



Shell Programming

By

Kishore Kumar Boddu

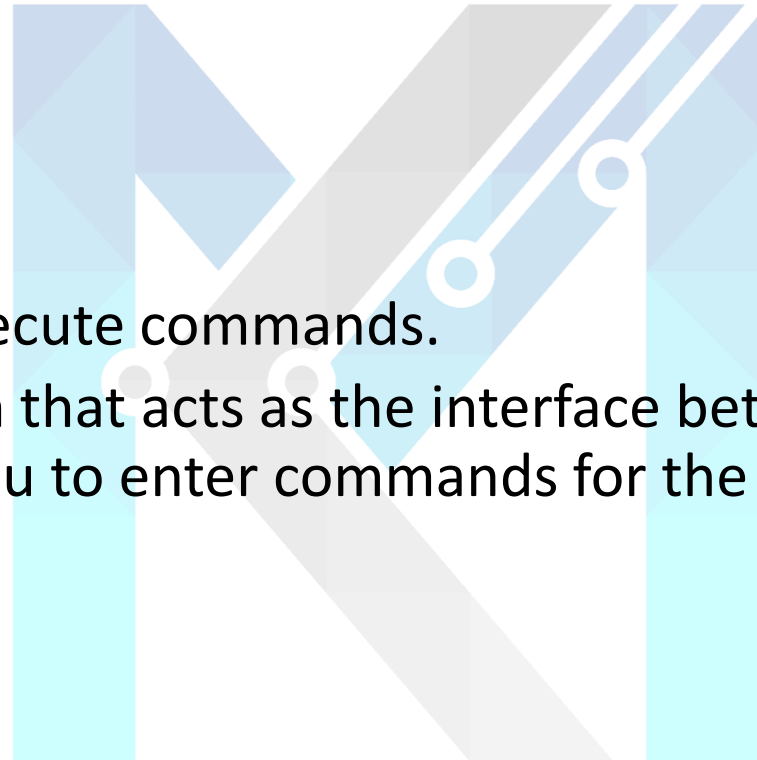
Content

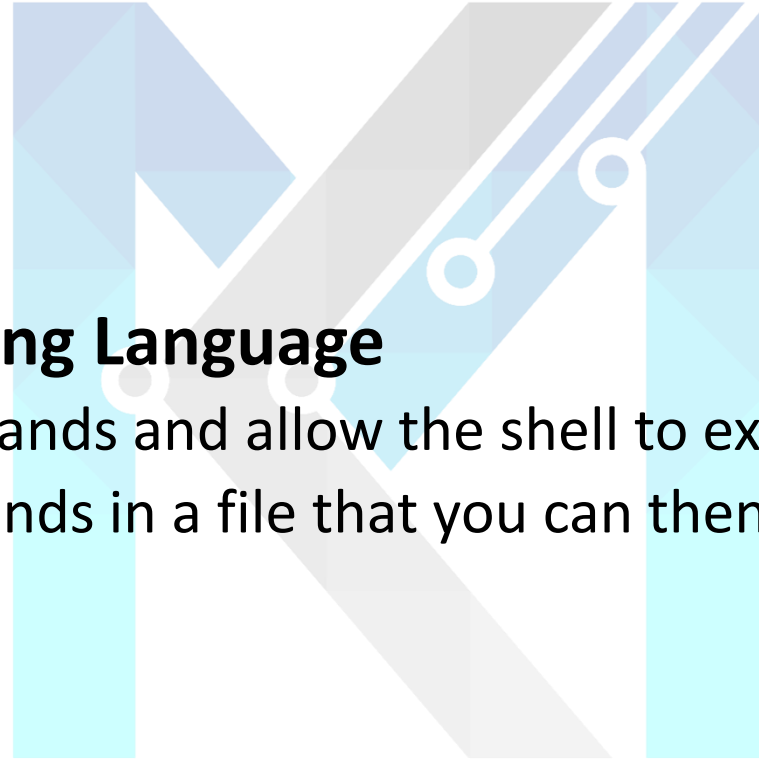
- Shell Basics
- Shell Scripts
- Shell Syntax
- Variables
- Substitution
- Conditions
- Program Control
- Lists
- Functions



What is Shell?

- ***Shell*** is a tool to execute commands.
- A ***shell*** is a program that acts as the interface between you and the Linux system, allowing you to enter commands for the operating system to execute.





