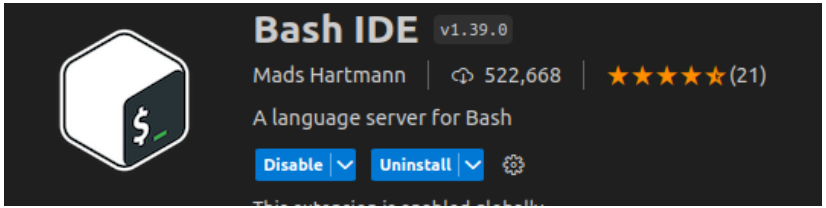
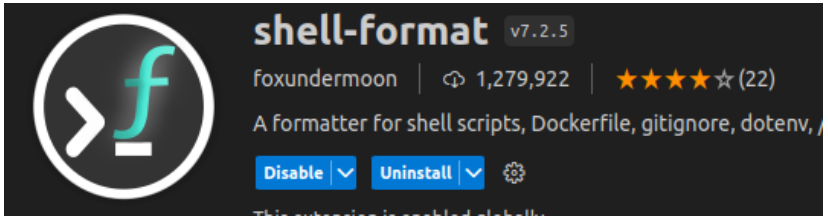
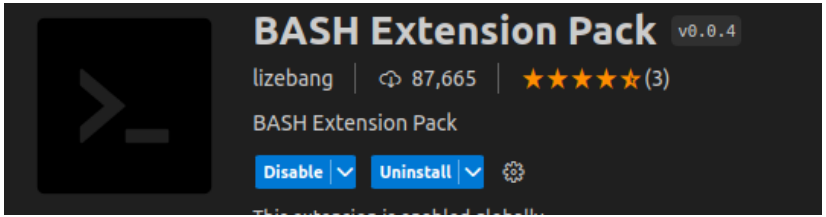
01_Introduction Bash

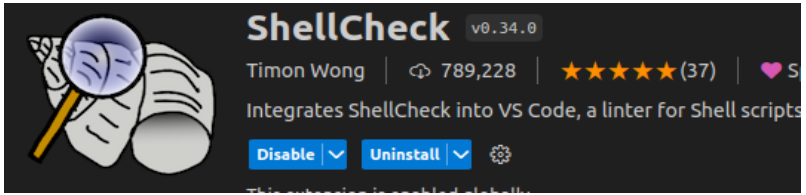
Moatasem Elsayed

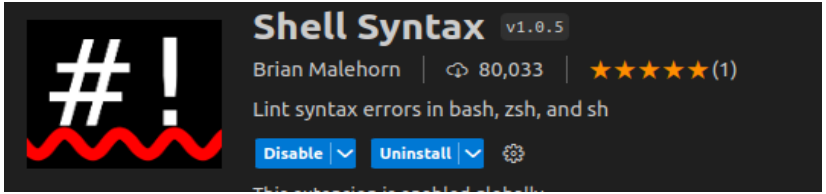
Vscode Extensions

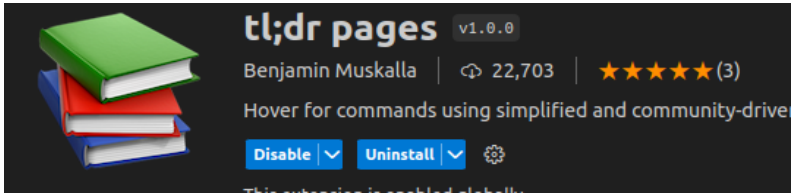












Content

Hello world

```bash
#!/bin/bash

echo "hello wolrd"
name="moatasem"
echo -n "hello world ${name}Elsayed"
printf "hello world %s \n" $name
```

shebang

When a text file with a shebang is used as if it is an executable in a Unix-like operating system, the program loader mechanism parses the rest of the file's initial line as an interpreter directive.

```
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~/Diploma/mypresetation/04bash/Bash_Basics/Basic_commands$ chmod u+x 01Echo.sh
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~/Diploma/mypresetation/04bash/Bash_Basics/Basic_commands$ ./01Echo.sh
hello wolrd
hello world moatasemElsayedhello world moatasem
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~/Diploma/mypresetation/04bash/Bash_Basics/Basic_commands$
```

```bash
#!/bin/bash

# Read the user input

echo "Enter the user name: "
read -p "name is" first_name      read without -r will mangle backslashes.
echo "The Current User Name is $first_name"
echo
read -sp "Password:" pass    read without -r will mangle backslashes.
echo "Password is $pass"
# echo "Enter other users'names: "
# read name1 name2 name3
# echo "$name1, $name2, $name3 are the other users."

#Array
echo "Enter names : "
read -a names      read without -r will mangle backslashes.
echo "The entered names are : ${names[0]}, ${names[1]}."
```

