The Linux Documentation Project (TLDP)
www.tldp.org

Become a Bash power user!

This book will teach you the tools and techniques that make the Bourne Again Shell the tool of choice of professional system administrators worldwide!

Learn about:

- ✔ Shell building blocks and common practices
- ✔ The `grep`, `awk` and `sed` tools
- ✔ Loops, conditional tests, functions and traps
- ✔ Building interactive scripts
- ✔ And more!

All chapters come packed with helpful examples and exercises.

About the Author

Linux advocate of the first hour, ***Machtelt Garrels*** *has made many contributions to the Open Source community and has been working for over fifteen years on the wider acceptance of Linux and other Open Source products. At the Linux Documentation Project she initially found a fertile ground for making her work read throughout the world. She writes whenever she has the time, closing gaps in existing documentation and taking the opportunity to simplify it when necessary, always keeping in mind that practice is the only way to learn.*

ISBN 978-1-59682-201-6

Published by Fultus Corporation

Fultus™

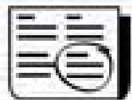
www.fultus.com

M.Garrels • Bash Guide for Beginners (2nd Edition)

Machtelt Garrels

bash guide for beginners

Second Edition



Your Advertising Here

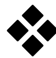
Linbrary Advertising Club

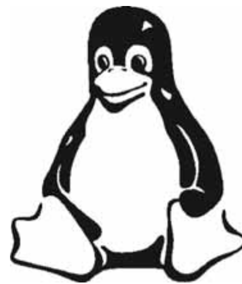
Linbrary™ - Linux Documentation Library
www.linbrary.com

Machtelt Garrels

Bash Guide for Beginners

Second Edition





Fultus ™ *Books*

Bash Guide for Beginners

Second Edition

by

Machtelt Garrels

Cover design by Fultus Corporation

ISBN-10: 1-59682-201-5
ISBN-13: 978-1-59682-201-6

All rights reserved.

Copyright © 2002-2010 by Machtelt Garrels

Published by Fultus Corporation

Corporate Web Site: *www.fultus.com*
Fultus eLibrary: elibrary.fultus.com
Online Book Superstore: store.fultus.com
email: production@fultus.com

The text of this book is freely distributable under the terms of the GNU Free Documentation License, which can be found at *http://www.gnu.org/copyleft/fdl.html*. Cover art and layout are copyright Fultus Corporation and may not be reproduced without permission; violators will be prosecuted to the fullest extent permissible by law.

The author and publisher have made every effort in the preparation of this book to ensure the accuracy of the information. However, the information contained in this book is offered without warranty, either express or implied. Neither the author nor the publisher nor any dealer or distributor will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Linux is a registered trademark of Linus Torvalds. Penguin logo based on artwork by Larry Ewing.
All product names and services identified throughout this manual are trademarks or registered trademarks of their respective companies.

Table of Contents

Introduction.......... 12

Chapter 1. Bash and Bash scripts 17

1.1. Common shell programs.......... 17

1.1.1. General shell functions 17

1.1.2. Shell types 17

1.2. Advantages of the Bourne Again SHell 18

1.2.1. Bash is the GNU shell 18

1.2.2. Features only found in bash 19

1.3. Executing commands 25

1.3.1. General.......... 25

1.3.2. Shell built-in commands 25

1.3.3. Executing programs from a script 26

1.4. Building blocks 27

1.4.1. Shell building blocks.......... 27

1.5. Developing good scripts 29

1.5.1. Properties of good scripts 29

1.5.2. Structure 30

1.5.3. Terminology.......... 30

1.5.4. A word on order and logic.......... 31

1.5.5. An example Bash script: mysystem.sh.......... 31

1.5.6. Example init script 33

1.6. 6. Summary 34

1.7. Exercises 34

Chapter 2. Writing and debugging scripts 36

2.1. Creating and running a script 36

2.1.1. Writing and naming.......... 36

2.1.2. script1.sh.......... 37

2.1.3. Executing the script 38
2.2. Script basics 40
2.2.1. Which shell will run the script? 40
2.2.2. Adding comments 40
2.3. Debugging Bash scripts 41
2.3.1. Debugging on the entire script 41
2.3.2. Debugging on part(s) of the script....... 42
2.4. Summary 44
2.5. Exercises 44

Chapter 3. The Bash environment....... 46

3.1. Shell initialization files 46
3.1.1. System-wide configuration files 46
3.1.2. Individual user configuration files 48
3.1.3. Changing shell configuration files....... 51
3.2. Variables....... 52
3.2.1. Types of variables 52
3.2.2. Creating variables 55
3.2.3. Exporting variables....... 56
3.2.4. Reserved variables 57
3.2.5. Special parameters 61
3.2.6. Script recycling with variables 63
3.3. Quoting characters....... 65
3.3.1. Why? 65
3.3.2. Escape characters 65
3.3.3. Single quotes....... 65
3.3.4. Double quotes....... 65
3.3.5. ANSI-C quoting 66
3.3.6. Locales 66
3.4. Shell expansion....... 66
3.4.1. General 66
3.4.2. Brace expansion....... 67
3.4.3. Tilde expansion 67
3.4.4. Shell parameter and variable expansion 68
3.4.5. Command substitution 69
3.4.6. Arithmetic expansion 70

3.4.7. Process substitution 71
3.4.8. Word splitting....72
3.4.9. File name expansion 72
3.5. Aliases....73
3.5.1. What are aliases? 73
3.5.2. Creating and removing aliases....74
3.6. More Bash options 75
3.6.1. Displaying options....75
3.6.2. Changing options....76
3.7. Summary 77
3.8. Exercises 77
Chapter 4. Regular expressions.... 78
4.1. Regular expressions....78
4.1.1. What are regular expressions? 78
4.1.2. Regular expression metacharacters 78
4.1.3. Basic versus extended regular expressions 79
4.2. Examples using grep 79
4.2.1. What is grep? 79
4.2.2. Grep and regular expressions....81
4.3. Pattern matching using Bash features....83
4.3.1. Character ranges....83
4.3.2. Character classes 84
4.4. Summary 84
4.5. Exercises 84
Chapter 5. The GNU sed stream editor 86
5.1. Introduction 86
5.1.1. What is sed? 86
5.1.2. sed commands 87
5.2. Interactive editing....87
5.2.1. Printing lines containing a pattern 87
5.2.2. Deleting lines of input containing a pattern....88
5.2.3. Ranges of lines 89
5.2.4. Find and replace with sed....89
5.3. Non-interactive editing 91

