Shell Programming

Dr. Shreeya Swagatika Sahoo
Assistant Professor
ITER S’O’A (Deemed to be University)

Introduction to shell

Kernel

Shell

Terminal

What is Kernel?

The kernel is a computer program that is the core of a computers operating system, with complete control over everything in the system. It manages the following resources of the Linux system

- File management
- Process management
- I/O management
- Memory management
- Device management etc.

What is Terminal

- A program which is responsible for providing an interface to a user so that he/she can access the shell.
- It allows users to enter commands and see the output of those commands in a text-based interface.

What is shell

- A shell is a special user program that provides an interface for the user to use operating system services.
- Shell accepts human-readable commands from users and converts them into something which the kernel can understand.
- It is a command language interpreter that executes commands read from input devices such as keyboards or from files.
- The shell gets started when the user logs in or starts the terminal.
- Shell is broadly classified into two categories –
  - Command Line Shell
  - Graphical shell

Command Line shell

- ➢ Shell can be accessed by users using a command line interface.
- ➢ A special program called Terminal in Linux/macOS, or Command Prompt in Windows OS is provided to type in the human-readable commands and then it is being executed.

Graphical shells

- ➢ Graphical shell provide means for manipulating programs based on the graphical user interface (GUI), by allowing for operations such as opening, closing, moving, and resizing windows, as well as switching focus between windows.

There are several shells are available for Linux systems

- **BASH (Bourne Again SHell)** – It is the most widely used shell in Linux systems. It is used as default login shell in Linux systems and in macOS. It can also be installed on Windows OS.
- **CSH (C SHell)** – The C shell's syntax and its usage are very similar to the C programming language.
- **KSH (Korn SHell)** – The Korn Shell was also the base for the POSIX Shell standard specifications etc.

Shell scripting

- shells are interactive, which means they accept commands as input from users and execute them to avoid this repetitive work.
- Each shell script is saved with ‘.sh’ file extension e.g., file1.sh.

A shell script comprises the following elements –

1. Shell Keywords – if, else, break etc.
2. Shell commands – cd, ls, echo, pwd, touch etc.
3. Functions
4. Control flow – if..then..else, case and shell loops etc.

Why shell scripting

- ➢To avoid repetitive work and automation
- ➢System admins use shell scripting for routine backups.
- ➢System monitoring
- ➢Adding new functionality to the shell etc.

Creating a Shell Script

- Open the terminal, navigate to the folder where you want to store the shell script. Shell scripts end with the extension “.sh”.
- Script file is not executable by default, give the executable permission to this file by using **chmod** command.
- Open this shell script with any text editor and add some commands.
- Save the changes, and run the shell script by typing in ./file1.sh

Variables in Shell Script

There are two types of variables:

System Defined variables

User-Defined Variables.

- ➢System-defined variables, also called environment variables, are generally Capitalised. (Use the printenv command.)

  Example: echo $USER

- ➢User-Defined variables are set by the user, and they exist only during script execution.
- ➢A variable can be defined by simply typing its name and assigning a value with = sign and access a variable by adding a $ before the variable name.

  Example : vname= "abc"

  ```
  echo $vname
  ```

Rules for variable definition

- A variable name could contain any alphabet (a-z, A-Z), any digits (0-9), and an underscore (_).
- A variable name must start with an alphabet or underscore. It can never start with a number.

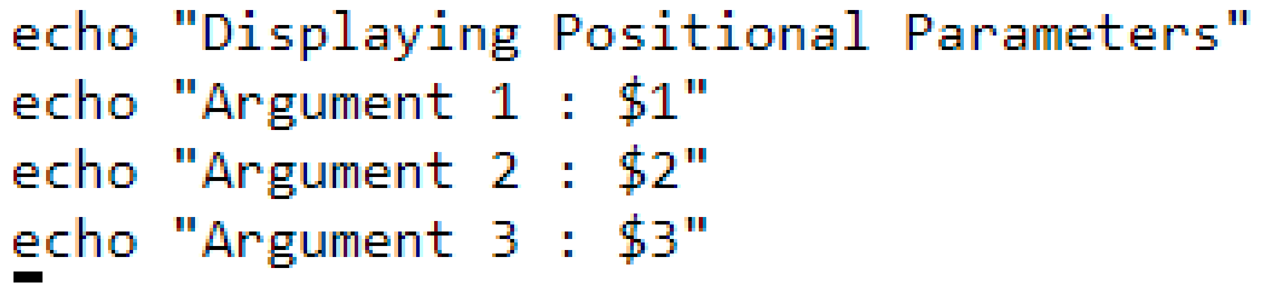
  Example: AB, _AB, AB4
- A Variable data could be accessed by appending the variable name with ‘$’

```
VAR_1="Welcome"
VAR_2="CSE"
echo "$VAR_1$VAR_2"
```

