hitbullseye

Shell Scripting Basics, Types of Shells

The tutorial for shell scripting explains both the fundamental and more complex ideas. Our training on shell scripting is intended for both experts and beginners. An open-source operating system is Shell Scripting. Our Shell Scripting lesson covers all scripting-related subjects, such as loops, parameters for scripts, shifting through parameters, source, getopts, case, eval, and let statements.

What is Shell Scripting

In Linux, shells like bash and Korn support programming constructs which are saved as scripts. These scripts become shell commands and hence many Linux commands are scripts. A system administrator should have a little knowledge about scripting to understand how their servers and applications are started, upgraded, maintained or removed and to understand how a user environment is built.

How to determine Shell

You can get the name of your shell prompt, with the following command:

Syntax:

1. echo $SHELL

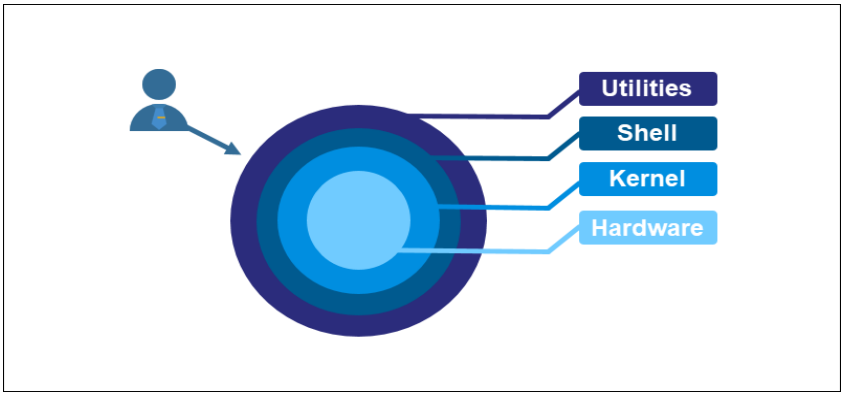
Look at the above snapshot, with the help of the above command we got the name of our shell which is **'bash'.** The $ sign stands for a shell variable; echo will return the text whatever you typed in.

Types of Shells:

Linux comes with several different shells. Although a distribution has a default shell, users can change to another type or install a new one. **This Lecture showcases the importance and features of eight different Linux shells.**

What is a Linux Shell and Why is it Important?

A **shell** is a command-line interpreter program that parses and sends commands to the operating system. This program represents an operating system's interactive interface and the kernel's outermost layer (or shell). It allows users and programs to send signals and expose an operating system's low-level utilities.



The terminal program (or terminal emulator) enables interaction with the system's utilities. When we run any command in the terminal, such as ls or cat, the shell parses, evaluates, searches for, and executes the corresponding program, if found.

Types of Linux Shells

Linux offers different shell types for addressing various problems through unique features. The shells developed alongside Unix often borrowed features from one another as development progressed.

Below is a brief **overview of different shell types and their features**.

1. Bourne Shell (sh)

The **Bourne shell** was the first default shell on Unix systems, released in 1979. The shell program name is **sh**, and the traditional location is */bin/sh*. The prompt switches to **$**, while the root prompt is **#**.

```
$ ps -p $$ -ocomm=
sh
$ foo=bar
$ echo $foo
bar
$ sudo -i
[sudo] password for kb:
# whoami
root
#
```

The Bourne shell quickly became popular because it is **compact** and **fast**. However, **sh** lacks some standard features, such as:

- Logical and arithmetic expansion.
- Command history.
- Other comprehensive features, such as autocomplete.

Modern Unix-like systems have the */bin/sh* executable file. The program does not start the Bourne shell but acts as an executable file pointing to the **default system shell**.

```
kb@phoenixNAP:~$ hostnamectl | grep "Operating System"
Operating System: Ubuntu 22.04.1 LTS
kb@phoenixNAP:~$ ls -l /bin/sh
lrwxrwxrwx 1 root root 4 cen  7 12:03 /bin/sh -> dash
kb@phoenixNAP:~$
```

```
[kb@phoenixNAP ~]$ hostnamectl | grep "Operating System"
  Operating System: CentOS Linux 7 (Core)
[kb@phoenixNAP ~]$ ls -l /bin/sh
lrwxrwxrwx. 1 root root 4 Oct 21 09:08 /bin/sh -> bash
[kb@phoenixNAP ~]$
```

2. C Shell (csh)

The **C shell** (**csh**) is a Linux shell from the late 1970s whose main objective was to improve interactive use and mimic the C language. Since the Linux kernel is predominantly written in C, the shell aims to provide stylistic consistency across the system.

The path to the C shell executable is */bin/csh*. The prompt uses **%** for regular users and **#** for the root user.

```
% ps -p $$ -ocomm=
csh
% set foo=bar
% echo $foo
bar
% sudo -i
[sudo] password for kb:
# whoami
root
#
```

New interactive features included:

- History of the previous command.
- User-defined aliases for programs.
- Relative home directory (~).
- Built-in expression grammar.

The main drawbacks of the C shell are:

- Syntax inconsistencies.
- No support for standard input/output (stdio) file handles or functions.
- Not fully recursive, which limits complex command handling.

The C shell improved readability and performance compared to the Bourne shell. The interactive features and innovations in csh influenced all subsequent Unix shells.

3. TENEX C Shell (tcsh)

The **TENEX C shell** (**tcsh**) is an extension of the C shell (**csh**) merged in the early 1980s. The shell is backward compatible with csh, with additional features and concepts borrowed from the TENEX OS. The TENEX C shell executable path is in */bin/tcsh*. The user prompt is **hostname: directory>** while the root prompt is **hostname:directory#**.