Read

Variables

```
var1="Moatasem"
var2=Moatasem
var3=123
var4="123"
var5=$(date)
var6=`date`    Use $(...) notation instead of legacy backticks `...`.
echo "$var1"
echo "$var2"
echo "${var2}Elsayed" #concat
echo "$var2Elsayed" #var not defined    var2Elsayed is referenced but not assigned.
echo "${var3}456" #behave like string
echo "${var4}456" # same
echo "$((++var3))" # behave like number
echo "$((++var4))" # same
echo "$var5"
echo "$var6"
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~/Diploma/mypresetation/04bash/Bash_Basics/Basic_commands$ ./01_vars.sh
Moatasem
Moatasem
MoatasemElsayed

123456
123456
124
124
Sun Oct  1 04:38:18 PM EEST 2023
Sun Oct  1 04:38:18 PM EEST 2023
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~/Diploma/mypresetation/04bash/Bash_Basics/Basic_commands$
```

comments

Basic_commands > $ 06comments.sh > ...

```
#!/bin/bash
#single comment

<<test    This redirection does
multiline
multiline
test

variable1="hello"
variable2="world"
echo "$variable1$variable2"
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL

```
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~/Di
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~/Di
helloworld
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~/Di
```

If ,test

```
ing-3-15IAH7:~$ man [ #man test
```

Manual

```
ME
     test - check file types and

NOPSIS
     test EXPRESSION
     test
     [ EXPRESSION ]
     [ ]
     [ OPTION

SCRIPTION
     Exit with the status determi

     --help display this help and exit

     --version
            output version information and exit

     An omitted EXPRESSION defaults to false.  Otherwise, EXPRESSION is true or false and sets exit status.  It is one of:

     ( EXPRESSION )
            EXPRESSION is true

     ! EXPRESSION
            EXPRESSION is false

     EXPRESSION1 -a EXPRESSION2
            both EXPRESSION1 and EXPRESSION2 are true

     EXPRESSION1 -o EXPRESSION2
            either EXPRESSION1 or EXPRESSION2 is true

     -n STRING
            the length of STRING is nonzero

     STRING equivalent to -n STRING

     -z STRING
            the length of STRING is zero

     STRING1 = STRING2
            the strings are equal

     STRING1 != STRING2
            the strings are not equal

     INTEGER1 -eq INTEGER2
            INTEGER1 is equal to INTEGER2

     INTEGER1 -ge INTEGER2
            INTEGER1 is greater than or equal to INTEGER2
```

```
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~$ test 1 -eq 1
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~$ echo $?
0
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~$ test 1 -eq 2
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~$ echo $?
1
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~$
```

```
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~$ if [ 1 -eq 1 ] ; then echo "hello"; fi
hello
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~$
```

```
read -p "enter String" str    read without -r will mangle backslashes.

if [ -z "$str" ]; then
    echo "the string is empty"
else
    echo "the string is : ${str}"
fi

read -p " Enter number : " number    read without -r will mangle backslashes.

if [ "$number" -lt 125 ]; then
    echo "Value is less than 125"
elif [ "$number" -lt 200 ]; then
    echo " the value is bettween 200 and 125"
else
    echo " value is greater than 200"
fi

test "moatasem" = "ali" && echo "true" || echo "this is false"
test 001 -eq 1 && echo true || echo false

read -p " Enter number : " str    read without -r will mangle backslashes.
### test command
test 125 == 100 && echo "hello"
x=001
if [ $x -eq 1 ]; then
    echo hello
fi
read -p "variable x" x    read without -r will mangle backslashes.
test -z "$x" && echo "empty" || echo "string has a value"

test -f "./01Echo.sh" && echo "file exist" || echo "file doesn't exist"

if [ -f "./01Ech.sh" ]; then
    echo "file exist "
else
    echo "file doesn't exist"
fi
```