5.3.1. Reading sed commands from a file 91
5.3.2. Writing output files 91
5.4. Summary 93
5.5. Exercises 93
Chapter 6. The GNU awk programming language 94
6.1. Getting started with gawk 94
6.1.1. What is gawk? 94
6.1.2. Gawk commands 95
6.2. The print program 95
6.2.1. Printing selected fields 95
6.2.2. Formatting fields 97
6.2.3. The print command and regular expressions 98
6.2.4. Special patterns 98
6.2.5. Gawk scripts 99
6.3. Gawk variables 100
6.3.1. The input field separator 100
6.3.2. The output separators 101
6.3.3. The number of records 102
6.3.4. User defined variables 102
6.3.5. More examples 103
6.3.6. The printf program 103
6.4. Summary 103
6.5. Exercises 104
Chapter 7. Conditional statements 106
7.1. Introduction to if 106
7.1.1. General 106
7.1.2. Simple applications of if 110
7.2. More advanced if usage 112
7.2.1. if/then/else constructs 112
7.2.2. if/then/elif/else constructs 115
7.2.3. Nested if statements 116
7.2.4. Boolean operations 117
7.2.5. Using the exit statement and if 117
7.3. Using case statements 119

7.3.1. Simplified conditions ... 119
7.3.2. Initscript example ... 121
7.4. Summary ... 121
7.5. Exercises ... 122
Chapter 8. Writing interactive scripts ... 124
8.1. Displaying user messages ... 124
8.1.1. Interactive or not? ... 124
8.1.2. Using the echo built-in command ... 125
8.2. Catching user input ... 127
8.2.1. Using the read built-in command ... 127
8.2.2. Prompting for user input ... 129
8.2.3. Redirection and file descriptors ... 130
8.2.4. File input and output ... 133
8.3. Summary ... 138
8.4. Exercises ... 139
Chapter 9. Repetitive tasks ... 141
9.1. The for loop ... 141
9.1.1. How does it work? ... 141
9.1.2. Examples ... 142
9.2. The while loop ... 143
9.2.1. What is it? ... 143
9.2.2. Examples ... 143
9.3. The until loop ... 146
9.3.1. What is it? ... 146
9.3.2. Example ... 146
9.4. I/O redirection and loops ... 147
9.4.1. Input redirection ... 147
9.4.2. Output redirection ... 147
9.5. Break and continue ... 148
9.5.1. The break built-in ... 148
9.5.2. The continue built-in ... 150
9.5.3. Examples ... 150
9.6. Making menus with the select built-in ... 151
9.6.1. General ... 151

9.6.2. Submenus ... 153
9.7. The shift built-in ... 153
9.7.1. What does it do? ... 153
9.7.2. Examples ... 153
9.8. 8. Summary ... 155
9.9. Exercises ... 155
Chapter 10. More on variables ... 157
10.1. Types of variables ... 157
10.1.1. General assignment of values ... 157
10.1.2. Using the declare built-in ... 158
10.1.3. Constants ... 159
10.2. Array variables ... 159
10.2.1. Creating arrays ... 159
10.2.2. Dereferencing the variables in an array ... 160
10.2.3. Deleting array variables ... 161
10.2.4. Examples of arrays ... 161
10.3. Operations on variables ... 163
10.3.1. Arithmetic on variables ... 163
10.3.2. Length of a variable ... 163
10.3.3. Transformations of variables ... 164
10.4. Summary ... 166
10.5. Exercises ... 167
Chapter 11. Functions ... 168
11.1. Introduction ... 168
11.1.1. What are functions? ... 168
11.1.2. Function syntax ... 168
11.1.3. Positional parameters in functions ... 169
11.1.4. Displaying functions ... 170
11.2. Examples of functions in scripts ... 171
11.2.1. Recycling ... 171
11.2.2. Setting the path ... 171
11.2.3. Remote backups ... 172
11.3. Summary ... 173
11.4. Exercises ... 174

Chapter 12. Catching signals 175
12.1. Signals 175
12.1.1. Introduction 175
12.1.2. Usage of signals with kill 176
12.2. Traps 177
12.2.1. General 177
12.2.2. How Bash interprets traps 178
12.2.3. More examples 178
12.3. Summary 179
12.4. Exercises 179
Appendix A. Shell Features 180
A.1. Common features 180
A.2. Differing features 181
Glossary 185
Index 202
Linbrary™ Advertising Club (LAC) 207
Your Advertising Here 213

List of Figures

Figure 2.1. script1.sh 38
Figure 2.2. Overview of set debugging options 43
Figure 3.1. Different prompts for different users 51
Figure 6.1. Fields in awk 96
Figure 7.1. Testing of a command line argument with if 113
Figure 7.2. Example using Boolean operators 117

List of Tables

Table 1.1. Typographic and usage conventions 14
Table 1.1. Overview of programming terms 30
Table 3.1. Reserved Bourne shell variables 57
Table 3.2. Reserved Bash variables 61
Table 3.3. Special bash variables 61
Table 3.4. Arithmetic operators 70
Table 4.1. Regular expression operators 79
Table 5.1. Sed editing commands 87
Table 5.2. Sed options 87
Table 6.1. Formatting characters for gawk 98
Table 7.1. Primary expressions 108
Table 7.2. Combining expressions 108
Table 8.1. Escape sequences used by the echo command 127
Table 8.2. Options to the read built-in 128
Table 10.1. Options to the declare built-in 158
Table 12.1. Control signals in Bash 176
Table 12.2. Common kill signals 177

Introduction

Why this guide?

The primary reason for writing this document is that a lot of readers feel the existing *HOWTO*[1] to be too short and incomplete, while the *Bash Scripting*[2] guide is too much of a reference work. There is nothing in between these two extremes. I also wrote this guide on the general principal that not enough free basic courses are available, though they should be.

This is a practical guide which, while not always being too serious, tries to give real-life instead of theoretical examples. I partly wrote it because I don't get excited with stripped down and over-simplified examples written by people who know what they are talking about, showing some really cool Bash feature so much out of its context that you cannot ever use it in practical circumstances. You can read that sort of stuff after finishing this book, which contains exercises and examples that will help you survive in the real world.

From my experience as UNIX/Linux user, system administrator and trainer, I know that people can have years of daily interaction with their systems, without having the slightest knowledge of task automation. Thus they often think that UNIX is not userfriendly, and even worse, they get the impression that it is slow and old-fashioned. This problem is another one that can be remedied by this guide.

Who should read this book?

