Walchand College of Engineering, Sangli

Walchand Linux Users' Group

WLUG
COMMUNITY | KNOWLEDGE | SHARE

OPEN SOURCE DAY

GIT

BASH

LEARN & CONTRIBUTE

Limited Seats	Hands-On

- Introduction to GIT
- Basic GIT commands
- Branching
- Advance Commands

Scan To Register

- Introduction to Shell
- Basic of Shell Scripting
- Loops and Functions
- Automation Scripts

3rd December

Venue: Main & Mini CCF

CONNECT WITH US

FREE FOR ALL

For any query :
+91 9420391705
+91 8550960012

Dr. P. G. Sonavane
Deputy Director
Walchand College of Engineering

Dr. M. A. Shah
HoD Computer Science and Engineering

Dr. A. J. Umbarkar
HoD Information Technology and Staff Advisor

Prof. A. R. Surve
Chairperson ECAC and Staff Advisor

Miss. D. A. Kolapkar
President
Walchand Linux Users' Group

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Shell Scripting

WLUG
COMMUNITY | KNOWLEDGE | SHARE

What is Shell?

- It is a command line interpreter.
- It acts as an interface between the kernel and the user.
- Shell is an environment in which we can run our commands, programs and shell scripts.

WLUG
COMMUNITY | KNOWLEDGE | SHARE

What is a Kernel?

- Kernel is the core component of the operating system.
- Main layer between the software and underlying computer hardware.
- Carries out many operations such as task scheduling, memory management etc.

WLUG
COMMUNITY | KNOWLEDGE | SHARE

What is Shell Scripting?

- A shell script is a list of commands in a program that is run by the Unix shell which is a command line interpreter.

- The different operations performed by shell scripts are program execution, file manipulation and text printing.

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Why Shell Scripting?

- Increases Efficiency
- Easy to learn
- Helps us automate our tasks
- Easily available on all linux based platforms and easier to debug

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Interpreting languages

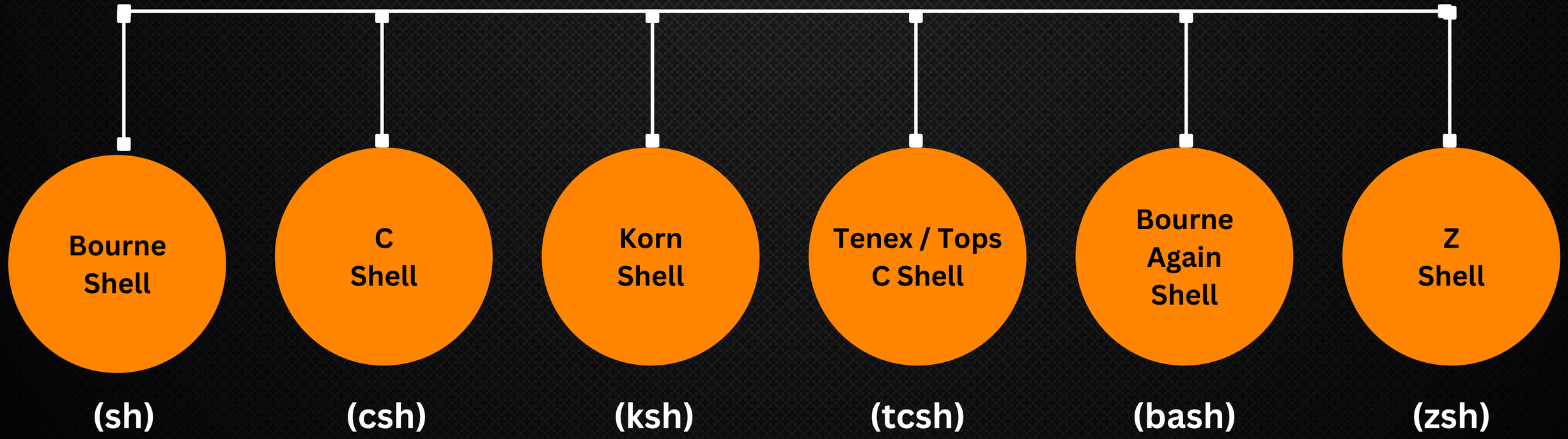
- Interpreter-based
- Use to combine existing components and automate specific task
- Do not create extra files
- Eg. Bash,Ruby,Python