```
if [ 1 -eq 1 -a 4 -gt 3 ]; then
    echo "#rule 1"
fi

if [ 1 -eq 1 ] && [ 5 -gt 4 ]; then
    echo "#rule 2"
fi

if [[ 1 -eq 1 && 5 -gt 4 ]]; then
    echo "#rule 3"
fi

#c style
if ((001 == 1)); then
    echo "#rule 4"
fi

x=0
if ((x++)); then
    echo "#rule 5"
fi
```

OR , AND

Basic_commands > $ 0Bnestedif.sh > ...

```bash
#!/bin/bash

x=5
y=7

if [ $x == $y ]
then
echo "true"
else
echo "false"
fi
echo

if [ 8 -gt 6 ] && [ 10 -eq 10 ]  #if [[ 8 -gt 6  &&  10 -eq 10 ]]     This expression is constant.
then
echo "Conditions are true"
fi
echo
if [ 8 -gt 7 ] || [ 10 -eq 3 ]    This expression is constant. Did you forget the $ on a variable?
then
echo " Condition is true. "
fi

#OR
if [[ 10 -eq 10 && 5 -gt 4 || 3 -eq 4 || 3 -lt 6 ]];
then
echo "Condition is true."
fi
echo
if [ $1 -gt 50 ]      Double quote to prevent globbing and word splitting.
then
  echo "Number is greater than 50."

  if (( $1 % 2 == 0 ))  #for(( i=0;i<size;i++))
  then
    echo "and it is an even number."
  fi
fi
```

Loops

for

```
for i in 1 2 3 4 5 6; do
    echo $i
done

for i in {1..10}; do
    echo "the Number is ${i}"
done

learn="Start learning from Javatpoint."
for i in $learn; do
    echo "$i"
done
```

```
#output command

x=$(ls)
for i in $x; do
    echo "the file name is ${i}"
done

files=$(ls test)
for i in ${files}; do
    echo "the content of file ${i} is :"
    cat "./test/${i}"
    echo
done

echo "Thank You."
```

While - until

```bash
#!/bin/bash

x=0
y=3
while [[ $x -le 5  &&  $y -ge 0 ]]
do
echo  "x= $x"
echo "y=$y"
((x++))
((y--))
done
echo " thanks "
```

```bash
#!/bin/bash
i=1
until [ $i -gt 10 ]
do
echo $i
((i++))
done

#OR
#Bash Until Loop example with multiple conditions

max=5
a=1
b=0

until [ $a -gt $max ] || [ $b -gt $max ];
do
echo "a = $a & b = $b."
((a++))
((b++))
done
```

exper

Expr , let ,$(()),bc

```
moatsem@moatsem-Idea
$  echo 1+3 | bc
4
$
```

```bash
#!/bin/bash

x=8
y=2
z= expr 1 + 2    Remove space after = if trying to assign a value (fo
echo $z    z is referenced but not assigned.

let sum=3+10    Instead of 'let expr', prefer (( expr )) .
echo $sum
((x++))
echo $x

echo "x=8, y=2"
echo "Addition of x & y"
echo $(( $x+ y ))      $/${} is unnecessary on arithmetic variables.
echo "Subtraction of x & y"
echo $(( $x - $y ))      $/${} is unnecessary on arithmetic variables
echo "Multiplication of x & y"
echo $(( $x * $y ))      $/${} is unnecessary on arithmetic variables
echo "Division of x by y"
echo $(( $x / $y ))      $/${} is unnecessary on arithmetic variables
echo "Exponentiation of x,y"
echo $(( $x ** $y ))      $/${} is unnecessary on arithmetic variable
echo "Modular Division of x,y"
echo $(( $x % $y ))      $/${} is unnecessary on arithmetic variables
echo "Incrementing x by 5, then x= "
(( x += 5 ))
echo $x
echo "Decrementing x by 5, then x= "
(( x -= 5 ))
echo $x
echo "Multiply of x by 5, then x="
(( x *= 5 ))
echo $x
echo "Dividing x by 5, x= "
(( x /= 5 ))
echo $x
echo "Remainder of Dividing x by 5, x="
(( x %= 5 ))
echo $x
```

1. **Basic Arithmetic Operations:**

   You can perform addition, subtraction, multiplication, and division as follows:

