100 SHELL PROGRAMS IN UNIX

By

Sarika Jain
Head, MCA Department
ABSS, Meerut (U.P.)

Shivani Jain
Senior Lecturer, Department of Computer Science
VCE, Meerut (U.P.)

FIREWAL MEDIA

(An Imprint of Laxmi Publications Pvt. Ltd.)

BANGALORE ● CHENNAI ● COCHIN ● GUWAHATI ● HYDERABAD
JALANDHAR ● KOLKATA ● LUCKNOW ● MUMBAI ● RANCHI
NEW DELHI

Published by :

FIREWAL MEDIA

(*An Imprint of Laxmi Publications Pvt. Ltd.*)
113, Golden House, Daryaganj,
New Delhi-110002

Phone : 011-43 53 25 00
Fax : 011-43 53 25 28

www.laxmipublications.com
info@laxmipublications.com

Price : **Rs. 100.00** *Only.* *First Edition* : 2009

OFFICES

✆ **Bangalore**	080-26 61 15 61	✆ **Chennai**	044-24 34 47 26
✆ **Cochin**	0484-239 70 04	✆ **Guwahati**	0361-254 36 69, 251 38 81
✆ **Hyderabad**	040-24 65 23 33	✆ **Jalandhar**	0181-222 12 72
✆ **Kolkata**	033-22 27 43 84	✆ **Lucknow**	0522-220 95 78
✆ **Mumbai**	022-24 91 54 15, 24 92 78 69	✆ **Ranchi**	0651-221 47 64

FSH-3030-100-100 SHELL PROG IN UNIX **C—17102/09/02**

Typeset at : Kalyani Computer Services, New Delhi. *Printed at* : Mehra Offset Press, Delhi.

Contents

PREFACE

The UNIX system is so successful. Why? *First*, because UNIX is portable, i.e., runs on a range of computers and adapts to particular requirements. *Second*, the UNIX programming environment is unusually rich and productive. The UNIX system has become very popular, and there are number of versions in wide use. Regardless of the version you run on your system, the difference in coding you find will be minor.

The book's small size is meant to keep your investment in time down to a minimum but with the greatest possible amount of knowledge. This book is organized as follows: Part I is an introduction to the most basic use of the system. It covers logging in, the file system, commonly used commands, and logging out. Part II contains 100 programs (including shell script and programs in C).

The best way to learn something is by doing it. Kindly practice the programs and verify or contradict what we say. All the examples in this text are actual, runnable code tested on UNIX system.

As a reader of this book, you are the most important critic and commentator. You can email or write to us directly to let us know what you did or didn't like about this book – as well as what we can do to make our book stronger.

—AUTHORS

ACKNOWLEDGEMENT

We are grateful to many people for constructive comments and criticisms, and for their help in improving our code. The work of an author is only as good as the support from their family members and friends. Sarika Jain would like to specially thank her husband, Anuj Jain for letting her off all her household chores while working in the tree house on this project. Likewise, Shivani Jain wants to thank her all family members for their understanding and encouragement throughout this project. We cound not have done this without all of you.