Shell as a Programming Language

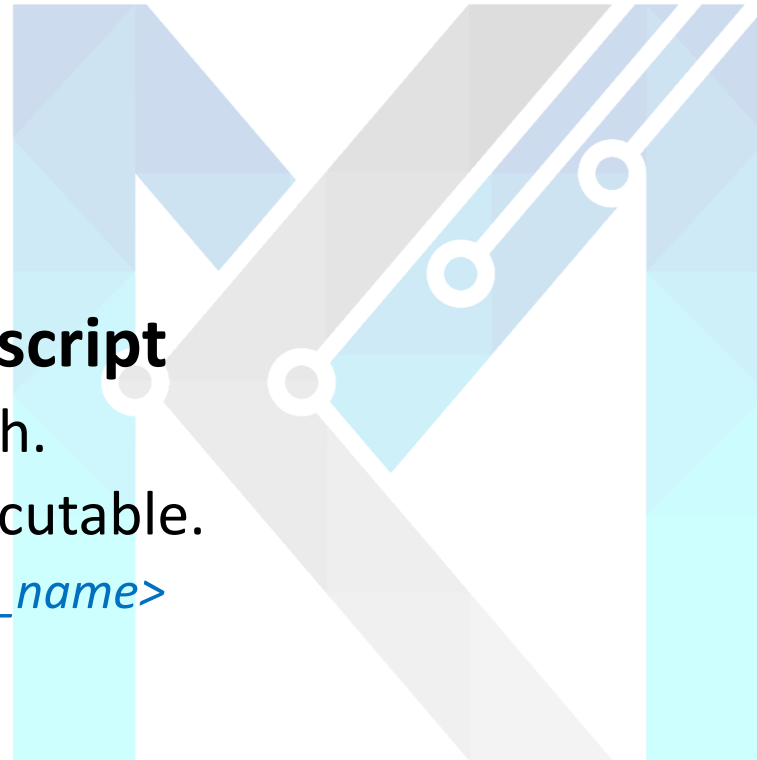
- Sequence of commands and allow the shell to execute them interactively.
- Store those commands in a file that you can then invoke as a program.

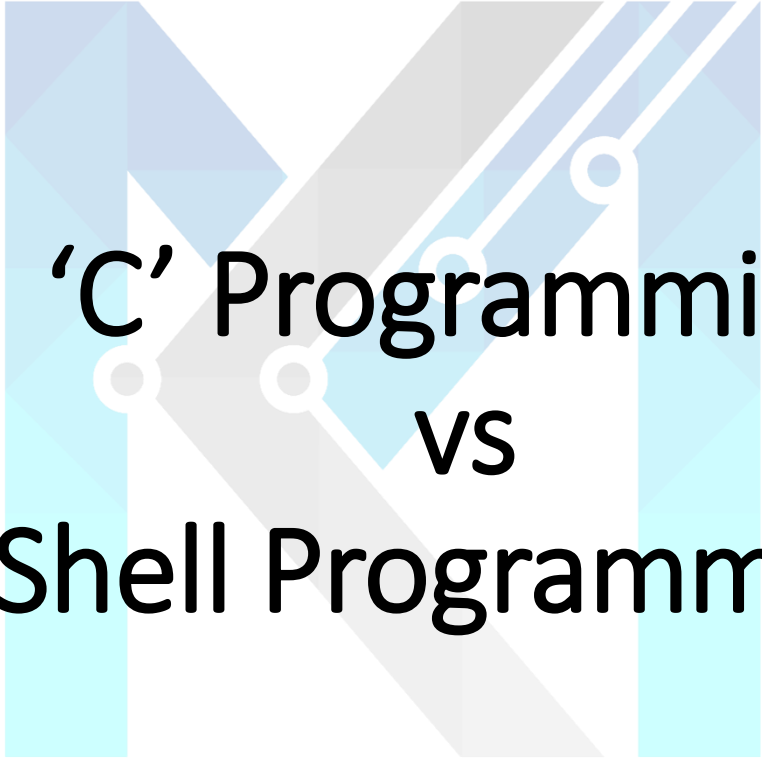
Shell introduction

How to write a Shell script

- Shell extension is .sh.
- Making a script executable.

\$ chmod a+x <script_name>





'C' Programming vs Shell Programming

Comments

C Programming

Single line Comment:

//

Multiple line comments:

/*

....

....