Compiling languages

- Compiler-based
- Used to develop an application or software
- Create extra files ex: *.exe* file
- Eg. C,C++,Java

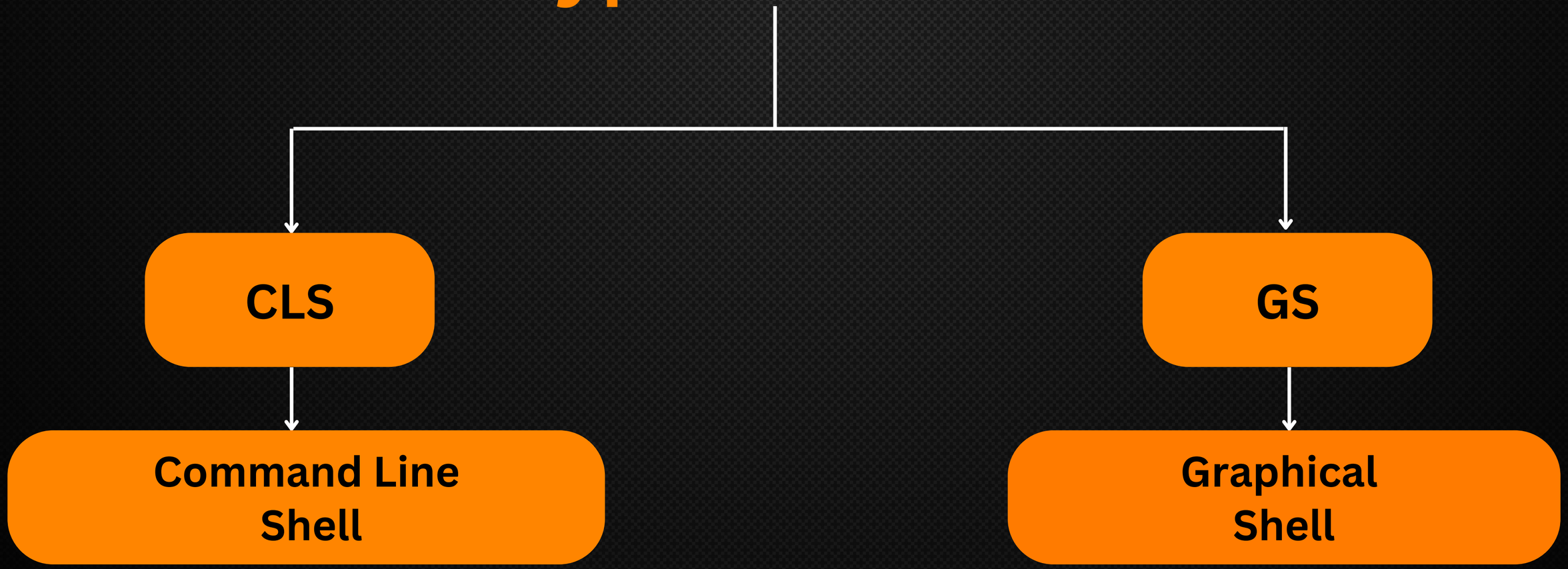
WLUG
COMMUNITY | KNOWLEDGE | SHARE

History of Shells



WLUG
COMMUNITY | KNOWLEDGE | SHARE

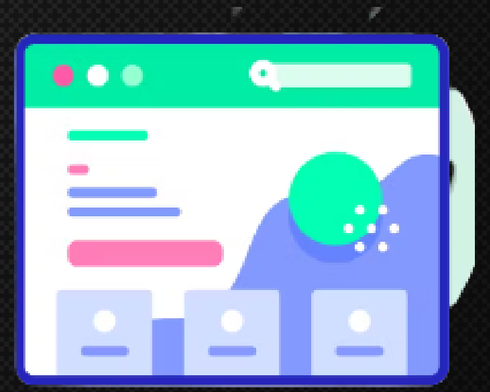
Types of Shell



CLI and GUI

Provides CLI

Allows the user to interact with the system using commands



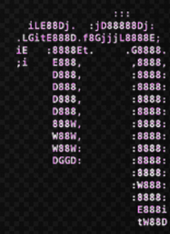
Provides GUI

Allows the user to interact with the system using graphics

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Text Editors

- VS CODE
- GEDIT
- NANO
- VIM
- KWRITE
- EMACS



WLUG
COMMUNITY | KNOWLEDGE | SHARE

File Permission

- Owner permissions
- Group/Directory permissions
- Other (world) permissions

To check file permission 'ls' command is used with flag -l

$ls -l [file_name]

read(-r)

write(-w)

execute(-x)

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Change File permission

- chmod command is used to define file permissions.
- chmod stands for Change Mode
- 2 ways to use chmod :

Symbolic Mode

Absolute Mode

Symbolic Mode

+	Adds the designated permission(s) to a file or directory.
-	Removes the designated permission(s) from a file or directory.
=	Sets the designated permission(s).

Example: $chmod +x testfile

Absolute Mode

