[**http://www.tldp.org/LDP/Bash-Beginners-Guide/html/**](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/)

[Introduction](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/f32.html)

1. [Bash and Bash scripts](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_01.html)

1.1. [Common shell programs](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_01.html)

1.2. [Advantages of the Bourne Again SHell](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_02.html)

1.3. [Executing commands](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_03.html)

1.4. [Building blocks](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_04.html)

1.5. [Developing good scripts](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_05.html)

1.6. [Summary](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_06.html)

1.7. [Exercises](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_07.html)

2. [Writing and debugging scripts](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_02.html)

2.1. [Creating and running a script](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_02_01.html)

2.2. [Script basics](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_02_02.html)

2.3. [Debugging Bash scripts](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_02_03.html)

2.4. [Summary](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_02_05.html)

2.5. [Exercises](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_02_06.html)

3. [The Bash environment](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_03.html)

3.1. [Shell initialization files](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_01.html)

3.2. [Variables](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_02.html)

3.3. [Quoting characters](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_03.html)

3.4. [Shell expansion](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_04.html)

3.5. [Aliases](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_05.html)

3.6. [More Bash options](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_06.html)

3.7. [Summary](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_07.html)

3.8. [Exercises](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_08.html)

4. [Regular expressions](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_04.html)

4.1. [Regular expressions](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_04_01.html)

4.2. [Examples using grep](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_04_02.html)

4.3. [Pattern matching using Bash features](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_04_03.html)

4.4. [Summary](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_04_04.html)

4.5. [Exercises](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_04_05.html)

5. [The GNU sed stream editor](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_05.html)

5.1. [Introduction](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_05_01.html)

5.2. [Interactive editing](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_05_02.html)

5.3. [Non-interactive editing](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_05_03.html)

5.4. [Summary](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_05_04.html)

5.5. [Exercises](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_05_05.html)

6. [The GNU awk programming language](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_06.html)

6.1. [Getting started with gawk](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_06_01.html)

6.2. [The print program](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_06_02.html)

6.3. [Gawk variables](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_06_03.html)

6.4. [Summary](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_06_04.html)

6.5. [Exercises](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_06_05.html)

7. [Conditional statements](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_07.html)

7.1. [Introduction to if](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html)

7.2. [More advanced if usage](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_02.html)

7.3. [Using case statements](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_03.html)

7.4. [Summary](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_04.html)

7.5. [Exercises](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_05.html)

8. [Writing interactive scripts](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_08.html)

8.1. [Displaying user messages](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_08_01.html)

8.2. [Catching user input](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_08_02.html)

8.3. [Summary](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_08_03.html)

8.4. [Exercises](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_08_04.html)

9. [Repetitive tasks](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_09.html)

9.1. [The for loop](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_01.html)

9.2. [The while loop](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_02.html)

9.3. [The until loop](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_03.html)

9.4. [I/O redirection and loops](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_04.html)

9.5. [Break and continue](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_05.html)

9.6. [Making menus with the select built-in](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_06.html)

9.7. [The shift built-in](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_07.html)

9.8. [Summary](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_08.html)

9.9. [Exercises](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_09.html)

10. [More on variables](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_10.html)

10.1. [Types of variables](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_10_01.html)

10.2. [Array variables](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_10_02.html)

10.3. [Operations on variables](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_10_03.html)

10.4. [Summary](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_10_04.html)

10.5. [Exercises](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_10_05.html)

11. [Functions](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_11.html)

11.1. [Introduction](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_11_01.html)

11.2. [Examples of functions in scripts](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_11_02.html)

11.3. [Summary](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_11_03.html)

11.4. [Exercises](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_11_04.html)

12. [Catching signals](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_12.html)

12.1. [Signals](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_12_01.html)

12.2. [Traps](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_12_02.html)

12.3. [Summary](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_12_03.html)

12.4. [Exercises](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_12_04.html)

A. [Shell Features](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/app3.html)

A.1. [Common features](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/x7243.html)

A.2. [Differing features](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/x7369.html)

[Glossary](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/gloss.html)

[Index](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/glossary.html)

**List of Tables**

1. [Typographic and usage conventions](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/intro_09.html#conventions)

1-1. [Overview of programming terms](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_05.html#table_01_01)

2-1. [Overview of set debugging options](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_02_03.html#table_02_01)