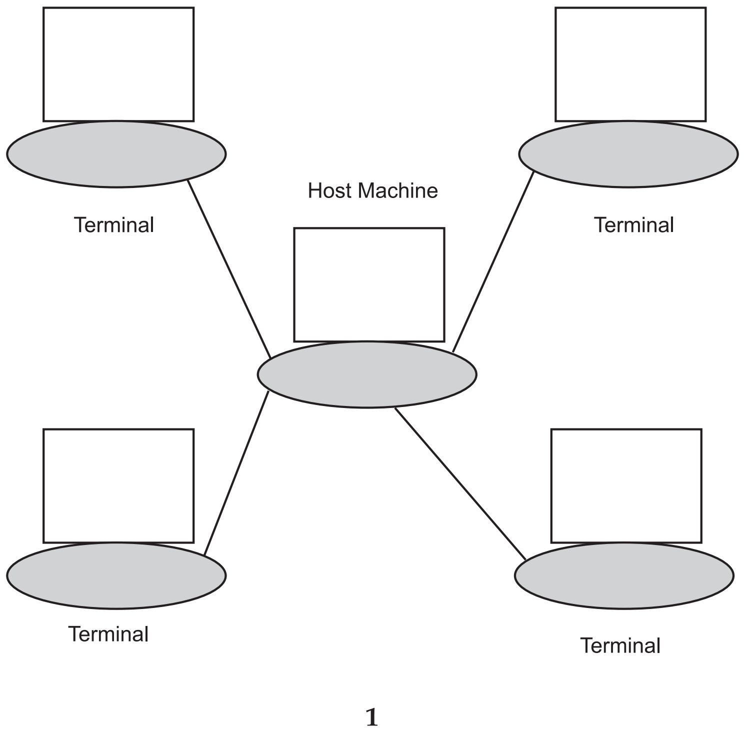
—AUTHORS

PART I

INTRODUCTION

I. AN OVERVIEW

UNIX is an operating system which was first developed in the 1960s, and has been under constant development ever since. By operating system, we mean the suite of programs, which make the computer work. It is a stable, multi-user, multi-tasking system for servers, desktops and laptops. UNIX systems also have a graphical user interface (GUI) similar to Microsoft Windows, which provides an easy to use environment. UNIX (and Linux, which is Linus Torvald's version of UNIX) has deep roots in the computer industry. UNIX is a very powerful multitasking and multi-user system. Multitasking means a user can run multiple programs simultaneously with in one single login of the system. Multi-user means that many users can simultaneously and securely use the same machine with their separate dumb terminals. The following figure shows a typical UNIX setup:



II. SALIENT FEATURES OF UNIX

Among many salient features the UNIX offers, few are listed below:

- Multi-user
- Multitasking
- Communication
- Security
- Portability
- Capability
- Time sharing
- Command interpreter & background processing
- Hierarchical file system
- Dos-Unix interface
- Simple command
- System administration & job accounting
- Tools & utilities
- Shell programming
- Availability of 4GL and RDBMS
- Library of application packages

III. HARDWARE REQUIREMENTS FOR UNIX

There are some prerequisites for a system that can host and take best advantage of UNIX. These are a PC/AT or higher with an 80 MB hard disk and at least 4MB of RAM on a 16-bit microprocessor (80286/80386/80486). The dumb terminals are connected to the host machine through a 4/8/16 port controller card installed in the expansion slot on the motherboard of the host machine. More the number of terminals more should be the memory on the host machine. Out of 80 MB disk space, almost 40MB is eaten away by the actual UNIX OS files and another 10-20 MB is used as swap space. For each terminal to be supported, 0.75 to 1 MB should be present in the host machine.

IV. GETTING STARTED

A system administrator supervises the working of UNIX on any installation. In UNIX, there are different types of accounts. The root account is the administrator user account. It has the most privileges available to the system. Then are individual user accounts having far fewer privileges. Access to many user accounts can also be controlled at once by assigning users to groups. UNIX is case sensitive and is strongly oriented towards devices with lower case. The system administrator configures every individual on the system and supplies them with credentials (username and password).

1. Logging-in

Given that your terminal is connected to the host computer and is powered on, the display prompts you for your login name.

```
login:
```

When you get the login: message, type your login name. Follow it by pressing RETURN, after which you receive the password prompt.

```
password:
```

At this stage, you must type in your password.

You get three to five attempts to get the login — password combination right before your terminal is disconnected. Once you successfully login, you get a *prompt*, usually a single character, indicating that the system is ready to accept commands from you. The prompt is most likely to be a dollar sign ($) (for Bourne Shell), or a percent sign (%) (for C Shell), but you can change it to anything you like.

2. Typing Commands

On receiving the prompt, commands can be typed. When you see the prompt ($), type who am I and press RETURN.

```
$ who am i
tom tty 3a Jul 18 10:10
```

The system should reply with your user name, system's name, and when the user logged on. If you make a mistake typing the name of a command, you will be told that no such command exists:

```
$ today's date
today's date: not found
```

You have two ways to recover from your typing mistakes, provided you see them before you press RETURN:

(*i*) You type the *line kill* character (@). It kills the whole line and you can type the whole line again.

(*ii*) Use erase characters one at a time using #. Each # erases the last character typed. For example,

```
$ who a i@
who am i
tom tty3a Jul 18 10:10
$ www##ho aa#mi# i
tom tty3a Jul 18 10:10
```

3. Some Special Keys

- RETURN key — The RETURN key signifies the end of a line of input. On any terminal, RETURN has a key of its own, or return may be typed by holding down the control key and typing a 'm'.
- DELETE: The DELETE key stops a program/command immediately, without waiting for it to finish. DELETE can be achieved equivalently with ctrl-c.

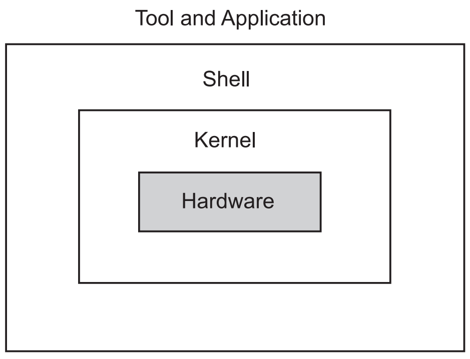
- Ctrl-s: Ctrl-s pauses the output and the program is suspended until you start it again.
- Ctrl-q: Ctrl-q resumes the program paused by ctrl-s.
- Ctrl-g: rings a bell on the terminal.
- Ctrl-h: can be used for backspace.
- Ctrl-I: can be used for tab (eight spaces on UNIX system).