.... */

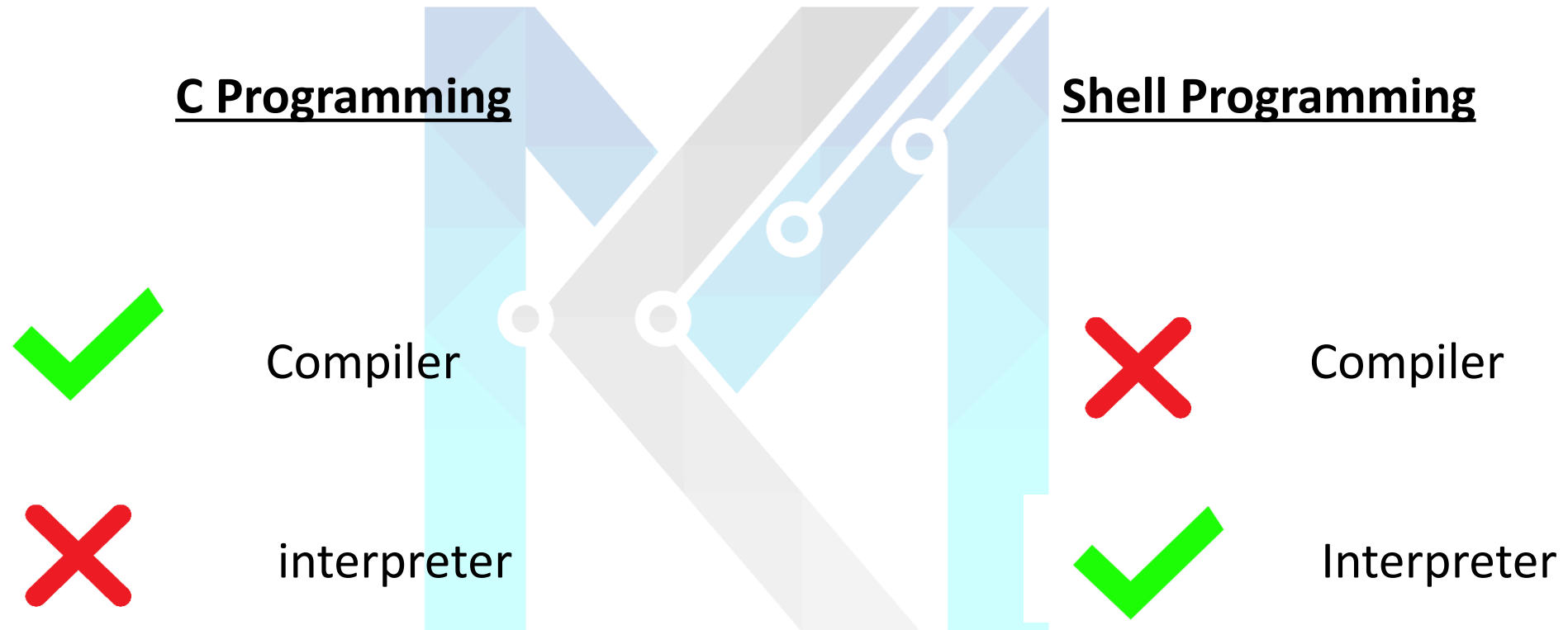
Shell Programming

indicate single line comment

#!/bin/sh indicate which shell
run?

That is bash, csh, tch etc...