```bash
# Addition
result=$((5 + 3))

# Subtraction
result=$((10 - 2))

# Multiplication
result=$((6 * 4))

# Division
result=$((12 / 3))
```

2. **Exponentiation:**

   Bash does not have a built-in operator for exponentiation, but you can use the `**` operator in double parentheses or utilize the `bc` command for floating-point exponentiation:

```bash
# Integer exponentiation
result=$((2 ** 3))  # Result: 8

# Floating-point exponentiation using bc
result=$(echo "2 ^ 3" | bc -l)  # Result: 8.00000000000000000000
```

3. **Modulo (Remainder):**

   To find the remainder of a division, use the `%` operator:

```bash
result=$((10 % 3))  # Result: 1
```

4. **Increment and Decrement:**

   You can increment and decrement variables as follows:

```bash
# Increment
((counter++))  # Increment the 'counter' variable by 1

# Decrement
((counter--))  # Decrement the 'counter' variable by 1
```

5. **Comparison Operators:**

   You can use comparison operators to evaluate expressions, and they return a Boolean result (`0` for false, `1` for true):

```bash
# Equal to
((5 == 5))  # Result: 1 (true)

# Not equal to
((3 != 2))  # Result: 1 (true)

# Greater than
((10 > 5))  # Result: 1 (true)

# Less than
((7 < 12))  # Result: 1 (true)
```

6. **Logical Operators:**

   You can use logical operators like `&&` (AND) and `||` (OR) within double parentheses to create compound expressions:

```bash
(( (5 > 3) && (4 < 6) ))  # Result: 1 (true)
(( (5 > 3) || (2 == 1) ))  # Result: 1 (true)
```

Arguments

arguments

- `$0` will be `./myscript.sh`
- `$1` will be `arg1`
- `$2` will be `arg2`

Here's a simple script that demonstrates how to access command-line arguments:

bash Copy code

```bash
#!/bin/bash

echo "Script name: $0"
echo "First argument: $1"
echo "Second argument: $2"
```

2. **Special Variables:**

In addition to `$0`, `$1`, `$2`, etc., there are some special variables you can use:

- `$#`: The number of command-line arguments.
- `$*`: All the command-line arguments as a single string.
- `$@`: All the command-line arguments as an array of strings.

For example, you can use `$#` to check how many arguments were passed to your script and loop through them using a `for` loop:

bash Copy code

```bash
#!/bin/bash

echo "Number of arguments: $#"

for arg in "$@"; do
  echo "Argument: $arg"
done
```

Shift Command:

The `shift` command allows you to shift the positional parameters (command-line arguments) to the left. This can be useful when you want to process arguments in a loop and remove them as they are processed:

bash Copy code

```bash
#!/bin/bash

echo "Original arguments: \$1=$1, \$2=$2, \$3=$3"

# Shift the arguments to the left
shift

echo "Shifted arguments: \$1=$1, \$2=$2, \$3=$3"
```

```bash
##echo $1 $2 $3 ' -> echo $1 $2 $3'
<<COMMENTS    This redirection doesn't have a command. Move to its command (or use 'true' as no-op).
$0 specifies the name of the script to be invoked.
$1-$9 stores the names of the first 9 arguments or can be used as the arguments' positions.
$# specifies the total number (count) of arguments passed to the script.
$* stores all the command line arguments by joining them together.  1 2 3
$@ stores the list of arguments as an array.                        1,2,3
$$ specifies the process ID of the current script.
$? specifies the exit status of the last command or the most recent execution process.