4. **Logging out**

Logout marks the end of a UNIX session. A user can log out by either typing ctrl-d or exit at the prompt.

V. UNIX ARCHITECTURE

Figure below shows the three layers of UNIX operating system. On the outermost layer sits the user with application programs and other utilities. The kernel interacts with the actual hardware. The shell acts as the command interpreter between the user and the kernel.



The Kernel

At the center of the UNIX onion is a program called the kernel. The kernel of UNIX is the hub of the operating system. The kernel has various functions. It manages files, carries out all the data transfer between the file system and the hardware, and also manages memory. It allocates time and memory too.

The Shell

The shell acts as an interface between the user and the kernel. Shell is an intermediate program that accepts the commands, typed at the terminal and executes them to obtain kernel understandable command set. Important features are given below:

1. **Programming Language Constructs**: Shell provides powerful constructs using which exiting commands can be used to frame the job sequences or develop new utilities.
2. **Shell Scripts**: Shell commands and constructs are stored in a file, which can later be used to execute these commands like a program. This file is generally called shell script.
3. **Shell Variables**: Like other programming language one can define variable in shell program also. These variables are identified by prefixing their names with '$' sign.

Variables can be classified into four categories:

- **Standard Variables**: These are predefined in the system and therefore called built-in-variables. They hold certain system information related to particular user environment. These are also called environmental variables.

PS1 : Represent the first prompt of the user, Default is '$'.

```
$echo $PS1
[\u@\h \W]\$
$PS1=#
#
```

PS2 : Represent the second prompt of the user, Default is '>'.

```
$echo $PS2
>
$
```

Logname: User's login name

```
$logname
root
$
```

- **Positional Parameters**: These are variables which receives their values from command-line as an argument. These are identified by their position on the command line as $0, $1, $2,… where $0 holds name of program and others denote another command-line argument.
- **Special Shell Variable**: These are the variables that hold certain other information such as $$ stores process ID of the current shell, $! Stores process ID of last back ground process.
- **User Defined Variables:** Users may define their own variables either in shell script or at the shell prompts. These variables can have any name except for those described above.

UNIX supports different types of shells. Some of these shells are:

- Bourne Shell (sh)
- C Shell (csh)
- Korn Shell (ksh)
- Job Shell (jsh)

VI. UNIX BASIC COMMANDS

A text editor is a program for storing and manipulating information in the computer. Three of the most popular editors in UNIX system are ed, vi and emacs. The ed editor works on any terminal as it takes no advantage of special terminal features.

Vi. Editor

Vi, (stands for Visual Editor,) is one of the most significant tools provided by UNIX and is used to create and edit text files. It is a screen-oriented editor that is extremely fast when scrolling through large documents. It does not support any document formatting like bold/italics, spell checking or any views of a document, as it will look when printed.

Table: Commands for quiting vi

Commands	*Functions*
ZZ	Write the buffer to the file and quits vi.
:wq	Write the buffer to the file and quits vi.
:w filename	Write the buffer to the file filename (new).
:q	Quits vi if changes made to the buffer were written to a file.
:w!	Overwrites the existing file filename with the contents of the buffer.
:q!	Quits vi whether or not change made to the buffer were written to a file. Does not incorporate changes made to the buffer since the last write (:w) command.

For example, addition of two numbers
Steps to write a program

Step-1:

```
vi  prg1
clear
echo “Input Value of a & b  :”
read a
read b
c=‘expr $a + $b‘
echo $c
```

Step-2:

(*i*). press Esc

(*ii*). :wq

Step-3:

```
sh prg1
```

Result:

Let a = 10

Let b = 12

Output is 22

The reader is motivated to practice the following commands with all possible options

1. **The ls command**: ls [option] filename

The ls command lists the names of files in given directory or in current directory if no filename is specified. The names of files are in ascending order.

[Options]: -l : long listing about each file.

-t : files listed in order in which they were last changed, most recent first.

2. **pwd** command — pwd
3. **mkdir** — mkdir <filename>
4. **cd** — cd <filename>
5. **rmdir** — rmdir <filename>
6. **chmod** — To Assign Permissions to files.

UNIX supports two levels of security one is through login and another security is implemented by assigning different types of access permissions to different files.