Translator



Standard I/O

C Programming

`printf("Hello World\n")`

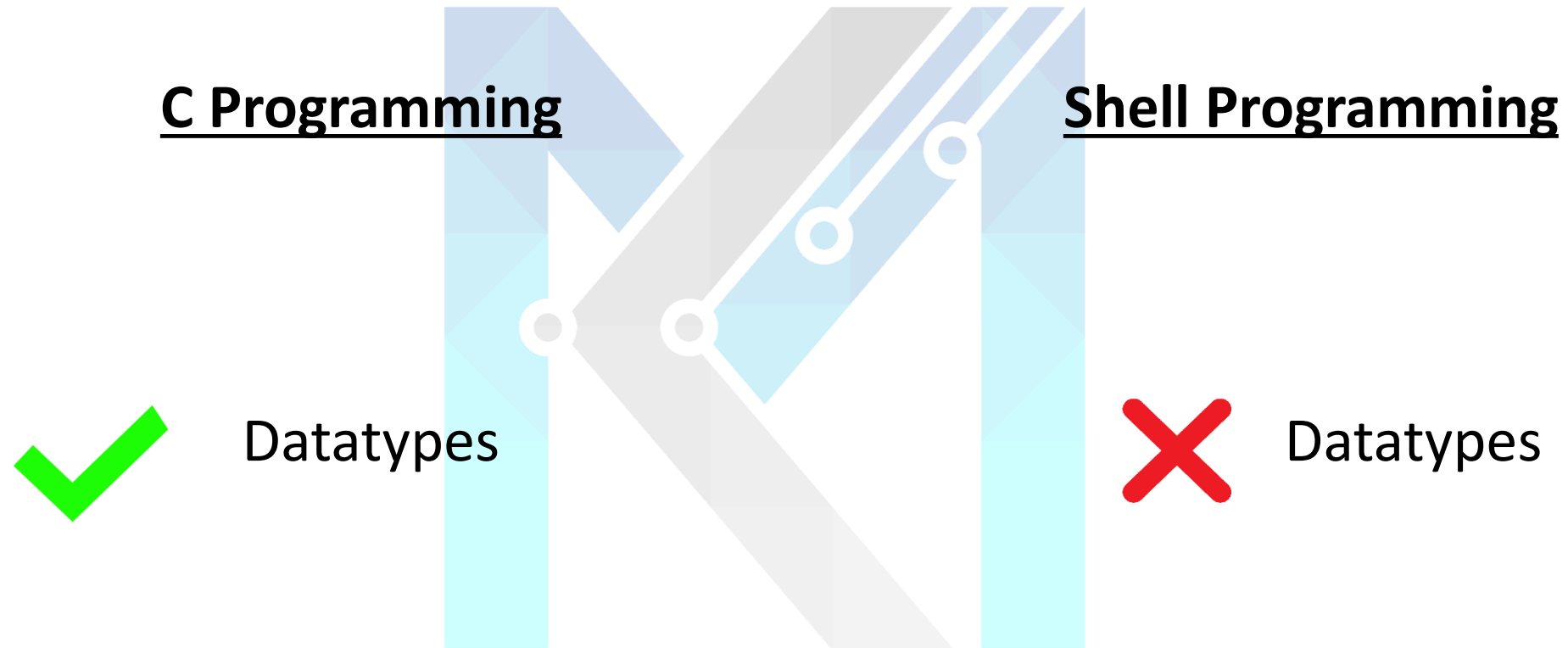
`scanf()`

Shell Programming

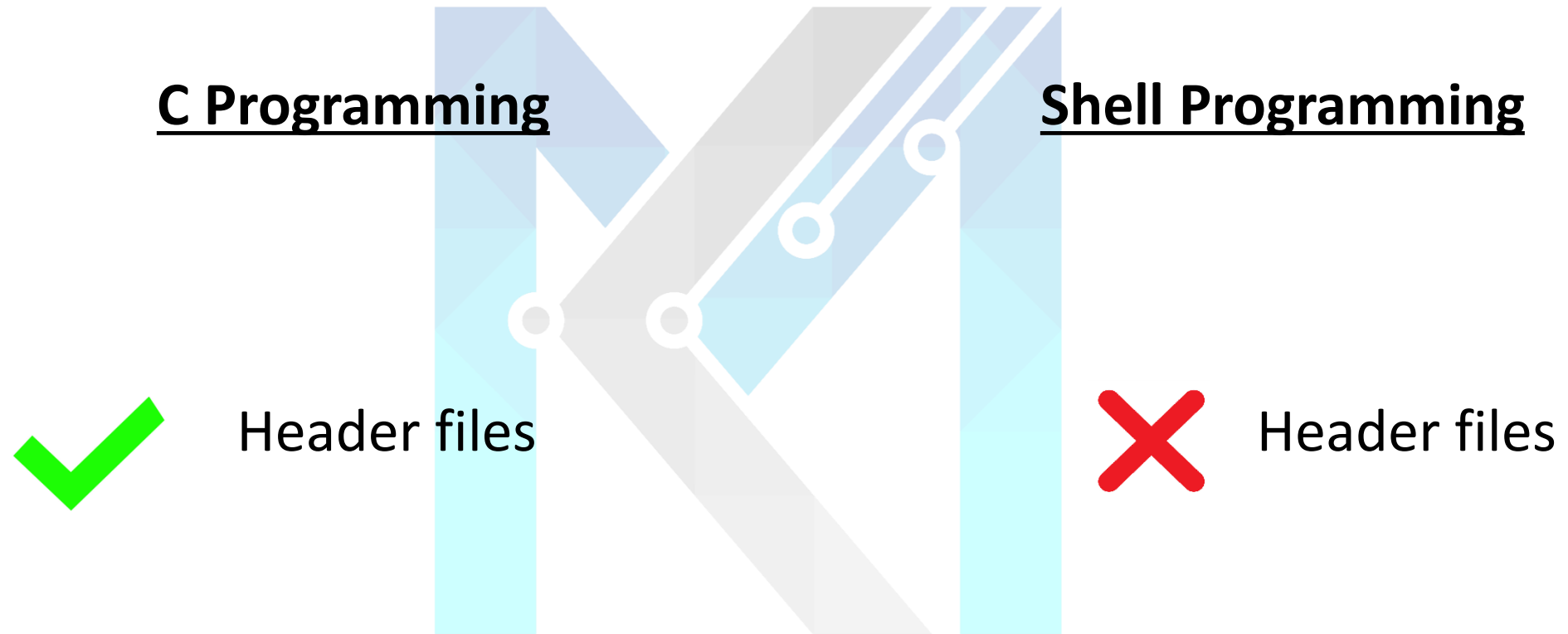
`echo "Hello World"`

`read`

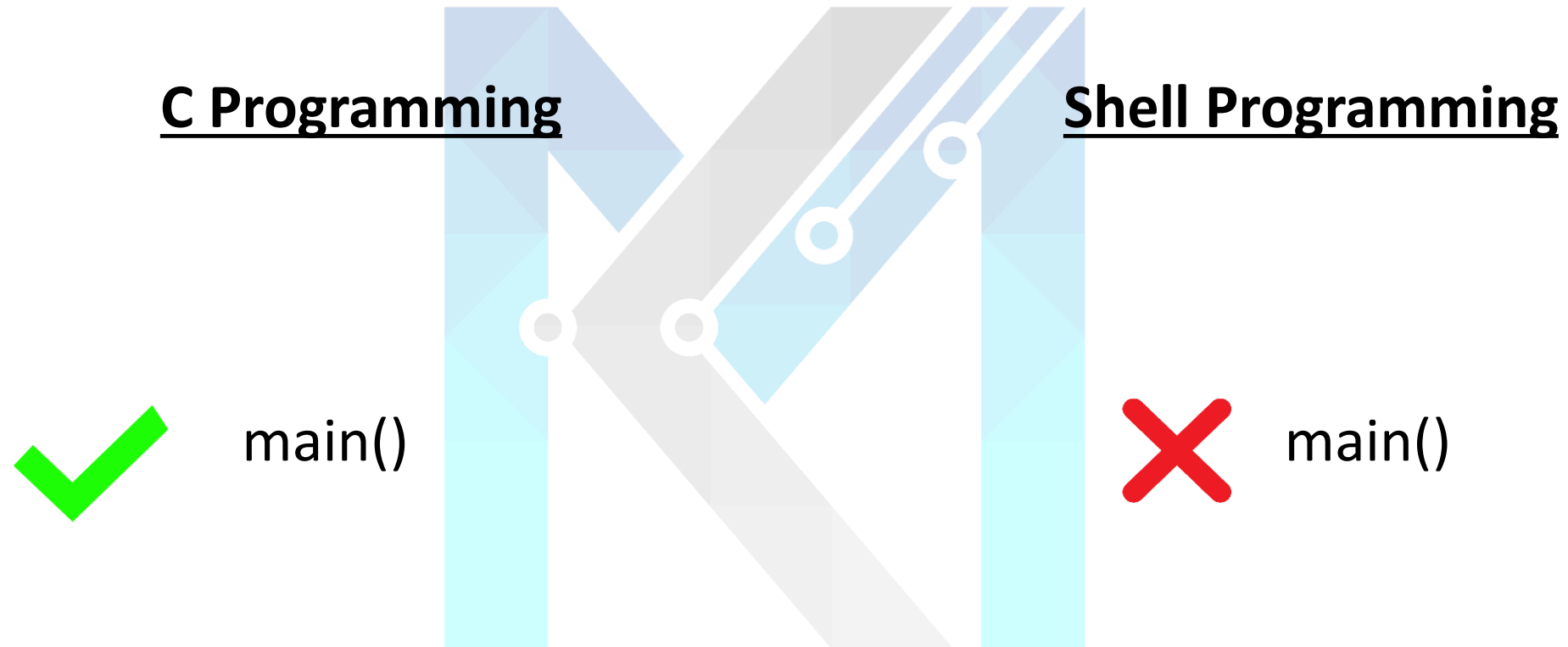
Data types



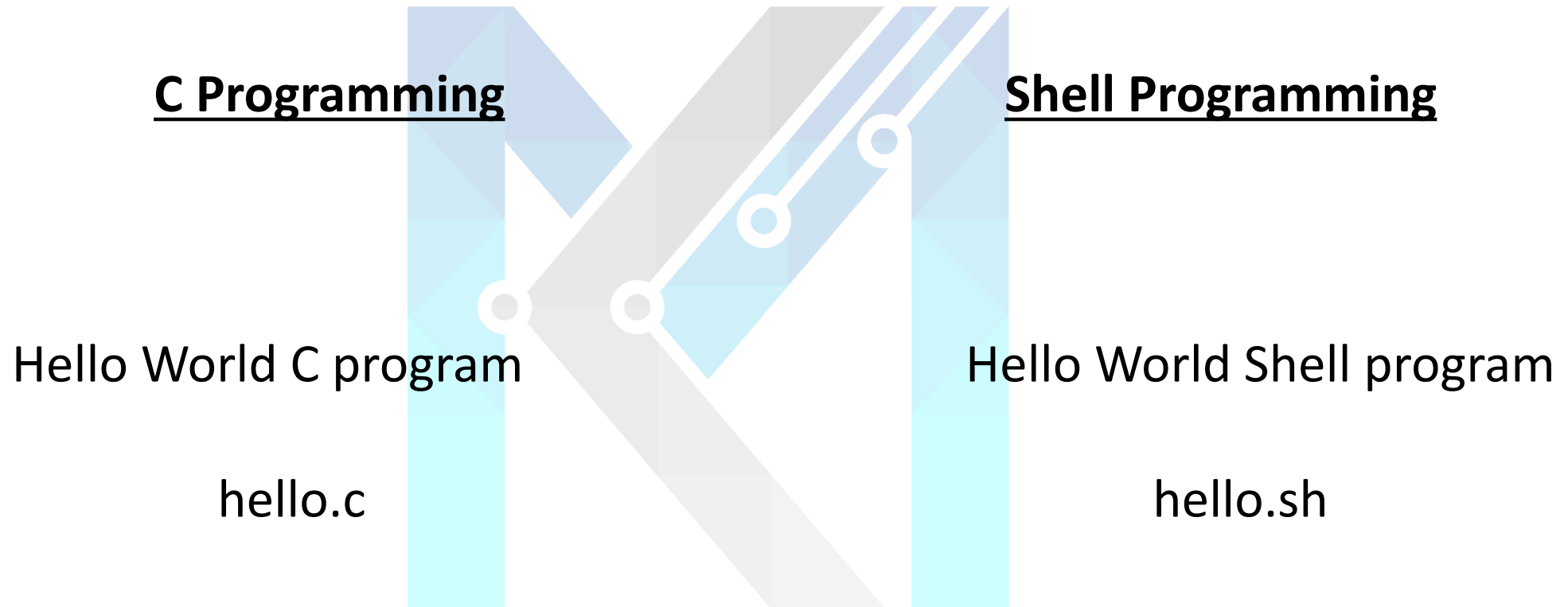
Header files



Starting function



Hands-on Session



Summary

Parameter	'C' Programming	Shell Programming