3-1. [Reserved Bourne shell variables](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_02.html#table_03_01)

3-2. [Reserved Bash variables](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_02.html#table_03_02)

3-3. [Special bash variables](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_02.html#table_03_03)

3-4. [Arithmetic operators](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_04.html#table_03_04)

4-1. [Regular expression operators](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_04_01.html#table_04_01)

5-1. [Sed editing commands](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_05_01.html#tab_05_01)

5-2. [Sed options](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_05_01.html#tab_05_02)

6-1. [Formatting characters for gawk](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_06_02.html#tab_06_01)

7-1. [Primary expressions](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html#tab_07_01)

7-2. [Combining expressions](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html#tab_07_02)

8-1. [Escape sequences used by the echo command](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_08_01.html#tab_08_01)

8-2. [Options to the read built-in](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_08_02.html#tab_08_02)

10-1. [Options to the declare built-in](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_10_01.html#tab_10_01)

12-1. [Control signals in Bash](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_12_01.html#tab_12_01)

12-2. [Common kill signals](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_12_01.html#tab_12_02)

A-1. [Common Shell Features](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/x7243.html#AEN7246)

A-2. [Differing Shell Features](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/x7369.html#AEN7387)

**List of Figures**

1. [Bash Guide for Beginners front cover](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/intro_03.html#AEN71)

2-1. [script1.sh](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_02_01.html#AEN1409)

3-1. [Different prompts for different users](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_01.html#AEN1878)

6-1. [Fields in awk](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_06_02.html#AEN4111)

7-1. [Testing of a command line argument with if](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_02.html#AEN5029)

7-2. [Example using Boolean operators](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_02.html#AEN5144)

**Chapter 1. Bash and Bash scripts**

**Table of Contents**

1.1. [Common shell programs](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_01.html)

1.1.1. [General shell functions](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_01.html#sect_01_01_01)

1.1.2. [Shell types](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_01.html#sect_01_01_02)

1.2. [Advantages of the Bourne Again SHell](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_02.html)

1.2.1. [Bash is the GNU shell](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_02.html#sect_01_02_01)

1.2.2. [Features only found in bash](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_02.html#sect_01_02_02)

1.3. [Executing commands](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_03.html)

1.3.1. [General](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_03.html#sect_01_03_01)

1.3.2. [Shell built-in commands](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_03.html#sect_01_03_02)

1.3.3. [Executing programs from a script](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_03.html#sect_01_03_03)

1.4. [Building blocks](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_04.html)

1.5. [Developing good scripts](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_05.html)

1.5.1. [Properties of good scripts](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_05.html#sect_01_05_01)

1.5.2. [Structure](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_05.html#sect_01_05_02)

1.5.3. [Terminology](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_05.html#sect_01_05_03)

1.5.4. [A word on order and logic](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_05.html#sect_01_05_04)

1.5.5. [An example Bash script: mysystem.sh](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_05.html#sect_01_05_05)

1.5.6. [Example init script](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_05.html#sect_01_05_06)

1.6. [Summary](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_06.html)

1.7. [Exercises](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_07.html)

In this introduction module we

* Describe some common shells
* Point out GNU Bash advantages and features
* Describe the shell's building blocks
* Discuss Bash initialization files
* See how the shell executes commands
* Look into some simple script examples

# 1.1. Common shell programs

## 1.1.1. General shell functions

|  |
| --- |
| The UNIX shell program interprets user commands, which are either **directly entered by the user, or which can be read from a file called the shell script** or **shell program**. Shell scripts are interpreted, not compiled.  The shell reads commands from the script line per line  while a compiler converts a program into machine readable form, an executable file - which may then be used in a shell script.  Apart from passing commands to the kernel, the **main task of a shell is providing a user environment**, which can be configured individually using shell resource configuration files. |