COMMENTS
echo $$
# # sleep 1000
echo $0    Double quote to prevent globbing and word splitting.
echo $1    Double quote to prevent globbing and word splitting.
echo $4    Double quote to prevent globbing and word splitting.
echo $#   #count
echo "$@" #list 1 2 3 4
echo $*   # string 1234    Use "$@" (with quotes) to prevent whitespace problems.
asdasd
echo $? # check last command 0 means success or not faild

if [ $? -eq 0 ]; then    This $? refers to echo/printf, not a previous command. Assign to variable to
    echo succes
fi
# argument with array ##
args=("$@")
# echo arguments to the shell
echo ${args[0]}    Double quote to prevent globbing and word splitting.
echo ${args[0]} ${args[1]} ${args[2]} ${args[3]} ' -> args=("$@"); echo ${args[0]} ${args[1]} ${args[2

for i in "$@"; do
    echo $i    Double quote to prevent globbing and word splitting.
done
```

Loop on arguments

```
# take care of ()
arg=("$@") # formailze array
size=$#    # count
echo "SIZE IS $size"
for i in "${arg[@]}"; do
    echo "$i"
done
```

PROBLEMS 25 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~/Diploma/mypresetation/04bash/Bash_Basics/Basic_commands$ ./07argumentwithloop.sh 1 2 3 4
SIZE IS 4
1
2
3
4
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~/Diploma/mypresetation/04bash/Bash_Basics/Basic_commands$
```

Dmenu labs

Basic_commands > $ dmenu_script.sh > ...

```bash
#!/bin/bash

value="${HOME}/scripts\n${HOME}/c++\n${HOME}/temppoky"

select=$(echo -e "$value" | dmenu)

nautilus "$select" &
```

```bash
#!/bin/bash

value="${HOME}/scripts\n${HOME}/c++\n${HOME}/temppoky"

select=$(echo -e "$value" | rofi -dmenu)

nautilus "$select" &
```

```bash
#!/bin/bash

value="${HOME}/scripts\n${HOME}/c++\n${HOME}/temppoky"

select=$(echo -e "$value" | fzf)

nautilus "$select" &
```

Compare files

```bash
#!/bin/bash

if [ -z "$1" -o -z "$2" ]; then    Prefer [ p ] || [ q ] as [ p -o q ] is not well defined.
    echo "usage 11_files_compare <file1> <file2>"
    exit 0
fi
file1=$(md5sum $1)    Double quote to prevent globbing and word splitting.
file2=$(md5sum $2)    Double quote to prevent globbing and word splitting.
# echo "file 1 is $file1"
# echo "file 2 is $file2"

IFS=' '
read -ra ADDR1 <<<$file1    Double quote to prevent globbing and word splitting.
read -ra ADDR2 <<<$file2    Double quote to prevent globbing and word splitting.
echo "${ADDR1[0]}"
echo "${ADDR2[0]}"

if [ "${ADDR1[0]}" = "${ADDR2[0]}" ]; then
    echo "these files are similar"
else
    echo "these files are different"
fi
```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS bash - Basic_commands

```
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~/Diploma/mypresetation/04bash/Bash_Basics/Basic_commands$ ./11_files_compare 03globalvariable.sh 04math.sh
71a1e47f26fe1f3f2523ff62bbc72d92
990bf371748f0c5c6b13b708203aad64
these files are different
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~/Diploma/mypresetation/04bash/Bash_Basics/Basic_commands$ ./11_files_compare 03globalvariable.sh 03globalvariable.sh
71a1e47f26fe1f3f2523ff62bbc72d92
71a1e47f26fe1f3f2523ff62bbc72d92
these files are similar
```

notify-send

```bash
#!/bin/bash

BATTERY_PATH="/sys/class/power_supply/BAT1/capacity"
BATTERY_Value=$(cat $BATTERY_PATH)

if [ "${BATTERY_Value}" -lt 120 ]; then
    notify-send "please take care your battery less than 50 "
fi
```

Disk monitor

$ disk_monitor.sh ×

$ disk_monitor.sh > ...

```bash
#!/bin/bash
MAX=90
storage=$(df -h | grep nvme0n1p2 | awk '{print $5}')
echo "$storage"

val=${storage::-1}
echo "$val"
if [ "$val" -gt $MAX ]; then
    dockersize=$(doas du -h -s /var/lib/docker/overlay2 | awk '{print $1}')
    notify-send " disk :$storage docker:$dockersize"
    /usr/bin/python3 /usr/bin/x-terminal-emulator -e "/bin/sh -i -c \"watch df -h \""
fi
```