main()	✓	✗
Header Files	✓	✗
Data types	✓	✗
Compiler	✓	✗
Interpreter	✗	✓
Std. Output	printf()	echo
Std. Input	scanf()	read

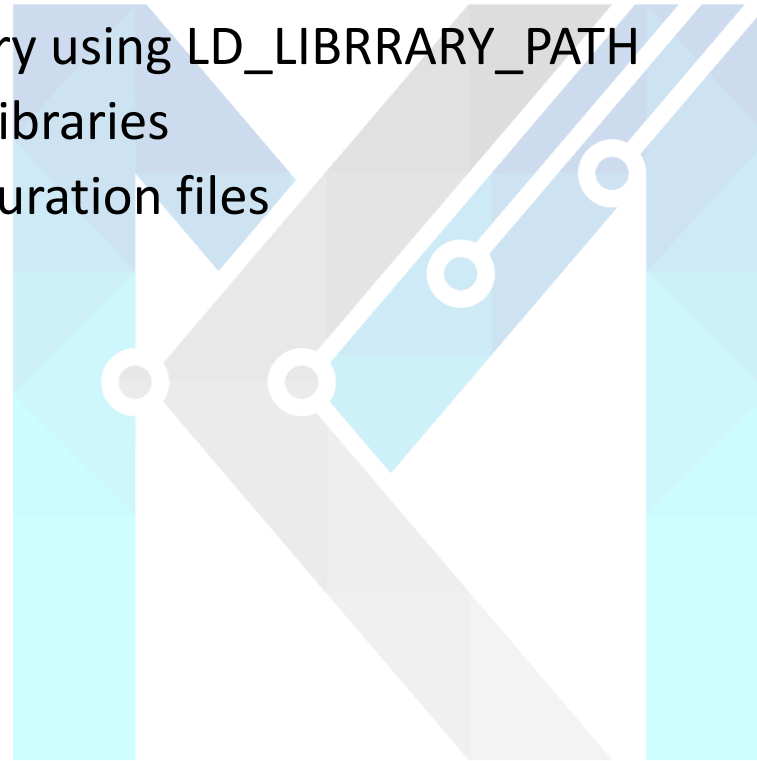


Shell Programming

Environment Variables

Environment variables usage

- Application need environment variables for
 - To find 3rd Party library using LD_LIBRARY_PATH
 - To find user defined libraries
 - To find system configuration files



Types of Environment variables

- Shells let the **user define *variables***.
They can be reused in shell commands.
Convention: lower case names
- You can also define ***environment variables***: variables that are also visible within scripts or executables called from the shell.
Convention: upper case names.
- **env**
Lists all defined environment variables and their value.



