Chapter 03
Configuring the Shell

Introduction

- One key component of the Bash shell is shell *variables*.
- Variables store vital system information and modify the behavior of the Bash shell, as well as many commands.
- The `PATH` variable affects how commands are executed and how other variables affect your ability to use the history of your commands.
- Initialization files make shell variables *persistent*, so they will be created each time you log into the system.

Shell Variables

- A *variable* is a name or identifier that can be assigned a value.
- To assign a value to a variable, type the name of the variable immediately followed by the equal sign = character and then the value.

```
name="value"
```

- Variable names should start with a letter (alpha character) or underscore and contain only letters, numbers and the underscore character. For example:
  - `a=1`
  - `_1=a`
  - `LONG_VARIABLE='OK'`
  - `Name='Jose Romero'`

Local Environment Variables

- A *local variable* is only available to the shell in which it was created.
- An *environment variable* is available to the shell in which it was created, and all other commands/programs started by the shell.
- To set the value of a variable, use the following assignment expression.

```
variable=value
```

```
sysadmin@localhost:~$ name='julie'
sysadmin@localhost:~$ echo $name
julie
```

- To create an environment variable, use the `export` command.

```
sysadmin@localhost:~$ export JOB=engineer
```

Displaying Variables

- There are several ways to display the values of variables.
- The `set` command will display all variables (local and environment).
- To display only environment variables, you can use several commands that provide nearly the same output:
  - `env`
  - `declare -x`
  - `typeset -x`
  - `export -p`
- To display the value of a specific variable, use the `echo` command with the name of the variable prefixed by the `$` (dollar sign). For example:

```
sysadmin@localhost:~$ echo $PATH
```

Unsetting Variables

- If you create a variable and then no longer want that variable to be defined, use the `unset` command to delete it:

```
unset VARIABLE
```

Warning

Do not unset critical system variables like the `PATH` variable, as this may lead to a malfunctioning environment.

PATH Variable

- The `PATH` variable contains a list of directories that are used to search for commands entered by the user.
- The `PATH` directories are searched for an executable file that matches the command name.
- The following example displays a typical `PATH` variable:

```
sysadmin@localhost:~$ echo $PATH
/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/us
r/games:/usr/local/games
```

PATH Variable

- To execute commands that are not contained in the directories that are listed in the `PATH` variable, several options exist:
  - Type the *absolute path* to the command.
  - Use the *relative path* to the command.
  - The PATH variable can be set to include the directory where the command is located.
  - Copy command to a directory that is listed in the `PATH` variable.
- An *absolute path* specifies the location of a file or directory from the top-level directory (i.e. `/usr/bin/ls`).
- A *relative path* specifies the location of a file or directory relative to the current directory (i.e. `test/newfile`).

Initialization Files

- Initialization files set the value of variables, create aliases and functions, and execute other commands that are useful in starting the shell.
- There are two types of initialization files:
  - Global initialization files - affect all users on the system.
  - Local initialization files - specific to an individual user.
- BASH initialization files include:
  - `/etc/profile`
  - `~/.bash_profile, ~/.bash_login, ~/.profile`
  - `~/.bashrc`
  - `/etc/bashrc`

Modifying Initialization Files

- The way a user's shell operates can be changed by modifying that user's initialization files.

- In some distributions, the `default ~/.bash_profile` file contains lines that customize the `PATH` environment variable:

```
PATH=$PATH:$HOME/bin
export PATH
```

  - The first line sets the `PATH` variable to the existing value with the addition of the `bin` subdirectory of the user's home directory.

  - The second line converts the local `PATH` variable into an environment variable.

BASH Exit Scripts

- The Bash shell may execute one or more files upon exiting.
- These files are used for "cleaning up" as the user exits the shell.
- The following exit files may exist:
  - `~/.bash_logout`
  - `/etc/bash_logout`

Command History

- The `~/.bash_history` file contains a history of the commands that a user has executed within the Bash shell.

- There are several ways that this command history is advantageous to the user:
    - The **Up ↑** and **Down ↓ Arrow Keys** can be used to review your history and select a previous command to execute again.
    - Select a previous command and modify it before executing it.
    - Press **Ctrl+R** and then begin typing a portion of a previous command to do a reverse search through history.
    - Execute a command again, based upon a number that is associated with the command.

Using the history Command

- The `history` command can be used to re-execute previously executed commands.

```
sysadmin@localhost:~$ history
1 ls
2 cd test
3 cat alpha.txt
4 ls -l
5 cd ..
```

- The most common options for the `history` command are:
  - `-c` = Clear list
  - `-r` = Read the history file and replace the current history
  - `-w` = Write the current history list to the history file

Configuring the history Command

- When the shell is closed, commands in the history list and stores them in `~/.bash_history`, also called the *history file*.
- The `HISTFILESIZE` variable will determine how many commands to write to this file.
- To store the history commands in a different file, edit the value of the `HISTFILE` variable.
- The `HISTCONTROL` variable can be set to different features such as ignoring spaces or duplicate commands.
- The `HISTIGNORE` variable can also be used to ignore commonly used commands.

Execute Previous Commands

- The `!` exclamation mark is a special character that indicates the execution of a command within the history list.

- The following are some examples of using the exclamation `!` character:

  - `!!` - Repeat the last command

  - `!-4` - Execute command that was run four commands ago

  - `!55` - Execute command number 55

  - `!to` - Execute the last command that starts with `to`

  - `!?bob` - Execute the last command that contained `bob`