Everybody working on a UNIX or UNIX-like system who wants to make life easier on themselves, power users and sysadmins alike, can benefit from reading this book. Readers who already have a grasp of working the system using the command line will learn the ins and outs of shell scripting that ease execution of daily tasks. System administration relies a great deal on shell scripting; common tasks are often automated using simple scripts. This document is full of examples that will encourage you to write your own and that will inspire you to improve on existing scripts.

---

[1] *http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html*
[2] *http://tldp.org/LDP/abs/html/*

Prerequisites/not in this course:

- You should be an experienced UNIX or Linux user, familiar with basic commands, man pages and documentation
- Being able to use a text editor
- Understand system boot and shutdown processes, init and initscripts
- Create users and groups, set passwords
- Permissions, special modes
- Understand naming conventions for devices, partitioning, mounting/unmounting file systems
- Adding/removing software on your system

See *Introduction to Linux*[3] (or your local *TLDP mirror*[4]) if you haven't mastered one or more of these topics. Additional information can be found in your system documentation (man and info pages), or at *the Linux Documentation Project*[5].

Contributions

Thanks to all the friends who helped (or tried to) and to my husband; your encouraging words made this work possible. Thanks to all the people who submitted bug reports, examples and remarks - among many, many others:

- Hans Bol, one of the groupies
- Mike Sim, remarks on style
- Dan Richter, for array examples
- Gerg Ferguson, for ideas on the title
- Mendel Leo Cooper, for making room
- #linux.be, for keeping my feet on the ground
- Frank Wang, for his detailed remarks on all the things I did wrong ;-)

Special thanks to Tabatha Marshall, who volunteered to do a complete review and spell and grammar check. We make a great team: she works when I sleep. And vice versa ;-)

What do you need?

bash, available from *http://www.gnu.org/directory/GNU/*. The Bash shell is available on nearly every Linux system, and can these days be found on a wide variety of UNIX systems.

[3] *http://tldp.org/LDP/intro-linux/html/*
[4] *http://www.tldp.org/mirrors.html*
[5] *http://tldp.org/*

Compiles easily if you need to make your own, tested on a wide variety of UNIX, Linux, MS Windows and other systems.

Conventions used in this document

The following typographic and usage conventions occur in this text:

Text type	*Meaning*
"Quoted text"	Quotes from people, quoted computer output.
terminal view	Literal computer input and output captured from the terminal, usually rendered with a light grey background.
command	Name of a command that can be entered on the command line.
VARIABLE	Name of a variable or pointer to content of a variable, as in`$VARNAME`.
option	Option to a command, as in "the `-a` option to the**ls** command".
argument	Argument to a command, as in "read **man *ls*** ".
prompt	User prompt, usually followed by a command that you type in a terminal window, like in `hilda@home>`**ls -l**
command options *arguments*	Command synopsis or general usage, on a separated line.
filename	Name of a file or directory, for example "Change to the`/usr/bin` directory."
Key	Keys to hit on the keyboard, such as "type **Q** to quit".
Button	Graphical button to click, like the OK button.
Menu→Choice	Choice to select from a graphical menu, for instance: "SelectHelp→About Mozilla in your browser."
Terminology	Important term or concept: "The Linux *kernel* is the heart of the system."
\	The backslash in a terminal view or command synopsis indicates an unfinished line. In other words, if you see a long command that is cut into multiple lines, \ means "Don't press **Enter** yet!"
See Chapter 1, *Bash and Bash scripts* (page 17)	link to related subject within this guide.
The author[6]	Clickable link to an external web resource.

Table 1.1. Typographic and usage conventions

[6] *http://tille.garrels.be/*

The following images are used:

This is a note

It contains additional information or remarks.

This is a caution

It means be careful.

This is a warning

Be *very* careful.

This is a tip

Tips and tricks.

Organization of this document

This guide discusses concepts useful in the daily life of the serious Bash user. While a basic knowledge of the usage of the shell is required, we start with a discussion of the basic shell components and practices in the first three chapters.

Chapters four to six are discussions of basic tools that are commonly used in shell scripts.

Chapters eight to twelve discuss the most common constructs in shell scripts.

All chapters come with exercises that will test your preparedness for the next chapter.

- Chapter 1, *Bash and Bash scripts* (page 17): Bash basics: why Bash is so good, building blocks, first guidelines on developing good scripts.
- Chapter 2, *Writing and debugging scripts* (page 36): Script basics: writing and debugging.
- Chapter 3, *The Bash environment* (page 46): The Bash Environment: initialization files, variables, quoting characters, shell expansion order, aliases, options.
- Chapter 4, *Regular expressions* (page 78): Regular expressions: an introduction.
- Chapter 5, *The GNU sed stream editor* (page 86): Sed: an introduction to the sed line editor.
- Chapter 6, *The GNU awk programming language* (page 94): Awk: introduction to the awk programming language.

- Chapter 7, *Conditional statements* (page 106): Conditional statements: constructs used in Bash to test conditions.
- Chapter 8, *Writing interactive scripts* (page 124): Interactive scripts: making scripts user-friendly, catching user input.
- Chapter 9, *Repetitive tasks* (page 141): Executing commands repetitively: constructs used in Bash to automate command execution.
- Chapter 10, *More on variables* (page 157): Advanced variables: specifying variable types, introduction to arrays of variables, operations on variables.
- Chapter 11, *Functions* (page 168): Functions: an introduction.
- Chapter 12, *Catching signals* (page 175): Catching signals: introduction to process signalling, trapping user-sent signals.

Chapter 1. Bash and Bash scripts

Abstract

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 and searches for those commands on the system (see Section 1.2, *Advantages of the Bourne Again SHell*), 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 is intended to conform to the IEEE POSIX P1003.2/ISO 9945.2 Shell and Tools standard. 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.

Like the other GNU projects, the bash initiative was started to preserve, protect and promote the freedom to use, study, copy, modify and redistribute software. It is generally known that such conditions stimulate creativity. This was also the case with the bash program, which has a lot of extra features that other shells can't offer.

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`

This file is usually referred to in `~/.bash_profile`:

```
if [ -f ~/.bashrc ]; then . ~/.bashrc; fi
```

See Chapter 7, *Conditional statements* 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**:

- `~/.bashrc`

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, freely available from *http://www.openssh.org*. Different client programs are available for non-UNIX systems as well, see your local software mirror.

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, *The Bash environment*).

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, *Shell built-in commands*.
- 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, *Variables*
- Section 3.6, *More Bash options*

- See Chapter 12, *Catching signals* for more about signals.
- Section 3.4, *Shell expansion* 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, *Conditional statements*.

More information about the file descriptors in Section 8.2.3, *Redirection and file descriptors*.

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, *The Bash environment*.