User defined Environment Variables

Environment Variables

Create a user defined Environment variable

```
export <variable_name>=<value>
```



System defined Environment Variables

```
$ echo $USER
```

Shell variables examples

Shell variables (bash)

- `projdir=/home/km/KM_GITHUB`
`ls -la $projdir; cd $projdir`

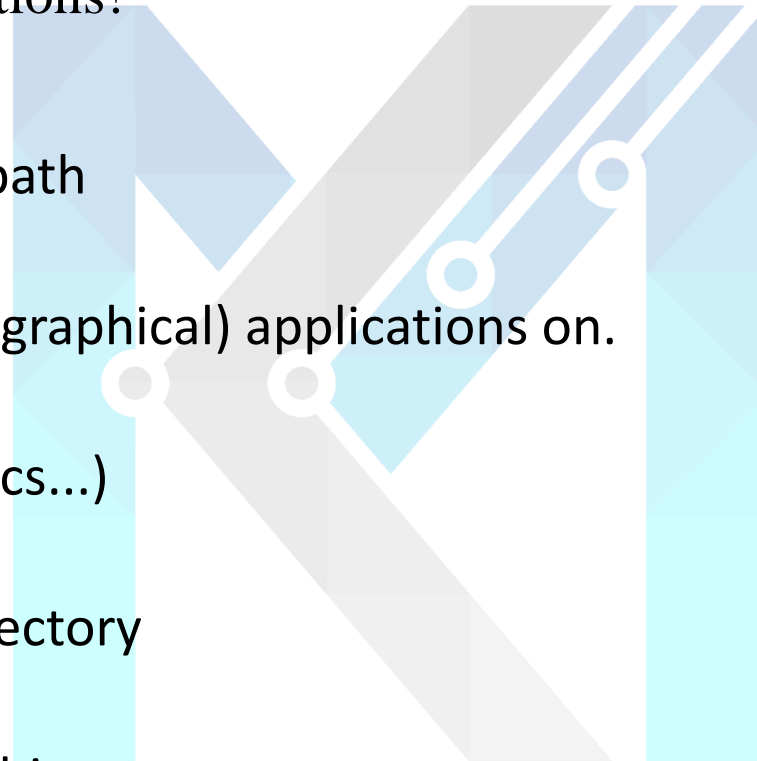
Environment variables (bash)

- `cd $HOME`
- `export DEBUG=1`
`./find_extraterrestrial_life`
(displays debug information if **DEBUG** is set)

Main standard environment variables

Used by lots of applications!

- **LD_LIBRARY_PATH**
Shared library search path
- **DISPLAY**
Screen id to display X (graphical) applications on.
- **EDITOR**
Default editor (vi, emacs...)
- **HOME**
Current user home directory
- **HOSTNAME**
Name of the local machine



MANPATH

Manual page search path

PATH

Command search path

PRINTER

Default printer name

SHELL

Current shell name

TERM

Current terminal type

USER

Current user name

PATH environment variables

- **PATH**

Specifies the shell search order for commands

`/home/abox/bin:/usr/local/bin:/usr/kerberos/bin:/usr/bin:/bin:/usr/X11R6/bin:/bin:/usr/bin`

- **LD_LIBRARY_PATH**

Specifies the shared library (binary code libraries shared by applications, like the C library) search order for `ld`

`/usr/local/lib:/usr/lib:/lib:/usr/X11R6/lib`

- **MANPATH**

Specifies the search order for manual pages

`/usr/local/man:/usr/share/man`

Alias

Shells let you define command *aliases*: shortcuts for commands you use very frequently.

Examples

- `alias ls='ls -la'`
Useful to always run commands with default arguments.
- `alias rm='rm -i'`
Useful to make `rm` always ask for confirmation.
- `alias frd='find_rambaldi_device --asap --risky'`
Useful to replace very long and frequent commands.
- `alias cia='. /home/sydney/env/cia.sh'`
Useful to set an environment in a quick way
(`.` is a shell command to execute the content of a shell script).

Add “abc” user defined variable in last line of ~/.bashrc file

Example: `$ export abc=123`



Add Greeting message in last line of ~/.bashrc file.