## 1.1.2. Shell types

Just like people know different languages and dialects, your UNIX system will usually offer a variety of shell types:

|  |
| --- |
| * **sh** or Bourne Shell: the original shell still used on UNIX systems and in UNIX-related environments. This is the basic shell, a small program with few features. While this is not the standard shell, it is still available on every Linux system for compatibility with UNIX programs. * **bash** or Bourne Again shell: the standard GNU shell, intuitive and flexible. Probably most advisable for beginning users while being at the same time a powerful tool for the advanced and professional user. On Linux, **bash** is the standard shell for common users. This shell is a so-called superset of the Bourne shell, a set of add-ons and plug-ins. This means that the Bourne Again shell is compatible with the Bourne shell: commands that work in **sh**, also work in **bash**. However, the reverse is not always the case. All examples and exercises in this book use **bash**. * **csh** or C shell: the syntax of this shell resembles that of the C programming language. Sometimes asked for by programmers. * **tcsh** or TENEX C shell: a superset of the common C shell, enhancing user-friendliness and speed. That is why some also call it the Turbo C shell. * **ksh** or the Korn shell: sometimes appreciated by people with a UNIX background. A superset of the Bourne shell; with standard configuration a nightmare for beginning users. |

|  |  |  |  |
| --- | --- | --- | --- |
| The file /etc/shells gives an overview of known shells on a Linux system:   |  | | --- | | mia:~> **cat /etc/shells**  /bin/bash  /bin/sh  /bin/tcsh  /bin/csh |   Your default shell is set in the /etc/passwd file, like this line for user *mia*:   |  | | --- | | mia:L2NOfqdlPrHwE:504:504:Mia Maya:/home/mia:/bin/bash |   To switch from one shell to another, just enter the name of the new shell in the active terminal. The system finds the directory where the name occurs using the PATH settings, and since a shell is an executable file (program), the current shell activates it and it gets executed. A new prompt is usually shown, because each shell has its typical appearance:   |  | | --- | | mia:~> **tcsh**  [mia@post21 ~]$ | |

# 1.2. Advantages of the Bourne Again SHell

## 1.2.1. Bash is the GNU shell

|  |
| --- |
| The GNU project (GNU's Not UNIX) provides tools for UNIX-like system administration which are free software and comply to UNIX standards.  Bash is an sh-compatible shell that incorporates useful features from the Korn shell (ksh) and C shell (csh)  It offers functional improvements over sh for both programming and interactive use; these include command line editing, unlimited size command history, job control, shell functions and aliases, indexed arrays of unlimited size, and integer arithmetic in any base from two to sixty-four. Bash can run most sh scripts without modification. |

## 1.2.2. Features only found in bash

### 1.2.2.1. Invocation

|  |
| --- |
| In addition to the single-character shell command line options which can generally be configured using the **set** shell built-in command, there are several multi-character options that you can use. We will come across a couple of the more popular options in this and the following chapters; the complete list can be found in the Bash info pages, Bash features->Invoking Bash. |

### 1.2.2.2. Bash startup files

|  |
| --- |
| **Startup files** are scripts that are read and executed by Bash when it starts. The following subsections describe different ways to start the shell, and the startup files that are read consequently. |

#### 1.2.2.2.1. Invoked as an interactive login shell, or with `--login'

|  |
| --- |
| Interactive means you can enter commands. The shell is not running because a script has been activated. A login shell means that you got the shell after authenticating to the system, usually by giving your user name and password.  **Files read**:   * /etc/profile * ~/.bash\_profile, ~/.bash\_login or ~/.profile: first existing readable file is read * ~/.bash\_logout upon logout.   **Error messages are printed if configuration files exist but are not readable**. If a file does not exist, bash searches for the next. |

#### 1.2.2.2.2. Invoked as an interactive non-login shell

|  |
| --- |
| A non-login shell means that you did not have to authenticate to the system. For instance, when you open a terminal using an icon, or a menu item, that is a non-login shell.  **Files read**:   * ~/.bashrc (rc mean: run command)   This file is usually referred to in ~/.bash\_profile:  **if [ -f ~/.bashrc ]; then . ~/.bashrc; fi**  See [Chapter 7](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_07.html) for more information on the **if** construct. |

#### 1.2.2.2.3. Invoked non-interactively

|  |
| --- |
| All scripts use non-interactive shells. They are programmed to do certain tasks and cannot be instructed to do other jobs than those for which they are programmed.  **Files read**:   * defined by BASH\_ENV   PATH is not used to search for this file, so if you want to use it, best refer to it by giving the full path and file name. |

#### 1.2.2.2.4. Invoked with the sh command

|  |
| --- |
| Bash tries to behave as the historical Bourne **sh** program while conforming to the POSIX standard as well.  **Files read**:   * /etc/profile * ~/.profile   When invoked interactively, the ENV variable can point to extra startup information. |