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, *Aliases*.

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, *More on variables*.

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, *Variables*
- Section 3.6, *More Bash options*
- Info Bash → Basic Shell Features → Redirections
- Section 8.2.3, *Redirection and file descriptors*: 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*.

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, *The Bash environment*. 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, *Functions*.

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, *Variables*, advanced use of variables in Chapter 10, *More on variables*.

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, *Shell expansion*.

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.

1.4.1.7. Executing commands

When executing a command, the words that the parser has marked as variable assignments (preceding the command name) and redirections are saved for later

reference. Words that are not variable assignments or redirections are expanded; the first remaining word after expansion is taken to be the name of the command and the rest are arguments to that command. Then redirections are performed, then strings assigned to variables are expanded. If no command name results, variables will affect the current shell environment.

An important part of the tasks of the shell is to search for commands. Bash does this as follows:

- Check whether the command contains slashes. If not, first check with the function list to see if it contains a command by the name we are looking for.
- If command is not a function, check for it in the built-in list.
- If command is neither a function nor a built-in, look for it analyzing the directories listed in `PATH`. Bash uses a *hash table* (data storage area in memory) to remember the full path names of executables so extensive `PATH` searches can be avoided.
- If the search is unsuccessful, bash prints an error message and returns an exit status of 127.
- If the search was successful or if the command contains slashes, the shell executes the command in a separate execution environment.
- If execution fails because the file is not executable and not a directory, it is assumed to be a shell script.
- If the command was not begun asynchronously, the shell waits for the command to complete and collects its exit status.

1.4.1.8. Shell scripts

When a file containing shell commands is used as the first non-option argument when invoking Bash (without `-c` or `-s`, this will create a non-interactive shell. This shell first searches for the script file in the current directory, then looks in `PATH` if the file cannot be found there.

1.5. Developing good scripts

1.5.1. Properties of good scripts

This guide is mainly about the last shell building block, scripts. Some general considerations before we continue:

1. A script should run without errors.
2. It should perform the task for which it is intended.

3. Program logic is clearly defined and apparent.
4. A script does not do unnecessary work.
5. Scripts should be reusable.

1.5.2. Structure

The structure of a shell script is very flexible. Even though in Bash a lot of freedom is granted, you must ensure correct logic, flow control and efficiency so that users executing the script can do so easily and correctly.

When starting on a new script, ask yourself the following questions:

- Will I be needing any information from the user or from the user's environment?
- How will I store that information?
- Are there any files that need to be created? Where and with which permissions and ownerships?
- What commands will I use? When using the script on different systems, do all these systems have these commands in the required versions?
- Does the user need any notifications? When and why?

1.5.3. Terminology

The table below gives an overview of programming terms that you need to be familiar with:

Term	What is it?
Command control	Testing exit status of a command in order to determine whether a portion of the program should be executed.
Conditional branch	Logical point in the program when a condition determines what happens next.
Logic flow	The overall design of the program. Determines logical sequence of tasks so that the result is successful and controlled.
Loop	Part of the program that is performed zero or more times.
User input	Information provided by an external source while the program is running, can be stored and recalled when needed.

Table 1.1. Overview of programming terms

1.5.4. A word on order and logic

In order to speed up the developing process, the logical order of a program should be thought over in advance. This is your first step when developing a script.

A number of methods can be used; one of the most common is working with lists. Itemizing the list of tasks involved in a program allows you to describe each process. Individual tasks can be referenced by their item number.

Using your own spoken language to pin down the tasks to be executed by your program will help you to create an understandable form of your program. Later, you can replace the everyday language statements with shell language words and constructs.

The example below shows such a logic flow design. It describes the rotation of log files. This example shows a possible repetitive loop, controlled by the number of base log files you want to rotate:

1. Do you want to rotate logs?
   - If yes:
     - Enter directory name containing the logs to be rotated.
     - Enter base name of the log file.
     - Enter number of days logs should be kept.
     - Make settings permanent in user's crontab file.
   - If no, go to step 3.
2. Do you want to rotate another set of logs?
   - If yes: repeat step 1.
   - If no: go to step 3.
3. Exit

The user should provide information for the program to do something. Input from the user must be obtained and stored. The user should be notified that his crontab will change.

1.5.5. An example Bash script: mysystem.sh

The `mysystem.sh` script below executes some well-known commands (**date**, **w**, **uname**, **uptime**) to display information about you and your machine.

```
tom:~> cat -n mysystem.sh
     1  #!/bin/bash
     2  clear
     3  echo "This is inf-tion provided by mysystem.sh. Program starts now."
     4
```

```
     5  echo "Hello, $USER"
     6  echo
     7
     8  echo "Today's date is `date`, this is week `date +"%V"`."
     9  echo
    10
    11  echo "These users are currently connected:"
    12  w | cut -d " " -f 1 - | grep -v USER | sort -u
    13  echo
    14
    15  echo "This is `uname -s` running on a `uname -m` processor."
    16  echo
    17
    18  echo "This is the uptime information:"
    19  uptime
    20  echo
    21
    22  echo "That's all folks!"
```

A script always starts with the same two characters, "#!". After that, the shell that will execute the commands following the first line is defined. This script starts with clearing the screen on line 2. Line 3 makes it print a message, informing the user about what is going to happen. Line 5 greets the user. Lines 6, 9, 13, 16 and 20 are only there for orderly output display purposes. Line 8 prints the current date and the number of the week. Line 11 is again an informative message, like lines 3, 18 and 22. Line 12 formats the output of the **w**; line 15 shows operating system and CPU information. Line 19 gives the uptime and load information.

Both **echo** and **printf** are Bash built-in commands. The first always exits with a 0 status, and simply prints arguments followed by an end of line character on the standard output, while the latter allows for definition of a formatting string and gives a non-zero exit status code upon failure.

This is the same script using the **printf** built-in:

```
tom:~> cat mysystem.sh
#!/bin/bash
clear
printf "This is information provided by mysystem.sh.  Program starts now.\n"

printf "Hello, $USER.\n\n"

printf "Today's date is `date`, this is week `date +"%V"`.\n\n"

printf "These users are currently connected:\n"
w | cut -d " " -f 1 - | grep -v USER | sort -u
printf "\n"

printf "This is `uname -s` running on a `uname -m` processor.\n\n"

printf "This is the uptime information:\n"
```

```
uptime
printf "\n"

printf "That's all folks!\n"
```

Creating user friendly scripts by means of inserting messages is treated in Chapter 8, *Writing interactive scripts*.

Standard location of the Bourne Again shell

This implies that the **bash** program is installed in `/bin`.