Command Line argument

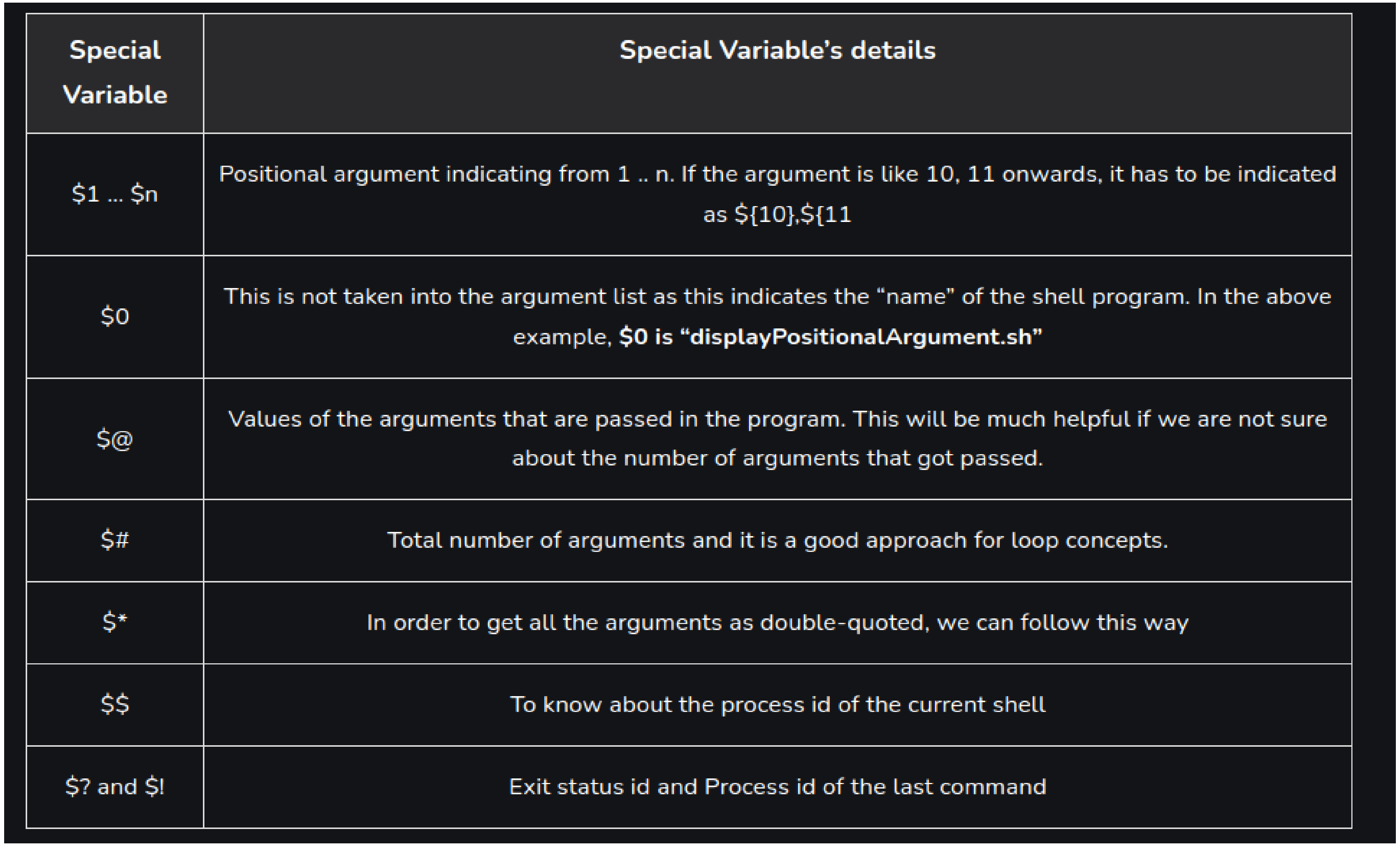
Arguments are inputs that are necessary to process the flow. Instead of getting input from a shell program or assigning it to the program, the arguments are passed in the execution part.

Example: file1.sh



```
echo "Displaying Positional Parameters"
echo "Argument 1 : $1"
echo "Argument 2 : $2"
echo "Argument 3 : $3"
```

```
./file1.sh welcome to   CSE
           $1      $2   $3
./file1.sh welcome to “CSE DEPT”
            $1      $2     $3
```



Special Variable	Special Variable’s details
$1 ... $n	Positional argument indicating from 1 .. n. If the argument is like 10, 11 onwards, it has to be indicated as ${10},${11
$0	This is not taken into the argument list as this indicates the “name” of the shell program. In the above example, **$0 is “displayPositionalArgument.sh”**
$@	Values of the arguments that are passed in the program. This will be much helpful if we are not sure about the number of arguments that got passed.
$#	Total number of arguments and it is a good approach for loop concepts.
$*	In order to get all the arguments as double-quoted, we can follow this way
$$	To know about the process id of the current shell
$? and $!	Exit status id and Process id of the last command

Thank u