SLAC–PUB–5338
October 1990
(M)

UNIX BY EXAMPLES*

by

Frank Nee

Stanford Linear Accelerator Center
Stanford University, Stanford, CA 94309

Presented at the Unix by Examples Seminar,
Stanford, CA, May 16, 1990.

---

* Work supported by Department of Energy contract DE–AC03–76SF00515.

OUTLINE

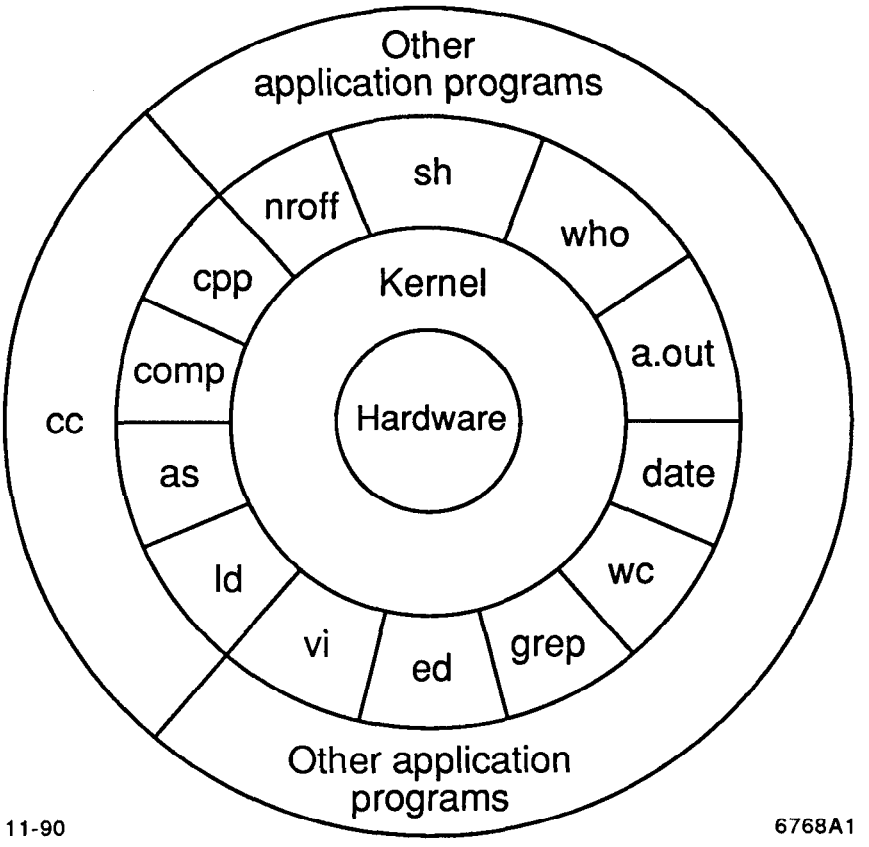
1.0 GENERAL OVERVIEW

1.1 Introduce the "Gang of Three":

Dennis Ritchie	———>	C
Ken Thompson	———>	UNIX
Brian Kernighan	———>	Evangelist

1.2 UNIX Hierarchical Diagram

The following diagram depicts the hierarchical structure of the UNIX system (courtesy of Ref. [3]).



1.3 Features Provided by the UNIX Operating System:

[1] Multitasking — capability of performing two or more computing tasks at the same time.

[2] Graphics — much faster and complicated operations can be achieved.

[3] Networking — networking capability has already been built into UNIX environment, so that resource sharing and file transfer are much easier and faster.

[4] Debugger — most UNIX systems provide some sort of SDB (symbolic debugger) tools to facilitate software development.

[5] Window System — this will give users "windows" into processes and applications not located only or specifically on their own computer system.

1.4 The strengths of the UNIX OS are listed below:

[1] System Code — the majority of UNIX system code (about 95%) is written in programming language C; C is portable because it is written in a high-level, machine independent language.

[2] Networking — "the network is the computer" (resource sharing and file transfer) is achieved through some built-in programs for terminal emulation, file transfer and other built-in communication functions.

[3] Open Systems — this will offer users more options for expanding their networks and more ways to preserve the data and applications they have already invested in. The three ingredients of "open systems" are: portability, scalability and interoperability.