If stdout is not available

If you execute a script from cron, supply full path names and redirect output and errors. Since the shell runs in non-interactive mode, any errors will cause the script to exit prematurely if you don't think about this.

The following chapters will discuss the details of the above scripts.

1.5.6. Example init script

An init script starts system services on UNIX and Linux machines. The system log daemon, the power management daemon, the name and mail daemons are common examples. These scripts, also known as startup scripts, are stored in a specific location on your system, such as `/etc/rc.d/init.d` or `/etc/init.d`. Init, the initial process, reads its configuration files and decides which services to start or stop in each run level. A run level is a configuration of processes; each system has a single user run level, for instance, for performing administrative tasks, for which the system has to be in an unused state as much as possible, such as recovering a critical file system from a backup. Reboot and shutdown run levels are usually also configured.

The tasks to be executed upon starting a service or stopping it are listed in the startup scripts. It is one of the system administrator's tasks to configure **init**, so that services are started and stopped at the correct moment. When confronted with this task, you need a good understanding of the startup and shutdown procedures on your system. We therefore advise that you read the man pages for **init** and `inittab` before starting on your own initialization scripts.

Here is a very simple example, that will play a sound upon starting and stopping your machine:

```
#!/bin/bash

# This script is for /etc/rc.d/init.d
# Link in rc3.d/S99audio-greeting and rc0.d/K01audio-greeting
```

```
case "$1" in
'start')
  cat /usr/share/audio/at_your_service.au > /dev/audio
  ;;
'stop')
  cat /usr/share/audio/oh_no_not_again.au > /dev/audio
  ;;
esac
exit 0
```

The **case** statement often used in this kind of script is described in Section 7.2.5, *Using the exit statement and if*.

1.6. 6. Summary

Bash is the GNU shell, compatible with the Bourne shell and incorporating many useful features from other shells. When the shell is started, it reads its configuration files. The most important are:

- `/etc/profile`
- `~/.bash_profile`
- `~/.bashrc`

Bash behaves different when in interactive mode and also has a POSIX compliant and a restricted mode.

Shell commands can be split up in three groups: the shell functions, shell built-ins and existing commands in a directory on your system. Bash supports additional built-ins not found in the plain Bourne shell.

Shell scripts consist of these commands arranged as shell syntax dictates. Scripts are read and executed line per line and should have a logical structure.

1.7. Exercises

These are some exercises to warm you up for the next chapter:

1. Where is the **bash** program located on your system?
2. Use the `--version` option to find out which version you are running.
3. Which shell configuration files are read when you login to your system using the graphical user interface and then opening a terminal window?
4. Are the following shells interactive shells? Are they login shells?
   - A shell opened by clicking on the background of your graphical desktop, selecting “Terminal” or such from a menu.
   - A shell that you get after issuing the command **ssh** ***`localhost`***.

- A shell that you get when logging in to the console in text mode.
- A shell obtained by the command **xterm &**.
- A shell opened by the **mysystem.sh** script.
- A shell that you get on a remote host, for which you didn't have to give the login and/or password because you use SSH and maybe SSH keys.

5. Can you explain why **bash** does not exit when you type **Ctrl+C** on the command line?
6. Display directory stack content.
7. If it is not yet the case, set your prompt so that it displays your location in the file system hierarchy, for instance add this line to `~/.bashrc`:

```
export PS1="\u@\h \w> "
```

8. Display hashed commands for your current shell session.
9. How many processes are currently running on your system? Use **ps** and **wc**, the first line of output of **ps** is not a process!
10. How to display the system hostname? Only the name, nothing more!

Glossary

Abstract

This section contains an alphabetical overview of common UNIX commands. More information about the usage can be found in the man or info pages.

A

a2ps

Format files for printing on a PostScript printer.

acroread

PDF viewer.

adduser

Create a new user or update default new user information.

alias

Create a shell alias for a command.

anacron

Execute commands periodically, does not assume continuously running machine.

apropos

Search the whatis database for strings.

apt-get

APT package handling utility.

aspell

Spell checker.

at, atq, atrm

Queue, examine or delete jobs for later execution.

aumix

Adjust audio mixer.

(g)awk

Pattern scanning and processing language.

B

bash

Bourne Again SHell.

batch

Queue, examine or delete jobs for later execution.

bg

Run a job in the background.

bitmap

Bitmap editor and converter utilities for the X window System.

bzip2

A block-sorting file compressor.

C

cat

Concatenate files and print to standard output.

cd

Change directory.

cdp/cdplay

An interactive text-mode program for controlling and playing audio CD Roms under Linux.

cdparanoia

An audio CD reading utility which includes extra data verification features.

cdrecord

Record a CD-R.

chattr

Change file attributes.

chgrp

Change group ownership.

chkconfig

Update or query run level information for system services.

chmod

Change file access permissions.

chown

Change file owner and group.

compress

Compress files.

cp

Copy files and directories.

crontab

Maintain crontab files.

csh

Open a C shell.

cut

Remove sections from each line of file(s).

D

date

Print or set system date and time.

dd

Convert and copy a file (disk dump).

df

Report file system disk usage.

dhcpcd

DHCP client daemon.

diff

Find differences between two files.

dig

Send domain name query packets to name servers.

dmesg

Print or control the kernel ring buffer.

du

Estimate file space usage.

E

echo

Display a line of text.

ediff

Diff to English translator.

egrep

Extended grep.

eject

Unmount and eject removable media.

emacs

Start the Emacs editor.

exec

Invoke subprocess(es).

exit

Exit current shell.

export

Add function(s) to the shell environment.

F

fax2ps

Convert a TIFF facsimile to PostScript.

fdformat

Format floppy disk.

fdisk

Partition table manipulator for Linux.

fetchmail

Fetch mail from a POP, IMAP, ETRN or ODMR-capable server.

fg

Bring a job in the foreground.

file

Determine file type.

find

Find files.

formail

Mail (re)formatter.

fortune

Print a random, hopefully interesting adage.

ftp

Transfer files (unsafe unless anonymous account is used!)services.

G

galeon

Graphical web browser.

gdm

Gnome Display Manager.

(min/a)getty

Control console devices.

gimp

Image manipulation program.

grep

Print lines matching a pattern.

grub

The grub shell.

gv

A PostScript and PDF viewer.

gzip

Compress or expand files.

H

halt

Stop the system.

head

Output the first part of files.

help

Display help on a shell built-in command.

host

DNS lookup utility.

httpd

Apache hypertext transfer protocol server.