- For absolute permission each permission is assigned a numerical value, and the total of each set of permission provides permission for that user

No permission	---
Execute permission	--x
Write permission	-w-

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Absolute Mode

Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
Read permission	r--
Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
Read and write permission: 4 (read) + 2 (write) = 6	rw-
All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwx

Example: $ chmod 755 testfile

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Shebang

- This sequence of character is called as Shebang.
- Used at the start of the script.
- Used to specify the interpreter.
- Syntax : #! interpreter [arguments]

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Variables

- Variables are an important aspect of any programming language, as it is used to store any required data.

Types of Variables

There are two types of variables in bash:

Environmental Variables

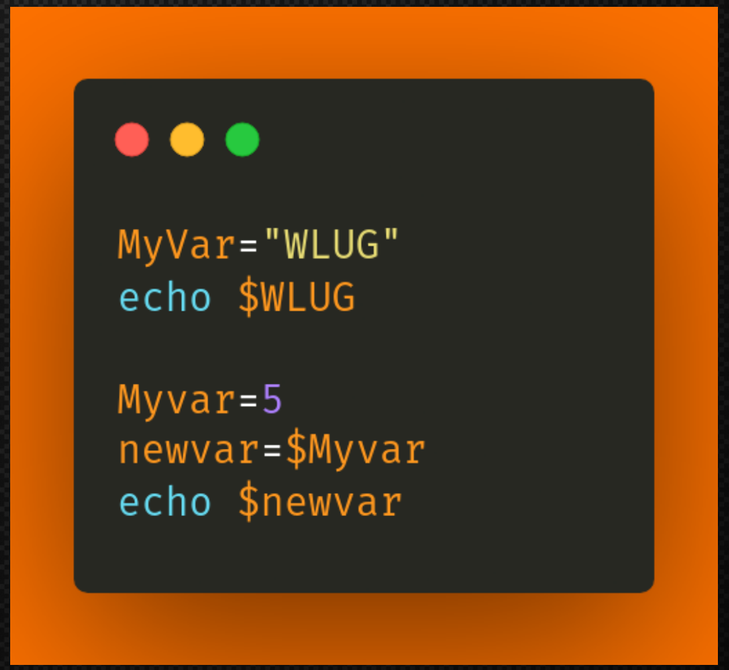
User-Defined Variables

Environmental Variables

- Pre-defined Variable
- set or env command can be used to see the list of Environmental Variables
- Example
  - BASH_VERSION
  - BASH
  - PWD
  - OSTYPE
  - HOME
  - HOSTNAME
  - PATH
  - COLUMNS
  - USER
  - LANG

