- $@ - All the arguments supplied to the Bash script.
- $? - The exit status of the most recently run process.
- $$ - The process ID of the current script.
- $USER - The username of the user running the script.
- $HOSTNAME - The hostname of the machine the script is running on.
- $RANDOM - Returns a different random number each time is it referred to.
- $LINENO - Returns the current line number in the Bash script.

## Conditionals in Bash

### If Statement

```
if [[ condition ]]
then
    #DoSomething
fi
```

## Conditionals in Bash

### If Statement Example

```
if [[ $x -eq 5 ]]
then
    echo "X equals 5"
fi
```

## Conditionals in Bash

### If-Elif Statement Example

```
if [[ condition ]]
then
    #DoSomething
elif [[ condition ]]
then
    #DoSomething
else
    #DoSomething
fi
```

## Conditionals in Bash

### Writing conditionals in BASH

- Start a condition with `if [[ condition ]]`
- The next line contains `then` which is roughly equivalent to '{'
- Write the commands that will execute if the condition is true.
- End your condition with `fi` which is roughly equivalent to '}'
  - Or start an `elif [[ condition ]]`, with `then` in the line after it.
    - Write the commands that will execute if the `elif` condition is true.
    - End your conditionals with `fi`
  - Or start an `else`, with **NO** `then` in the line after it.
    - Write the commands that will execute if the `else` condition is true.
    - End your conditionals with `fi`

## Conditions

Comparing Numerical Variables

| Expression in C | Expression in BASH | Evaluates to true when: |
|---|---|---|
| `a == b` | `$a -eq $b` | a is equal to b |
| `a != b` | `$a -ne $b` | a is not equal to b |
| `a < b` | `$a -lt $b` | a is less than b |
| `a > b` | `$a -gt $b` | a is greater than b |
| `a >= b` | `$a -ge $b` | a is greater than or equal to b |
| `a <= b` | `$a -le $b` | a is less than or equal to b |

## Conditions

Comparing String Variables

| Expression in C | Expression in BASH | Evaluates to true when: |
|---|---|---|
| `a == b` | `$a = $b` or `$a == $b` | a is the same as b |
| `a != b` | `$a != $b` | a is different from b |
| `strlen(a) == 0` | `-z $a` | a is empty |

## Conditions

Combining Conditions

| Expression in C | Expression in BASH |
|---|---|
| `(cond. A \|\| cond. B)` | `[[ cond. A \|\| cond. B ]]` |
| `(cond. A && cond. B)` | `[[ cond. A && cond. B ]]` |
| `(!cond. A)` | `[[ ! cond. A ]]` |

## Conditionals

### Case Statements

```
case <variable> in
<pattern 1>)
    <commands>
    ;;
<pattern 2>)
    <other commands>
    ;;
esac
```

## Conditionals

### Case Statements Examples

```
case $1 in
    start)
        echo starting
        ;;
    stop)
        echo stopping
        ;;
    restart)
        echo restarting
        ;;
    *)
        echo don\'t know
        ;;
esac
```

## Loops

### For Loops

```
for VAR in RANGE
do
    #SOMETHING
done
```

## Loops

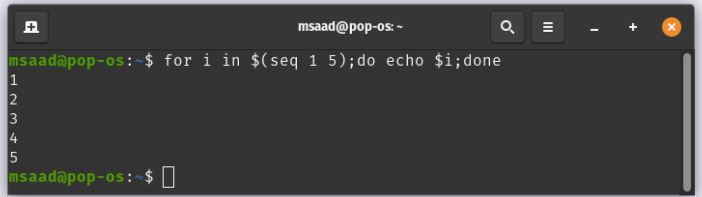
### For Loops Example

```
read x
for i in $(seq 1 $x)
do
    echo "This is Line $i"
done
```

## Loops

**TIP**

You can write a for loop in you terminal in one line:



```
msaad@pop-os:~$ for i in $(seq 1 5);do echo $i;done
1
2
3
4
5
msaad@pop-os:~$ 
```

## Loops

### While Loop

```
while [[ CONDITION ]]
do
    #SOMETHING
done
```

## Loops

### While Loop Example

```
x=1
while [[ $x -le 10 ]]
do
    echo "This is line $x"
    let x+=1
done
```

## Loops

### Break Statements

```
while [[ x -lt 10 ]]
do
    read i

    if [[ i -eq 0 ]]
    then
        break
    fi
    echo $i
done
echo "break sent me here"
```

## Loops

### Continue Statements

```
while [[ x -lt 10 ]]
do
    read i

    if [[ i -eq 0 ]]
    then
        echo "Skipping the rest of the code!"
        continue
    fi
    echo $i
done
```

## Functions

### Function in Bash

```
function NAME #Function Definition
{
    #DoThings
}
NAME #Function call
```

OR

```
NAME() #Function Definition
{
     #DoThings
}
NAME #Function call
```

## Functions

### Functions Examples

```
function hello
{
    for i in `seq 1 5`
    do
        echo "Hello!"
    done
}

hello
```

## Functions

### Functions Examples

```
function list
{
    ls $1
}
```

## Functions

### Fork Bomb

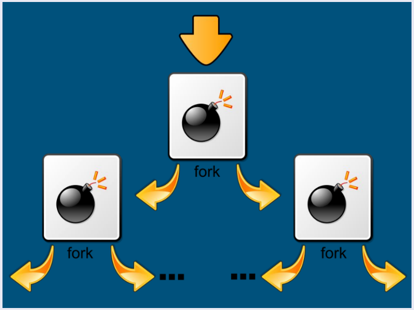


Figure 4: Fork Bomb

## Functions

### Fork Bomb

```
# :(){:|:&};:
:()            # Create a function named ' : '
{              # Start of the function body
    : | :&     # Calls itself, once in the foreground
               # and once in the background
}              # End of the function body

:              # Function call
```