I

id

Print real and effective UIDs and GIDs.

ifconfig

Configure network interface or show configuration.

info

Read Info documents.

init

Process control initialization.

iostat

Display I/O statistics.

ip

Display/change network interface status.

ipchains

IP firewall administration.

iptables

IP packet filter administration.

J

jar

Java archive tool.

jobs

List backgrounded tasks.

K

kdm

Desktop manager for KDE.

kill(all)

Terminate process(es).

ksh

Open a Korn shell.

L

ldapmodify

Modify an LDAP entry.

ldapsearch

LDAP search tool.

less

more with features.

lilo

Linux boot loader.

links

Text mode WWW browser.

ln

Make links between files.

loadkeys

Load keyboard translation tables.

locate

Find files.

logout

Close current shell.

lp

Send requests to the LP print service.

lpc

Line printer control program.

lpq

Print spool queue examination program.

lpr

Offline print.

lprm

Remove print requests.

ls

List directory content.

lynx

Text mode WWW browser.

M

mail

Send and receive mail.

man

Read man pages.

mcopy

Copy MSDOS files to/from Unix.

mdir

Display an MSDOS directory.

memusage

Display memory usage.

memusagestat

Display memory usage statistics.

mesg

Control write access to your terminal.

mformat

Add an MSDOS file system to a low-level formatted floppy disk.

mkbootdisk

Creates a stand-alone boot floppy for the running system.

mkdir

Create directory.

mkisofs

Create a hybrid ISO9660 filesystem.

more

Filter for displaying text one screen at the time.

mount

Mount a file system or display information about mounted file systems.

mozilla

Web browser.

mt

Control magnetic tape drive operation.

mtr

Network diagnostic tool.

mv

Rename files.

N

named

Internet domain name server.

ncftp

Browser program for ftp services (insecure!).

netstat

Print network connections, routing tables, interface statistics, masquerade connections, and multi-cast memberships.

nfsstat

Print statistics about networked file systems.

nice

Run a program with modified scheduling priority.

nmap

Network exploration tool and security scanner.

ntsysv

Simple interface for configuring run levels.

P

passwd

Change password.

pdf2ps

Ghostscript PDF to PostScript translator.

perl

Practical Extraction and Report Language.

pg

Page through text output.

ping

Send echo request to a host.

pr

Convert text files for printing.

printenv

Print all or part of environment.

procmail

Autonomous mail processor.

ps

Report process status.

pstree

Display a tree of processes.

pwd

Print present working directory.

Q

quota

Display disk usage and limits.

R

rcp

Remote copy (unsafe!)

rdesktop

Remote Desktop Protocol client.

reboot

Stop and restart the system.

renice

Alter priority of a running process.

rlogin

Remote login (telnet, insecure!).

rm

Remove a file.

rmdir

Remove a directory.

rpm

RPM Package Manager.

rsh

Remote shell (insecure!).

S

scp

Secure remote copy.

screen

Screen manager with VT100 emulation.

set

Display, set or change variable.

setterm

Set terminal attributes.

sftp

Secure (encrypted) ftp.

sh

Open a standard shell.

shutdown

Bring the system down.

sleep

Wait for a given period.

slocate

Security Enhanced version of the GNU Locate.

slrnn

Text mode Usenet client.

snort

Network intrusion detection tool.

sort

Sort lines of text files.

source

Read commands from file.

ssh

Secure shell.

ssh-keygen

Authentication key generation.

stty

Change and print terminal line settings.

su

Switch user.

T

tac

Concatenate and print files in reverse.

tail

Output the last part of files.

talk

Talk to a user.

tar

Archiving utility.

tcsh

Open a Turbo C shell.

telnet

User interface to the TELNET protocol (insecure!).

tex

Text formatting and typesetting.

time

Time a simple command or give resource usage.

tin

News reading program.

top

Display top CPU processes.

touch

Change file timestamps.

traceroute

Print the route packets take to network host.

tripwire

A file integrity checker for UNIX systems.

twm

Tab Window Manager for the X Window System.

U

ulimit

Controll resources.

umask

Set user file creation mask.

umount

Unmount a file system.

uncompress

Decompress compressed files.

uniq

Remove duplicate lines from a sorted file.

update

Kernel daemon to flush dirty buffers back to disk.

uptime

Display system uptime and average load.

userdel

Delete a user account and related files.

V

vi(m)

Start the vi (improved) editor.

vimtutor

The Vim tutor.

vmstat

Report virtual memory statistics.

W

w

Show who is logged on and what they are doing.

wall

Send a message to everybody's terminal.

wc

Print the number of bytes, words and lines in files.

which

Shows the full path of (shell) commands.

who

Show who is logged on.

who am i

Print effective user ID.

whois

Query a whois or nicname database.

write

Send a message to another user.

X

xauth

X authority file utility.

xcdroast

Graphical front end to cdrecord.

xclock

Analog/digital clock for X.

xconsole

Monitor system console messages with X.

xdm

X Display Manager with support for XDMCP, host chooser.

xdvi

DVI viewer.

xfs

X font server.

xhost

Server access control program for X

xinetd

The extended Internet services daemon.

xload

System load average display for X.

xlsfonts

Server font list displayer for X.

xmms

Audio player for X.

xpdf

PDF viewer.

xterm

Terminal emulator for X.

Z

zcat

Compress or expand files.

zgrep

Search possibly compressed files for a regular expression.

zmore

Filter for viewing compressed text.

Index

.

.bash_login ... 49
.bash_logout ... 51
.bash_profile ... 49
.bashrc ... 50
.profile ... 49

/

/etc/bashrc ... 47
/etc/passwd ... 17
/etc/profile ... 46
/etc/shells ... 17

A

aliases ... 73
ANSI-C quoting ... 66
arguments ... 113, 125, 141, 153
arithmetic expansion ... 71
arithmetic operators ... 71
array ... 159
awk ... 94
awkprogram ... 95

B

bash ... 17, 18
batch editor ... 86
boolean operators ... 117, 122, 156
Bourne shell ... 17
brace expansion ... 67
break ... 148, 155
built-in commands ... 22, 25

C

case statements ... 34, 115, 117
character classes ... 81, 84
child process ... 25
combined expressions ... 107
command substitution ... 70, 117, 158, 163
comments ... 40
conditionals ... 106
configuration files ... 46
constants ... 159
continue ... 150
control signals ... 176
creating variables ... 55
csh ... 17