User-Defined Variable

- Variables that we define
- Working with variables
  - Direct assigning values
  - Set value



Operations on Variables

- read- To take input for variables
- Length of a variable

```
#read
read a
echo $a
read -p "Do you like music?" ans

#Length of a variable
a=Hello
echo ${#a}
```

Special Variables

- These are the special shell variables that are set internally by the shell and which are available for the user:

Variables	Description
$0	The name of the current script.
$$	The process id of the current shell.
$@	Stores the list of arguments as an array.
$?	Specifies the exit status of the last command or the most recent execution process.
$!	Shows the pid of the last background command.

Comment

- Annotation in code/explanation

```
#Single Line Comment

: '
This
is
Multi-Line
Comment
'

<<COMMENT
Another way to write
Multi-Line Comment
COMMENT
```

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Basic Operators

- Some basic operators in bash/shell scripting:

Arithmetic Operators

File Test Operators

Boolean Operators

Conditional Operators

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Arithmetic Operators

Operator	Description
+	Addition
-	Substraction
*	Multiplication
/	Division
**	Exponentiation
%	Modulo

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Arithmetic Operators

Operator	Description
+=	Increment
-=	Decrement
*=	Multiply by Constant
/=	Divide by Constant
%=	Remainder of Dividing by Constant

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Conditional Operators

Operator	Description
-eq (==)	Returns true if two numbers are equivalent
-lt (<)	Returns true if a number is less than another number
-gt (>)	Returns true if a number is greater than another number
-le (<=)	Returns true if two strings are equivalent
-ge (>=)	Returns true if two strings are not equivalent
-ne (!=)	Returns true if two numbers are not equivalent

Conditional Statements

- if statement

  Syntax :

```
if [ expression ] # (( expression ))
then
    <statement>
fi
```

Conditional Statements

WLUG
COMMUNITY | KNOWLEDGE | SHARE

if-else statement

```
if [ expression ]
then
    statement1
else
    statement2
fi
```

if-elif-else statement

```
if [ expression ]
then
    statement1
elif [ expression ]
then
    statement2
else
    statement3
fi
```

nested if statement

```
if [ expression ]
then
    if [ expression ]
    then
        statement1
    fi
fi
```

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Boolean Operators

Operator	Description	Example
!	logical negation	[! false] is true.
-o	logical OR	[$a -lt 20 -o $b -gt 100] is true.
-a	logical AND.	[$a -lt 20 -a $b -gt 100] is false.

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Array in Shell Scripts

- Data Structure consisting of collection of Elements
- Initialization of an array

```
a=('Welcome' 'To' 'Bash' 5)

echo ${a[0]} ${a[2]} ${a[1]}

echo ${a[@]}

echo ${!a[@]}

echo ${#a[@]}
```

Loops and Functions

WLUG
COMMUNITY | KNOWLEDGE | SHARE

For Loop

- The for loop is used when you are looping through a range of variables.
- Syntax:

```
for (( START; END; OPERATION ))
do
    command1
    command2
done
```

While Loop

- The while loop is used when you are looping until some condition is false
- Syntax:

Until Loop

- Sometimes you need to execute a set of commands until a condition becomes true.
  This is exactly the opposite of a while loop.
- Syntax:

Loop Control

- Break:
  The break statement is used to terminate the execution of the entire loop.

- Continue:
  The continue statement is used to skip the current iteration rather than breaking the entire loop.

Functions

- A function is a block of structured, reusable code that is used to perform a single, related task whenever it is required.

- Syntax :

```
function_name () {

list of commands

}
```

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Automation Demo

WLUG
COMMUNITY | KNOWLEDGE | SHARE

Thank You!!