Example: `echo "Welcome to Kernel Masters"`



Shell Programming

Hello World Program

script2.c – Hello World Program

```
#!/bin/sh  
#Hello world script program  
echo "Hello World"  
read var  
echo "var value:$var"
```

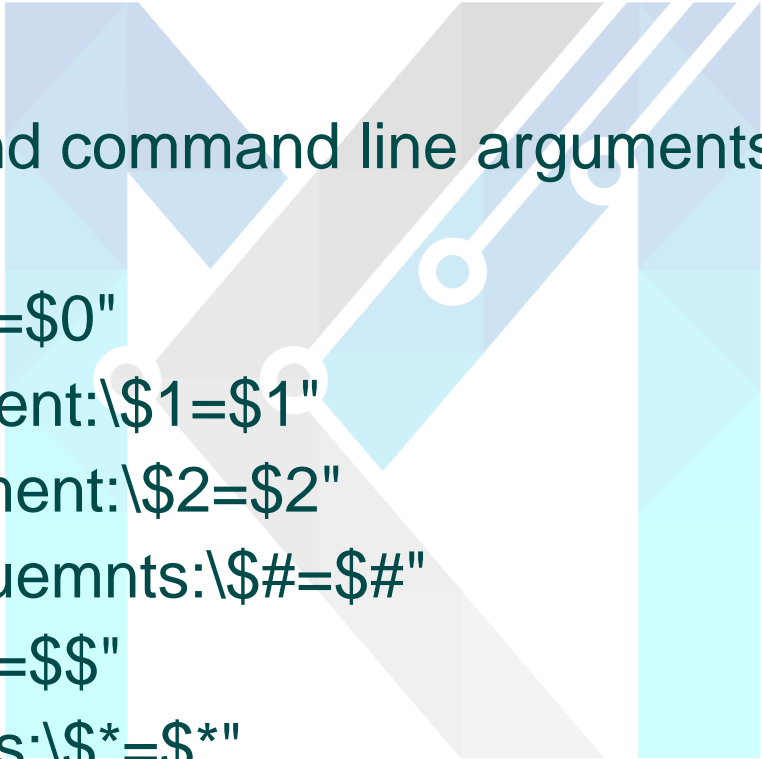




Shell Programming

Shell Command Line arguments

script3.c – Shell command line arguments



```
#!/bin/sh
# Shell parameters and command line arguments
IFS='*'
echo "Shell name:\$0=$0"
echo "shell 1st argument:\$1=$1"
echo "shell 2nd argument:\$2=$2"
echo "shell no. of arguemnts:\$#=$#"
echo "shell pid no:\$$=$$"
echo "list all paramters:\$*=$*"
echo "list all paramters:\$@=$@"
```



Shell Programming

Shell Quotations & Substitutions

Quotations & Substitutions

\$ variable expression in

“ ” (Double quotes) then substitution takes place

‘ ’ (single quotes) then no substitution takes place

**** (Escape Character) remove the special meaning of the \$ symbol by prefacing

script2.c – Shell Quotations & Substitutions

```
#!/bin/sh
```

```
# Shell Quotations & Substitutions
```

```
var="kernel"
```

```
echo $myvar
```

```
echo "$myvar"
```

```
echo '$myvar'
```

```
echo \ $myvar
```

```
echo \ $myvar: $myvar
```

Output:

kernel

kernel

\$myvar

\$myvar

\$myvar:kernel



Shell Programming

Shell Conditions

Conditions: String Comparison

String Comparison	Result
<code>string1 = string2</code>	True if the strings are equal
<code>string1 != string2</code>	True if the strings are not equal
<code>-n string</code>	True if the string is not null
<code>-z string</code>	True if the string is null (an empty string)

Conditions: Arithmetic Comparison

Arithmetic Comparison	Result
expression1 -eq expression2	True if the expressions are equal
expression1 -ne expression2	True if the expressions are not equal
expression1 -gt expression2	True if expression1 is greater than expression2
expression1 -ge expression2	True if expression1 is greater than or equal to expression2
expression1 -lt expression2	True if expression1 is less than expression2
expression1 -le expression2	True if expression1 is less than or equal to expression2
! expression	True if the expression is false, and vice versa

Conditions: File Comparison

File Conditional	Result
-d file	True if the file is a directory
-e file	True if the file exists. Note that historically the -e option has not been portable, so -f is usually used.
-f file	True if the file is a regular file
-g file	True if set-group-id is set on file
-r file	True if the file is readable
-s file	True if the file has nonzero size
-u file	True if set-user-id is set on file
-w file	True if the file is writable
-x file	True if the file is executable