(1) portability: this refers to a user's freedom to run the same application program on computers from different vendors without rewriting the program's code.

(2) scalability: this refers to a user's ability to move applications and data among larger and smaller computer systems to meet changing needs.

(3) interoperability: this refers to the ability to run applications programs on networks built up of different kinds of machines manufactured by different vendors.

1.5 Popular UNIX Versions:

[1] AT&T's System V, the newest AT&T offering.

[2] AT&T's System III, a subset of System V.

[3] Bell Lab's Version 7.

[4] Berkeley's 4.3 BSD.

[5] IEEE's POSIX.

[6] SCO's XENIX.

2.0 FILE STRUCTURE*

2.1 Types of Files

Types of Files
- Ordinary File
- Directory File
- Special File
  - Block
  - Character

Ordinary File: Stores user data, such as textual information and programs.

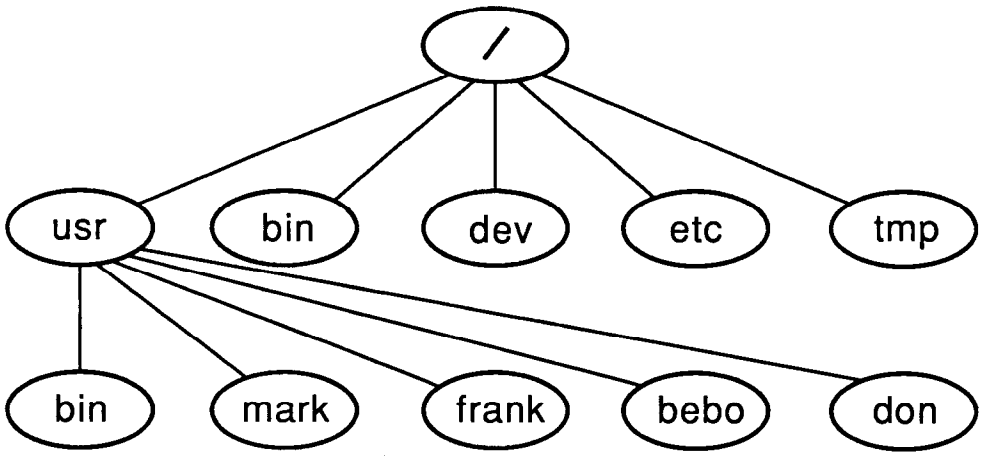
Directory File: A disk file with a standard format that stores a list of names of ordinary files and other directories.

Special File: Represents routines in the Kernel that provides access to some feature of the operating system, such as device drivers that let you communicate with peripheral devices. By convention, special files appear in the /dev directory.

2.2 Important Directories In a File Structure

The UNIX system file structure is called a "tree." It is usually set up according to a convention; this convention may vary from installation to installation.

The following figure depicts the usual locations of some important directories and files.



11-90 6768A2

2.3 Contents of Some Important Directories

/(root)	The root directory is present in all UNIX system file structures. It is the ancestor of all files in the file system.
/usr	Each user's home directory is typically one of many subdirectories of /usr, although many systems use other conventions. Because /usr traditionally includes subdirectories that contain information used by the system, on some systems the users' directories are in a subdirectory of /usr called /usr/users.
/bin and /usr/bin	These directories contain the standard UNIX utility programs. By convention, /bin contains the most frequently used standard utilities, including all those necessary to bring the system up, while /usr/bin contains almost all the rest of the utilities as well as programs that are specific to an installation.
/dev	All files that represent peripheral devices such as terminals and printers are kept in this directory.
/etc	Administrative and configuration programs and other system files are kept here. The most useful is the "passwd" file, containing a list of all users who have permission to use the system.
/tmp	Many programs use this directory to hold temporary files.

*Part of the material in this and the next sections are abridged from Reference 6.

3.0 FREQUENTLY USED COMMANDS/UTILITIES

3.1 Introduction

Commands are those built into the shells, such as: date, ls, mv. Utilities are programs that can do more comprehensive jobs for us, such as: awk, grep, sed.

It's relatively hard to tell the exact number of commands and utilities under the UNIX Operating System, because:

[1] Different versions of UNIX would contain different sets of commands/utilities, such as AT&T System V and U.C. Berkeley BSD 4.3.