#### 1.2.2.2.5. POSIX mode

|  |
| --- |
| This option is enabled either using the **set** built-in:  **set -o posix**  or by calling the **bash** program with the --posix option. Bash will then try to behave as compliant as possible to the POSIX standard for shells. Setting the POSIXLY\_CORRECT variable does the same.  Files read:  defined by ENV variable. |

#### 1.2.2.2.6. Invoked remotely

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Files read when invoked by **rshd**: remote connect   * ~/.bashrc  |  |  | | --- | --- | | Warning | **Avoid use of r-tools** | |  | Be aware of the dangers when using tools such as **rlogin**, **telnet**, **rsh** and **rcp**. They are intrinsically insecure because confidential data is sent over the network unencrypted. If you need tools for remote execution, file transfer and so on, use an implementation of Secure SHell, generally known as SSH | |

#### 1.2.2.2.7. Invoked when UID is not equal to EUID

No startup files are read in this case.

### 1.2.2.3. Interactive shells

#### 1.2.2.3.1. What is an interactive shell?

|  |
| --- |
| An interactive shell generally reads from, and writes to, a user's terminal: input and output are connected to a terminal. Bash interactive behavior is started when the **bash** command is called upon without non-option arguments, except when the option is a string to read from or when the shell is invoked to read from standard input, which allows for positional parameters to be set (see [Chapter 3](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_03.html) ). |

#### 1.2.2.3.2. Is this shell interactive?

|  |  |
| --- | --- |
| Test by looking at the content of the special parameter -, **it contains an 'i' when the shell is interactive**:   |  | | --- | | eddy:~> **echo $-**  himBH |   In non-interactive shells, the prompt, PS1, is unset. |

#### 1.2.2.3.3. Interactive shell behavior

|  |
| --- |
| Differences in interactive mode:   * Bash reads startup files. * Job control enabled by default. * Prompts are set, PS2 is enabled for multi-line commands, it is usually set to ">". This is also the prompt you get when the shell thinks you entered an unfinished command, for instance when you forget quotes, command structures that cannot be left out, etc. * Commands are by default read from the command line using **readline**. * Bash interprets the shell option ignoreeof instead of exiting immediately upon receiving EOF (End Of File). * Command history and history expansion are enabled by default. History is saved in the file pointed to by HISTFILE when the shell exits. By default, HISTFILE points to ~/.bash\_history. * Alias expansion is enabled. * In the absence of traps, the SIGTERM signal is ignored. * In the absence of traps, SIGINT is caught and handled. Thus, typing **Ctrl**+**C**, for example, will not quit your interactive shell. * Sending SIGHUP signals to all jobs on exit is configured with the huponexit option. * Commands are executed upon read. * Bash checks for mail periodically. * Bash can be configured to exit when it encounters unreferenced variables. In interactive mode this behavior is disabled. * When shell built-in commands encounter redirection errors, this will not cause the shell to exit. * Special built-ins returning errors when used in POSIX mode don't cause the shell to exit. The built-in commands are listed in [Section 1.3.2](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_01_03.html#sect_01_03_02). * Failure of **exec** will not exit the shell. * Parser syntax errors don't cause the shell to exit. * Simple spell check for the arguments to the **cd** built-in is enabled by default. * Automatic exit after the length of time specified in the TMOUT variable has passed, is enabled.   More information:   * [Section 3.2](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_02.html) * [Section 3.6](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_06.html) * See [Chapter 12](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_12.html) for more about signals. * [Section 3.4](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_04.html) discusses the various expansions performed upon entering a command. |

### 1.2.2.4. Conditionals

|  |
| --- |
| **Conditional expressions** are used by the **[[** compound command and by the **test** and **[** built-in commands.  Expressions may be unary or binary.  **Unary expressions are often used to examine the status of a file**. You only need one object, for instance a file, to do the operation on.  There are string operators and numeric comparison operators as well; these are binary operators, requiring two objects to do the operation on. If the FILE argument to one of the primaries is in the form /dev/fd/N, then file descriptor N is checked. If the FILE argument to one of the primaries is one of /dev/stdin, /dev/stdout or /dev/stderr, then file descriptor 0, 1 or 2 respectively is checked.  Conditionals are discussed in detail in [Chapter 7](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_07.html).  More information about the file descriptors in [Section 8.2.3](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_08_02.html#sect_08_02_03). |