UNIX divides all users into three categories:

1. Owner
2. Group
3. Others

Syntax

```
Chmod nnn<file>
```

Where n is a number from 0 to 7 representing an octal value. First n denotes the permission for owner, next n for group and the last n for others. These numbers are:

4:	For Read Permission (r)
2:	For Write Permission (w)
1:	For Execute Permission (x)

To assign more than one permission, respective octal values are added. As to assign read and write permission, octal value will be the sum of 4 (read) and 2 (write), *i.e.,* 6. The permission set by these digits and their sum are given below:

Absolute Value	*Break Up*	*Meaning*
0	-	No permission
1	-	Only execute
2	-	Only write
3	2+1	Write & execute
4	-	Only read
5	4+1	Read & execute
6	4+2	Read & write
7	4+2+1	Read, write, execute

Examples:

```
$ chmod 400 <filename> : owner has only read permission.
$ chmod 700 <filename> : owner has read, write and execute permissions.
$ chmod 777 <filename> : owner, group and others have all permissions.
```

Another method to assigning permissions to files is symbolic method. To change permissions through this method one must specify:

- Type of user (u,g,o).
- Type of permission (r,w,x).
- Whether the permission is to be granted(+) or revoked(-).
- Name of the file.

Examples:

```
$ chmod u+r<file>    : Add read permissions to owner.
$ chmod a+rw<file> : Add read/write permission to all users. (a means all users)
$ chmod –w<file>     : Remove write permission from all users.
```

7. **mv**: Move files.

Syntax:

```
$ mv <file1> <file2>
```

8. **cp:** Copy files.

Syntax:

```
$ cp <file1> <file2>
```

9. **rm:** Remove files

Syntax:

```
$ rm <file1>
```

10. **ln:** Link files.

'ln' command is used for establishing an additional name for the same ordinary file so that it can be shared between two users.

Syntax:

```
$ ln <source> <target>
```

11. **find:** To Find files.

Syntax:

```
$ find <pathname><condition><action>
```

12. **cat:** To view files.

Syntax:

```
$ cat <filename>
```

13. Combine files.

Syntax:

```
$ cat file1.dat file2.bac file3.pqr>file4
```

This command merges the files (file1.dat, file2.bac and file3.pqr) into file4 to make a combined file.

14. **pr**: To Print files.

Syntax:

```
$pr <filename>
```

15. **sort:** To Sort the contents of a file.

Syntax:

```
$ sort <filename>
```

To explain this let us prepare a file:

```
$ cat temp.dat
        Hyderabad
        Delhi
        Lucknow
        Agra
        Banglore
```

Now to arrange file in alphabetic order we can sort the file in this manner :

```
$ sort temp.dat
```

It will display the following result on the screen

```
        Agra
        Banglore
        Delhi
        Hyderabad
        Lucknow
```

16. **cmp:** To compare files.

Syntax:

```
$ cmp <file1> <file2>
```

Result will look like

```
File1 file2 differ: char 280, line 18
```

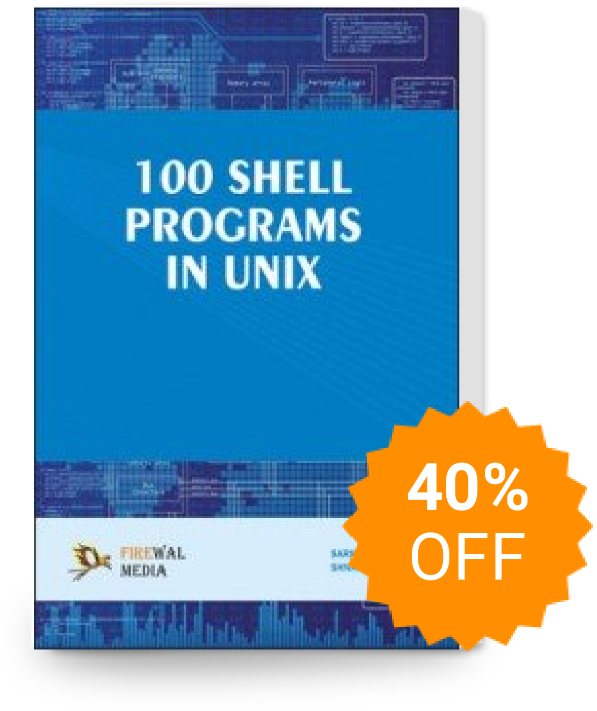
Filters and Pipes

A filter is a program that takes input from the standard input, filters it and sends output to standard output. Some of filters provided by UNIX are grep, pg, wc, tr etc.

Filters and pipes commands

1. **grep** - search a file for keywords.

100 Shell Programs In Unix By Sarika Jain, Shivani Jain



Publisher : Laxmi Publications ISBN : 9788131807088 Author : Sarika Jain, Shivani Jain

Type the URL : http://www.kopykitab.com/product/3452

Get this eBook