[2] New commands/utilities are added after each new release, such as AT&T System III and AT&T System V.

If we want to know more about any command/utility, we can just type "man cmd_name" or "man utility_name" in the UNIX environment; then the syntax, usage, etc., of the respective command/utility will be displayed on the terminal. There is no need to remember the syntax, usage, etc., of commands/utilities.

3.2 Command/Utility Classification

The following is a list of frequently used commands/utilities grouped by function.

Commands/Utilities That Display and Manipulate Files

awk	search for and process patterns in a file [created by Aho, Weinberger, and Kernighan].
cat	join or display files.
comm	compare sorted files.
cp	copy files.
cpio	store and retrieve files in an archive format [System V only].
diff	display the differences between two files.
find	find files.
grep	search for a pattern in files [grep stands for general regular expression parser].
ln	make a link to a file.
lp	print files [System V only].
lpr	print files [Berkeley only].
ls	display information about files.
mkdir	make a directory.
more	display a file one screenful at a time [Berkeley only].
mv	move (rename) a file.

od	dump a file.
pg	display a file one screenful at a time [System V only].
pr	paginate a file.
rm	remove a file.
rmdir	remove a directory.
sed	stream editor (noninteractive).
sort	sort and/or merge files.
spell	check a file for spelling errors.
tail	display the last part of a file.
tar	store or retrieve files from an archive file.
uniq	display lines of a file that are unique.
wc	display counts of lines, words, and characters.

Communication Commands/Utilities

mail	send or receive electronic mail [Berkeley only].
mailx	send or receive electronic mail [System V only].
mesg	enable/disable reception of messages.
msgs	display system-wide news [Berkeley only].
news	display system-wide news [System V only].
write	send a message to another user.
cd	change to another working directory.
chgrp	change the group that is associated with a file.
chmod	change the access mode of a file.
chown	change the owner of a file.
date	display or set the time and date.
df	display the amount of available disk space.
du	display information on disk usage.
file	display file classification.
kill	terminate a process.
newgrp	temporarily change the group identification of a user [System V only].
nice	change the priority of a command.
nohup	run a command that will keep running after we log out.
ps	display process status.
sleep	process that sleeps for a specified interval.
stty	display or set terminal parameters.
umask	set file-creation permissions mask.
who	display names of users.

Utilities That Are Programming Tools

cc	C compiler.
make	keep a set of programs current.
touch	updates a file's modification time.

Source Code Control System (SCCS) Utilities

admin	create an SCCS file or change the characteristics of one.
delta	record changes in an SCCS file.
get	retrieve an SCCS file.
prs	print the history of an SCCS file.
rmdel	remove a delta from an SCCS file.

Miscellaneous Commands/Utilities

at	execute a shell script at a specified time.
cal	display a calendar.
calendar	present reminders.
echo	display a message.
expr	evaluate an expression.
fsck	check and repair a file system.
shl	call the shell layer manager [System V only].
tee	copy the standard input to the standard output and to one or more files.
test	evaluate an expression.
tty	display the terminal pathname.

4.0 CONTROL STRUCTURES USED IN SHELL SCRIPTS

4.1 Classification of the Shell Script

Bourne Shell:	Suitable for Shell Programming; subset of Korn Shell.
C-Shell:	Suitable for interactive jobs; has special commands such as: history, alias, job control, etc.
Korn Shell:	Superset of Bourne Shell, plus some special features of C-Shell.

4.2 Control Structures of Bourne Shell and C-Shell

Bourne Shell

```
if [ EXPR. ]
    then
        COMMANDS
    else
        COMMANDS
fi
```

```
for LOOP-INDEX
do
    COMMANDS
done
```

```
for LOOP-INDEX in ARG-LIST
do
    COMMANDS
done
```

```
until [ EXPR. ]
do
    COMMANDS
done
```

```
while [ EXPR. ]
do
    COMMANDS
done
```

C-Shell

```
if ( EXPR. ) then
        COMMANDS
    else
        COMMANDS
endif
```

```
foreach LOOP-INDEX (ARG-LIST)
        COMMANDS
end
```

```
while ( EXPR. )
        COMMANDS
end
```

Bourne Shell

```
case TEST-STRING in
    PTN-1)
        CMDS-1
    ;;
    PTN-2)
        CMDS-2
    ;;
    . . .
    . . .
    *)
        CMDS-N
    ;;
esac
```