D

debugging scripts ... 41
declare ... 158, 159
double quotes ... 65

E

echo ... 31, 37, 42, 52, 72, 125
editors ... 36
else ... 112
emacs ... 36
env ... 52
esac ... 34, 115, 117

escape characters 65, 97, 127
escape sequences 125
exec 25, 134
execute permissions 38
execution 38
exit 34, 115, 117
exit status 110, 113
expansion 23, 28, 66
export 56
extended regular expressions 79

F

file descriptors 23, 25, 71, 130, 133
file name expansion 72
find and replace 89
for 141
fork 25
functions 168

G

gawk 94
gawk commands 95
gawk fields 95
gawk formatting 97, 99
gawk scripts 99
gawk variables 92, 100
gedit 36
global variables 52
globbing 42, 52, 72
grep 79

H

here document 137, 154

I

if 106
init 25, 33
initialization files 46
input field separator 57, 61, 92, 100, 141
interactive editing 87
interactive scripts 124
interactive shell 19, 20, 21
invocation 19

K

kill 176
killall 176
ksh 17

L

length of a variable 163
line anchors 81
locale 66
locate 17, 36
logic flow 31
login shell 19

M

menu 151
metacharacters 78

N

nested if statements 116
noglob 42, 52, 72
non-interactive editing 91
non-interactive shell 20
non-login shell 20
numeric comparisons 71, 111

O

options 75
output field separator 101
output record separator 101

P

parameter expansion....69
PATH....37
pattern matching....83
positionalparams....61, 141, 169
POSIX....18
POSIX mode....20
primary expressions....107
printenv....52
printf....31, 103
process substitution....72, 128
Process substitution....72, 128
prompt....51

Q

quoting characters....65

R

rbash....24
read....127
readonly....159
redirection....23, 25, 28, 71, 76, 130, 147
regular expression operators....78, 87, 98
regular expressions....78
remote invocation....21
removing aliases....48, 74
reserved variables....57
return....169

S

sed....86
sed editing commands....87
sed options....87
sed script....91, 103
select....151
set....54, 75, 170
shift....153, 156
signals....175
single quotes....65
source....38
special parameters....61, 141
special variables....61, 141
standard error....130
standard input....130
standard output....130
string comparisons....111, 164
stty....175
submenu....153
subshell....40
substitution....164, 166
substring....165
syntax....27, 66

T

tcsh....17
terminology....30
then....109
tilde expansion....68
transformation of variables....164
traps....177
true....144, 146

U

unalias....48, 73, 74
unset....55, 161, 170
until....146
user input....127, 129
user messages....124

V

variable expansion....69
variables....22, 25, 28, 47, 52, 157
verbose....42, 52, 72
vi(m)....36

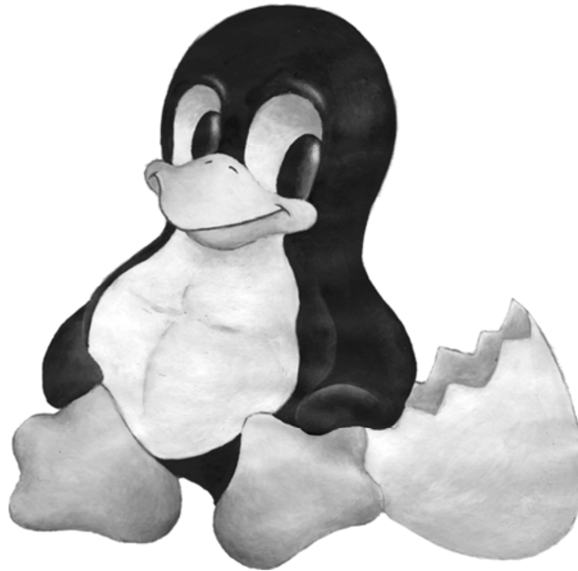
W

wait....178
whereis....36
which....36
while....143
wildcards....82
word anchors....81
word splitting....72

X

xtrace....41, 42, 52, 72

Linbrary™ Advertising Club (LAC)



Official Docs as a Real Books
http://www.linbrary.com

Linux Library

Linbrary Advertising Club

Linux Documentation Project - Machtelt Garrels

http://www.tldp.org/

Version	Title	Edition	ISBN- 10	ISBN- 13
TLDP	**Introduction to Linux** (Third Edition)	paperback	1-59682-199-X	978-1-59682-199-6
		eBook (pdf)	1-59682-200-7	978-1-59682-200-9
	Bash Guide for Beginners (Second Edition)	paperback	1-59682-201-5	978-1-59682-201-6
		eBook (pdf)	1-59682-202-3	978-1-59682-202-3
http://www.linbrary.com/linux-tldp/				

Fedora Project Official Documentation

http://fedoraproject.org/

Version	Title	Edition	ISBN- 10	ISBN- 13
Fedora 12	Fedora 12 **Installation Guide**	paperback	1-59682-179-5	978-1-59682-179-8
		eBook (pdf)	1-59682-184-1	978-1-59682-184-2
	Fedora 12 **User Guide**	paperback	1-59682-180-9	978-1-59682-180-4
		eBook (pdf)	1-59682-185-X	978-1-59682-185-9
	Fedora 12 **Security Guide**	paperback	1-59682-181-7	978-1-59682-181-1
		eBook (pdf)	1-59682-186-8	978-1-59682-186-6
	Fedora 12 **SE Linux User Guide**	paperback	1-59682-182-5	978-1-59682-182-8
		eBook (pdf)	1-59682-187-6	978-1-59682-187-3
	Fedora 12 **Virtualization Guide**	paperback	1-59682-183-3	978-1-59682-183-5
		eBook (pdf)	1-59682-188-4	978-1-59682-188-0
Fedora 11	Fedora 12 **Installation Guide**	paperback	1-59682-142-6	978-1-59682-142-2
		eBook (pdf)	1-59682-146-9	978-1-59682-146-0
	Fedora 12 **User Guide**	paperback	1-59682-180-9	978-1-59682-180-4
		eBook (pdf)	1-59682-185-X	978-1-59682-185-9
	Fedora 12 **Security Guide**	paperback	1-59682-181-7	978-1-59682-181-1
		eBook (pdf)	1-59682-186-8	978-1-59682-186-6
	Fedora 12 **SE Linux User Guide**	paperback	1-59682-182-5	978-1-59682-182-8
		eBook (pdf)	1-59682-187-6	978-1-59682-187-3
http://www.linbrary.com/fedora/				