Logout

```bash
#!/bin/bash

choice=$(echo -e "Logout\nReboot\nShutdown" | dmenu -p "Select an action:")

if [ "$choice" = "Logout" ]; then
    # Logout
    pkill -KILL -u "$USER"
elif [ "$choice" = "Reboot" ]; then
    # Reboot
    sudo reboot
elif [ "$choice" = "Shutdown" ]; then
    # Shutdown
    sudo shutdown -h now
else
    echo "Invalid choice"
fi
```

Class generator

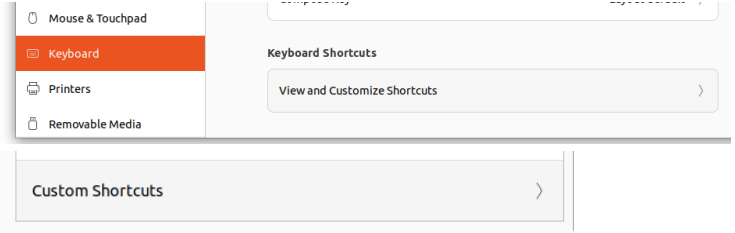
Class_Generator > $ classGenerator.sh

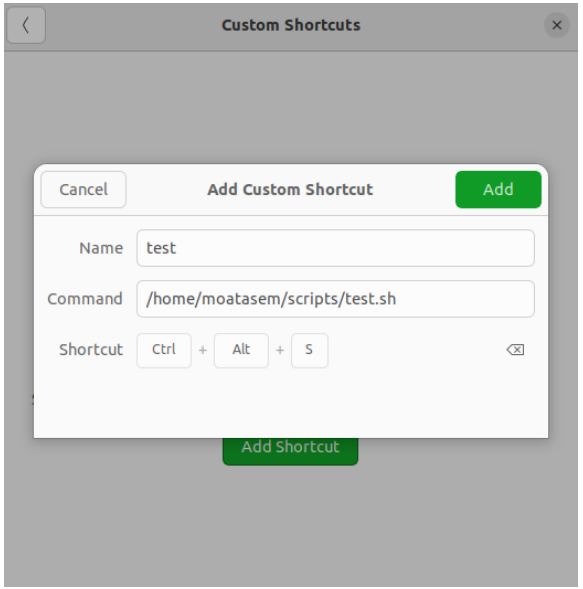
You, 1 second ago | 2 authors (Moatasem-Elsayed and others)

```bash
#!/bin/bash

read -p " please enter your name:" name    read without -r will mangle backslashes.
echo "hello mr ${name}"
read -p " please enter your class name " classname    read without -r will mangle backslashes.
read -p " please enter your namespace " namespace    read without -r will mangle backslashes.
echo " your the class is ${classname} and your namespace is ${namespace}"

echo "
#pragma once
/*******************************************/
//
//              CopyRight ${name}
//
/*******************************************/
/*
author : ${name}
date :$(date)
brief:
*/
namespace $namespace {
class ${classname}{

public:
    ${classname}();
    ~${classname}();
private:      Moatasem-Elsayed, 10 months ago • basic script …

};
}
" >${classname}.hpp    Double quote to prevent globbing and word splitting.
read -p " do you want the cpp file  Y\N" answer    read without -r will mangle backslashes.

if [ ${answer} == "Y" ]; then    Double quote to prevent globbing and word splitting.
    echo "
/*******************************************/
//
//              CopyRight ${name}
//
/*******************************************/
/*
author : Moatasem Elsayed
date :$(date)
brief:
*/
#include \"${classname}.hpp\"

namespace $namespace {
    ${classname}::${classname}(){}
    ${classname}::~${classname}(){}
```

shortcut

Ubuntu





sxhkd

```
sxhkd -c $HOME/.config/sxhkd/sxhkdrc &
export SXHKD_SHELL="/bin/sh"
~
```

```
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~$ head .config/sxhkd/sxhkdrc
#firefox
#youtube
super + ctrl + y
    firefox https://www.youtube.com/
#gpt
super + ctrl + t
    firefox https://chat.openai.com/

#firefox link
super + ctrl + f
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~$
```

```
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~$ alias restartsxhkd
alias restartsxhkd='kill -10 "$(pidof sxhkd)"'
moatsem@moatsem-IdeaPad-Gaming-3-15IAH7:~$
```

```
SXHKD(1)                                                    Sxhkd Manual

NAME
       sxhkd - Simple X hotkey daemon

SYNOPSIS
       sxhkd [OPTIONS] [EXTRA_CONFIG ...]

DESCRIPTION
       sxhkd is a simple X hotkey daemon with a powerful and compact configuration syntax.

OPTIONS
       -h
```