Early versions of Mac OS and the default root shell of FreeBSD use tcsh.

```
phoenixNAP:~> ps -p $$ -ocomm=
tcsh
phoenixNAP:~> set foo=bar
phoenixNAP:~> echo $foo
bar
phoenixNAP:~> sudo -i
[sudo] password for kb:
phoenixNAP:~# whoami
root
phoenixNAP:~#
```

Additional features of the shell include:

- Advanced command history.
- Programmable autocomplete.
- Wildcard matching.
- Job control.
- Built-in where command.

Since tcsh is an extension of the C shell, many drawbacks persist in the extended version.

4. KornShell (ksh)

The **KornShell** (**ksh**) is a Unix shell and language based on the Bourne shell (sh) developed in the early 1980s. The location is in */bin/ksh* or */bin/ksh93*, while the prompt is the same as the Bourne shell (**$** for a user and **#** for root).

```
$ ps -p $$ -ocomm=
ksh93
$ foo=bar
$ echo $foo
bar
$ sudo -i
[sudo] password for kb:
# whoami
root
#
```

The shell implements feature from the C shell and Bourne shell, aiming to focus on both interactive commands and programming features. The KornShell adds new features of its own, such as:

- Built-in mathematical functions and floating-point arithmetic.
- Object-oriented programming.
- Extensibility of built-in commands.
- Compatible with the Bourne shell.

The shell is faster than both the C shell and the Bourne shell.

5. Debian Almquist Shell (dash)

The **Debian Almquist Shell** (**dash**) is a Unix shell developed in the late 1990s from the Almquist shell (ash), which was ported to Debian and renamed. Dash is famous for being the default shell for Ubuntu and Debian.

The shell is minimal and POSIX compliant, making it convenient for OS startup scripts.

The executable path is */bin/dash*, in addition to */bin/sh* pointing to */bin/dash* on Ubuntu and Debian. The default and root user prompt is the same as in the Bourne shell.

```
$ ps -p $$ -ocomm=
dash
$ foo=bar
$ echo $foo
bar
$ sudo -i
[sudo] password for kb:
# whoami
root
#
```

Dash features include:

- Execution speeds up to 4x faster than bash and other shells.
- Requires minimal disk space, CPU, and RAM compared to alternatives.

The main drawback is that dash is not bash-compatible. The features not included in dash are known as "bashisms." Therefore, bash scripts require additional reworkings of bashisms to run successfully.

6. Bourne Again Shell (bash)

The **Bourne Again shell** is a Unix shell and command language created as an extension of the Bourne shell (**sh**) in 1989. The shell program is the default login shell for many Linux distributions and earlier versions of macOS.

The shell name shortens to **bash**, and the location is */bin/bash*. Like the Bourne shell, the bash prompt is **$** for a regular user and **#** for root.

```
kb@phoenixNAP:~$ ps -p $$ -ocomm=
bash
kb@phoenixNAP:~$ foo=bar
kb@phoenixNAP:~$ echo $foo
bar
kb@phoenixNAP:~$ sudo -i
[sudo] password for kb:
root@phoenixNAP:~# whoami
root
root@phoenixNAP:~#
```

Bash introduces features not found in the Bourne shell, some of which include:

- Brace expansion.
- Command completion.
- Basic debugging and signal handling.
- Command history.
- Conditional commands, such as the bash if and bash case statements.
- Heredoc support.

7. Z Shell (zsh)

The **Z shell** (**zsh**) is a Unix shell created as an extension for the Bourne shell in the early 1990s. The feature-rich shell borrows ideas from ksh and tcsh to create a well-built and usable alternative. The executable location is in */bin/zsh*. The prompt is **user@hostname location %** for regular users and **hostname#** for the root user. The Z shell is the default shell of Kali Linux and Mac OS.

```
kb@phoenixNAP ~ % ps -p $$ -ocomm=
zsh
kb@phoenixNAP ~ % foo=bar
kb@phoenixNAP ~ % echo $foo
bar
kb@phoenixNAP ~ % sudo -i
[sudo] password for kb:
phoenixNAP# whoami
root
phoenixNAP# ps -p $$
```

Some new features added to the zsh include:

- Shared history among all running shell sessions.
- Improved array and variable handling.
- Spelling corrections and command name autofill.
- Various compatibility modes.
- Extensibility through plugins.

The shell is highly configurable and customizable due to the community-driven support through the Oh My Zsh framework.

8. Friendly Interactive Shell (fish)

The **Friendly Interactive Shell** (**fish**) is a Unix shell released in the mid-2000s with a focus on usability. The feature-rich shell does not require additional configuration, which makes it user-friendly from the start. The default executable path is */usr/bin/fish*. The user prompt is **user@hostname location>**, while the root prompt is **root@hostname location#**.

```
Welcome to fish, the friendly interactive shell
Type help for instructions on how to use fish
kb@phoenixNAP ~> ps -p $fish_pid -ocomm=
fish
kb@phoenixNAP ~> set foo bar
kb@phoenixNAP ~> echo $foo
bar
kb@phoenixNAP ~> sudo -i
[sudo] password for kb:
Welcome to fish, the friendly interactive shell
Type help for instructions on how to use fish
root@phoenixNAP ~# whoami
root
root@phoenixNAP ~#
```

Features in the shell include:

- Advanced suggestions/tab completion based on the current directory history.
- Helpful syntax highlighting and descriptive error messages.
- Web-based configuration.
- Command history with search options.

The main drawback of fish is non-POSIX compliance. However, the developers aim to improve flawed designs from POSIX.