Linbrary Advertising Club

Ubuntu Official Documentation

http://www.ubuntu.com/

Version	Title	Edition	ISBN- 10	ISBN- 13
Ubuntu 10.04 LTS	Ubuntu 10.04 LTS **Installation Guide**	paperback	1-59682-203-1	978-1-59682-203-0
		eBook (pdf)	1-59682-207-4	978-1-59682-207-8
	Ubuntu 10.04 LTS **Desktop Guide**	paperback	1-59682-204-X	978-1-59682-204-7
		eBook (pdf)	1-59682-208-2	978-1-59682-208-5
	Ubuntu 10.04 LTS **Server Guide**	paperback	1-59682-205-8	978-1-59682-205-4
		eBook (pdf)	1-59682-209-0	978-1-59682-209-2
	Ubuntu 10.04 LTS **Packaging Guide**	paperback	1-59682-206-6	978-1-59682-206-1
		eBook (pdf)	1-59682-210-4	978-1-59682-210-8
Ubuntu 9.10	Ubuntu 9.10 **Installation Guide**	paperback	1-59682-171-X	978-1-59682-171-2
		eBook (pdf)	1-59682-175-2	978-1-59682-175-0
	Ubuntu 9.10 **Desktop Guide**	paperback	1-59682-172-8	978-1-59682-172-9
		eBook (pdf)	1-59682-176-0	978-1-59682-176-7
	Ubuntu 9.10 **Server Guide**	paperback	1-59682-173-6	978-1-59682-173-6
		eBook (pdf)	1-59682-177-9	978-1-59682-177-4
	Ubuntu 9.10 **Packaging Guide**	paperback	1-59682-174-4	978-1-59682-174-3
		eBook (pdf)	1-59682-178-7	978-1-59682-178-1
Ubuntu 9.04	Ubuntu 9.04 **Installation Guide**	paperback	1-59682-150-7	978-1-59682-150-7
		eBook (pdf)	1-59682-154-X	978-1-59682-154-5
	Ubuntu 9.04 **Desktop Guide**	paperback	1-59682-151-5	978-1-59682-151-4
		eBook (pdf)	1-59682-155-8	978-1-59682-155-2
	Ubuntu 9.04 **Server Guide**	paperback	1-59682-152-3	978-1-59682-152-1
		eBook (pdf)	1-59682-156-6	978-1-59682-156-9
	Ubuntu 9.04 **Packaging Guide**	paperback	1-59682-153-1	978-1-59682-153-8
		eBook (pdf)	1-59682-157-4	978-1-59682-157-6
http://www.linbrary.com/ubuntu/				

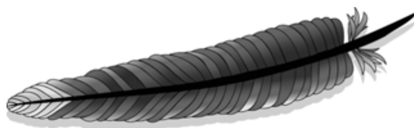
Linbrary Advertising Club

PostgreSQL Official Documentation

http://www.postgresq.org/

Version	Title	Edition	ISBN- 10	ISBN- 13
PostgreSQL 8.04	PostgreSQL 8.04 **Volume I. The SQL Language**	paperback	1-59682-158-2	978-1-59682-158-3
		eBook (pdf)	1-59682-163-9	978-1-59682-163-7
	PostgreSQL 8.04 **Volume II. Server Administration**	paperback	1-59682-159-0	978-1-59682-159-0
		eBook (pdf)	1-59682-164-7	978-1-59682-164-4
	PostgreSQL 8.04 **Volume III. Server Programming**	paperback	1-59682-160-4	978-1-59682-160-6
		eBook (pdf)	1-59682-165-5	978-1-59682-165-1
	PostgreSQL 8.04 **Volume IV. Reference**	paperback	1-59682-161-2	978-1-59682-161-3
		eBook (pdf)	1-59682-166-3	978-1-59682-166-8
	PostgreSQL 8.04 **Volume V. Internals & Appendixes**	paperback	1-59682-162-0	978-1-59682-162-0
		eBook (pdf)	1-59682-167-1	978-1-59682-167-5
http://www.linbrary.com/postgresql/				

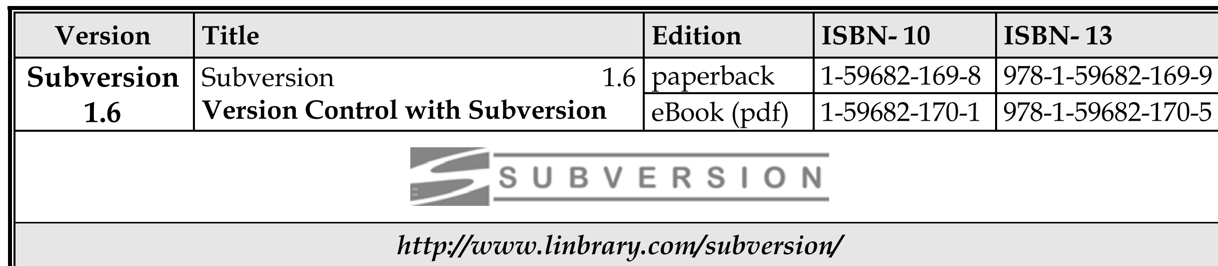
Linbrary Advertising Club



The Apache Software Foundation Official Documentation

http://www.apache.org/

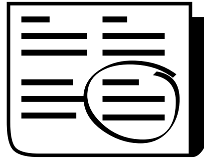
Version	Title	Edition	ISBN- 10	ISBN- 13
Apache Web Server 2.2	Apache HTTP Server 2.2 **Vol.I. Server Administration**	paperback	1-59682-191-4	978-1-59682-191-0
		eBook (pdf)	1-59682-195-7	978-1-59682-195-8
	Apache HTTP Server 2.2 **Vol.II. Security & Server Programs**	paperback	1-59682-192-2	978-1-59682-192-7
		eBook (pdf)	1-59682-196-5	978-1-59682-196-5
	Apache HTTP Server 2.2 **Vol.III. Modules (A-H)**	paperback	1-59682-193-0	978-1-59682-193-4
		eBook (pdf)	1-59682-197-3	978-1-59682-197-2
	Apache HTTP Server 2.2 **Vol.IV. Modules (I-V)**	paperback	1-59682-194-9	978-1-59682-194-1
		eBook (pdf)	1-59682-198-1	978-1-59682-198-9
http://www.linbrary.com/apache-http/				



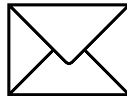
Version	Title	Edition	ISBN- 10	ISBN- 13
Subversion 1.6	Subversion 1.6 **Version Control with Subversion**	paperback	1-59682-169-8	978-1-59682-169-9
		eBook (pdf)	1-59682-170-1	978-1-59682-170-5
SUBVERSION				
http://www.linbrary.com/subversion/				

Linbrary Advertising Club

Your Advertising Here



More Books Coming Soon!!!



Please Feel Free to Contact Us at

production@fultus.com