### 1.2.2.5. Shell arithmetic

|  |
| --- |
| The shell allows arithmetic expressions to be evaluated, as one of the shell expansions or by the **let** built-in.  Evaluation is done in fixed-width integers with no check for overflow, though division by 0 is trapped and flagged as an error. The operators and their precedence and associativity are the same as in the C language, see [Chapter 3](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_03.html). |

### 1.2.2.6. Aliases

|  |
| --- |
| Aliases allow a string to be substituted for a word when it is used as the first word of a simple command. The shell maintains a list of aliases that may be set and unset with the **alias** and **unalias** commands.  Bash always reads at least one complete line of input before executing any of the commands on that line. Aliases are expanded when a command is read, not when it is executed. Therefore, an alias definition appearing on the same line as another command does not take effect until the next line of input is read. The commands following the alias definition on that line are not affected by the new alias.  Aliases are expanded when a function definition is read, not when the function is executed, because a function definition is itself a compound command. As a consequence, aliases defined in a function are not available until after that function is executed.  We will discuss aliases in detail in [Section 3.5](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_05.html). |

### 1.2.2.7. Arrays

|  |
| --- |
| Bash provides one-dimensional array variables. Any variable may be used as an array; the **declare** built-in will explicitly declare an array. There is no maximum limit on the size of an array, nor any requirement that members be indexed or assigned contiguously. Arrays are zero-based. See [Chapter 10](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_10.html). |

### 1.2.2.8. Directory stack

The directory stack is a list of recently-visited directories. The **pushd** built-in adds directories to the stack as it changes the current directory, and the **popd** built-in removes specified directories from the stack and changes the current directory to the directory removed.

Content can be displayed issuing the **dirs** command or by checking the content of the DIRSTACK variable.

More information about the workings of this mechanism can be found in the Bash info pages.

### 1.2.2.9. The prompt

Bash makes playing with the prompt even more fun. See the section Controlling the Prompt in the Bash info pages.

### 1.2.2.10. The restricted shell

|  |
| --- |
| When invoked as **rbash** or with the --restricted or -r option, the following happens:   * The **cd** built-in is disabled. * Setting or unsetting SHELL, PATH, ENV or BASH\_ENV is not possible. * Command names can no longer contain slashes. * Filenames containing a slash are not allowed with the **.** (**source**) built-in command. * The **hash** built-in does not accept slashes with the -p option. * Import of functions at startup is disabled. * SHELLOPTS is ignored at startup. * Output redirection using **>**, **>|**, **><**, **>&**, **&>** and **>>** is disabled. * The **exec** built-in is disabled. * The -f and -d options are disabled for the **enable** built-in. * A default PATH cannot be specified with the **command** built-in. * Turning off restricted mode is not possible.   When a command that is found to be a shell script is executed, **rbash** turns off any restrictions in the shell spawned to execute the script.  More information:   * [Section 3.2](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_02.html) * [Section 3.6](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_06.html) * Info Bash->Basic Shell Features->Redirections * [Section 8.2.3](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_08_02.html#sect_08_02_03): advanced redirection |

# 1.3. Executing commands

## 1.3.1. General

|  |
| --- |
| Bash determines the type of program that is to be executed. Normal programs are system commands that exist in compiled form on your system. When such a program is executed, a new process is created because Bash makes an exact copy of itself. **This child process has the same environment as its parent, only the process ID number is different**. This procedure is called *forking(chia nhiều nhánh*.)  After the forking process, the address space of the child process is overwritten with the new process data. This is done through an *exec* call to the system.  **The *fork-and-exec*** **mechanism** thus switches an old command with a new, while the environment in which the new program is executed remains the same, including configuration of input and output devices, environment variables and priority. **This mechanism is used to create all UNIX processes, so it also applies to the Linux operating system**.  Even the first process, **init**, with process ID 1, is forked during the boot procedure in the so-called *bootstrapping* procedure. |