Report

ul_scripts > $ report.sh > ...

```bash
#!/bin/bash

# Define the output file for the report
output_file="system_report.txt"

# Create or overwrite the output file
>"$output_file"    This redirection doesn't have a command. Move to its command (or use 'true' as

# Function to add a section header to the report
add_section() {
    echo "----------------------------------------" >>"$output_file"    Consider using { cmd1; cm
    echo "$1" >>"$output_file"
    echo "----------------------------------------" >>"$output_file"
}

# Add a header to the report
add_section "System Report - $(date)"

# Hostname and User Information
hostname >>"$output_file"    Consider using { cmd1; cmd2; } >> file instead of individual redirec
whoami >>"$output_file"
echo -e "\n" >>"$output_file"

# System Information
add_section "System Information"
uname -a >>"$output_file"
echo -e "\n" >>"$output_file"

# CPU Information
add_section "CPU Information"
lscpu >>"$output_file"
echo -e "\n" >>"$output_file"

# Memory Information
add_section "Memory Information"
free -h >>"$output_file"
echo -e "\n" >>"$output_file"

# Disk Usage
add_section "Disk Usage"
df -h >>"$output_file"
echo -e "\n" >>"$output_file"

# Network Information
add_section "Network Information"
ifconfig >>"$output_file"
echo -e "\n" >>"$output_file"

# Print a message indicating where the report is saved
```

Create project

```
moatsem:useful_scripts$> tree hello/
hello/
├── build
├── CMakeLists.txt
└── main.cpp

1 directory, 2 files
moatsem:useful_scripts$>
```

seful_scripts > $ create_project.sh > ...

```bash
#!/bin/bash

# Check if a project name was provided as an argument
if [ -z "$1" ]; then
    echo "Usage: ./create_cpp_project.sh <project_name>"
    exit 1
fi

project_name="$1"
project_dir="./$project_name"

# Create the project directory
mkdir -p "$project_dir"
cd "$project_dir" || exit

# Create CMakeLists.txt file
cat <<EOL >CMakeLists.txt
cmake_minimum_required(VERSION 3.10)
project($project_name)

set(CMAKE_CXX_STANDARD 17)

add_executable($project_name main.cpp)
EOL

# Create main.cpp file
cat <<EOL >main.cpp
#include <iostream>

int main() {
    std::cout << "Hello, $project_name!" << std::endl;
    return 0;
}
EOL

# Create a build directory
mkdir build

echo "C++ project '$project_name' has been created."
echo "To build the project, run the following commands:"
echo "cd $project_name/build"
echo "cmake .."
echo "make"
```

Test network speed

```bash
#!/bin/bash

# Check if speedtest-cli is installed
if ! command -v speedtest-cli &>/dev/null; then
    echo "speedtest-cli is not installed. Please install it using your package manager."
    exit 1
fi

# Run the speed test and capture the results
speedtest_result=$(speedtest-cli)

# Print the results
echo "Speed Test Results:"
echo "$speedtest_result"
```

```
moatsem:useful_scripts$> ./speed_test.sh
Speed Test Results:
Retrieving speedtest.net configuration...
Testing from Vodafone Egypt (41.69.76.160)...
Retrieving speedtest.net server list...
Selecting best server based on ping...
Hosted by we (Alexandria) [180.54 km]: 22.127 ms
Testing download speed.....................................................
Download: 32.44 Mbit/s
Testing upload speed.......................................................
Upload: 3.23 Mbit/s
moatsem:useful_scripts$>
```

Tasks

1- write bash script to create project based on Makefile

2- write bash script to generate systemd service file simple example

3- write bash script to download video from youtube

4- write a wiki bash script to help you on development
- give you example about c++ hello world
- give you example about python hello world
- give you example about linux commands
- give you example about bash hello world

5- write bash script to perform calculator operations