## 1.3.2. Shell built-in commands

|  |
| --- |
| **Built-in commands are contained within the shell itself**. When the name of a built-in command is used as the first word of a simple command, **the shell executes the command directly, without creating a new process**. Built-in commands are necessary to implement functionality impossible or inconvenient to obtain with separate utilities.  Bash supports 3 types of built-in commands:   * Bourne Shell built-ins:   **:**, **.**, **break**, **cd**, **continue**, **eval**, **exec**, **exit**, **export**, **getopts**, **hash**, **pwd**, **readonly**, **return**, **set**, **shift**, **test**, **[**, **times**, **trap**, **umask** and **unset**.   * Bash built-in commands:   **alias**, **bind**, **builtin**, **command**, **declare**, **echo**, **enable**, **help**, **let**, **local**, **logout**, **printf**, **read**, **shopt**, **type**, **typeset**, **ulimit** and **unalias**.   * Special built-in commands:   When Bash is executing in POSIX mode, the special built-ins differ from other built-in commands in three respects:   * 1. Special built-ins are found before shell functions during command lookup.   2. If a special built-in returns an error status, a non-interactive shell exits.   3. Assignment statements preceding the command stay in effect in the shell environment after the command completes.   The POSIX special built-ins are **:**, **.**, **break**, **continue**, **eval**, **exec**, **exit**, **export**, **readonly**, **return**, **set**, **shift**, **trap** and **unset**.  Most of these built-ins will be discussed in the next chapters. For those commands for which this is not the case, we refer to the Info pages. |

## 1.3.3. Executing programs from a script

|  |
| --- |
| **When the program being executed is a shell script, bash will create a new bash process using a fork.** This subshell reads the lines from the shell script one line at a time. Commands on each line are read, interpreted and executed as if they would have come directly from the keyboard.  *While the subshell processes each line of the script, the parent shell waits for its child process to finish. When there are no more lines in the shell script to read, the subshell terminates. The parent shell awakes and displays a new prompt*. |

# 1.4. Building blocks

## 1.4.1. Shell building blocks

### 1.4.1.1. Shell syntax

|  |
| --- |
| If input is not commented, the shell reads it and divides it into words and operators, employing quoting rules to define the meaning of each character of input. Then these words and operators are translated into commands and other constructs, which return an exit status available for inspection or processing. The above fork-and-exec scheme is only applied after the shell has analyzed input in the following way:   * **The shell reads its input from a file, from a string or from the user's terminal**. * Input is broken up into words and operators, obeying the quoting rules, see [Chapter 3](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_03.html). These tokens are separated by metacharacters. Alias expansion is performed. * The shell parses (analyzes and substitutes) the tokens into simple and compound commands. * Bash performs various shell expansions, breaking the expanded tokens into lists of filenames and commands and arguments. * Redirection is performed if necessary, redirection operators and their operands are removed from the argument list. * Commands are executed. * Optionally the shell waits for the command to complete and collects its exit status. |

### 1.4.1.2. Shell commands

|  |
| --- |
| A simple shell command such as **touch file1 file2 file3** consists of the command itself followed by arguments, separated by spaces.  More complex shell commands are composed of simple commands arranged together in a variety of ways: in **a pipeline in which the output of one command becomes the input of a second, in a loop or conditional construct, or in some other grouping.** A couple of examples:  **ls | more**  **gunzip file.tar.gz | tar xvf *–*** |

### 1.4.1.3. Shell functions

|  |
| --- |
| **Shell functions are a way to group commands for later execution using a single name for the group**. They are executed just like a "regular" command. When the name of a shell function is used as a simple command name, the list of commands associated with that function name is executed.  Shell functions are executed in the current shell context; no new process is created to interpret them.  Functions are explained in [Chapter 11](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_11.html). |

### 1.4.1.4. Shell parameters

|  |
| --- |
| **A parameter is an entity that stores values**. It can be a name, a number or a special value. For the shell's purpose, **a variable is a parameter that stores a name**. A variable has a value and zero or more attributes. Variables are created with the **declare** shell built-in command.  If no value is given, a variable is assigned the null string. Variables can only be removed with the **unset** built-in.  Assigning variables is discussed in [Section 3.2](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_02.html), advanced use of variables in [Chapter 10](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/chap_10.html). |

### 1.4.1.5. Shell expansions

|  |
| --- |
| Shell expansion is performed after each command line has been split into tokens. These are the expansions performed:   * Brace expansion * Tilde expansion * Parameter and variable expansion * Command substitution * Arithmetic expansion * Word splitting * Filename expansion   We'll discuss these expansion types in detail in [Section 3.4](http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_03_04.html). |

### 1.4.1.6. Redirections

|  |
| --- |
| Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. Redirection may also be used to open and close files for the current shell execution environment. |