C-Shell

```
switch ( TEST-STRING )
    case PTN-1:
        COMMANDS
    breaksw
    case PTN-2:
        COMMANDS
    breaksw
. . .
. . .
    default:
        COMMANDS
    breaksw
endsw
```

Special Keywords Provided by Both Shells

break	out of the control loop
continue	back to the beginning of the loop
goto	jump to anywhere

Interrupt Handling

Bourne Shell

trap 'CMDS' SIGNAL-NUMBER

1:	hangup
2:	terminal interrupt
3:	exit
9:	kill
15:	software termination

C-Shell

```
onintr  LABEL

. . .
. . .
. . .
LABEL:
    COMMANDS
```

5.0 C-SHELL PROGRAMMING

We'll use the concepts, commands, utilities, etc., developed earlier as the building blocks to solve practical problems.

One problem we would like to solve is as follows:

Suppose we were given a data file called "datebook"; each record in the datebook has the layout —> name:phone number:birthdate:salary
for example ——> Peter Jennings:301-234-7700:10/26/39: 1267000

We would like to write a C-Shell Script to implement the following steps:

[1] Sort the data file "datebook" by last name.

[2] Greet the user.

[3] Tell the user "This program will print out pertinent infor. about people listed in the datebook file. Do you wish to see that information?" If the user types, "yes," include in your script at this point, the command:

```
awk –f  awklook   datebook
```

[4] Ask the user if he would like to add any entries to the datebook file. If he types "yes," prompt him for a new name, phone, birthdate, and salary.

[5] Check for duplicates and sort the datebook.

[6] Tell the user the new entry is in the datebook and the number of the record.

[7] Ask him if he would like to see the new entry.

The source code is listed on the next two pages, followed by the execution result.

```
#
#
#       Name:    Frank Nee              File Name:  lookup
#       This C-Shell Script is used to do database manipulation
#
sort        -f  datebook
echo
echo                "How are you,  $user"
echo
echo                "This program will print out pertinent information about people listed in the datebook file."
echo                "Do you wish to see that information\?"
echo
set         ans  =  $<
if          ( ( $ans  ==  "y" )  | |  ( $ans  ==  "Y" ))  then  #  print          pertinent information
            awk  -f  awklook datebook
endif
#
echo                "Would you like to add any entries to the datebook file\?"
echo
set         answer  =  $<
if          ( ( $answer  ==  "y" )  | |  ( $answer  ==  "Y" ) ) then  #  get info from user
            echo       "Please enter the name in the form —> last name, first name"
            set name  =  $<
            echo       "Please enter the phone number in the form —> XXX-YYY-ZZZZ"
            set phone  =  $<
            echo       "Please enter the birthdate in the form —> MM/DD/YY"
            set birthdate  =  $<
            echo       "Please enter the salary in the form —> NNNNN"
            set salary  =  $<
#
            echo       "$name" ' : ' $phone ' : ' $birthdate ' : ' $salary  >>  datebook
#
endif
```

```
#
sort        +0 -u datebook -o datebook
#
echo        "The new entry is in the datebook file"
#           Tell the user the entry number just entered and the total number in the file
#
echo
set         total = `awk 'END { print NR }' datebook`  #  prt total rec number

#
echo        "There are "  $total  " records in the datebook file"
set         num = `awk "/$name/ { print NR }"  datebook`  #  prt rec number
echo        "The entry you entered is:  "  $num  "th record in the datebook file"
#
#           display the entry that was just entered by the user
#
echo        "Would you like to see the new entry\?"
set         response = $<
if          ( ( $response == "y" ) || ( $response == "Y" ) ) then  #  prt entry info
            echo    "The new entry is:"
            echo    "Name = " "$name"              "Phone number = " "$phone"
            echo    "Birthdate = " "$birthdate"    "Salary = " "$salary"
endif
```

```
csh>          lookup
Chang, Bob:408-255-3276:7/1/57:50000
Class, George:415-345-8608:3/5/38:60000
Crane, Chuck:415-678-1234:6/6/59:45000
Daane, Jeff:415-340-2578:2/15/52:35000
Montana, Joe:408-252-6600:5/20/56:29000
Vinson, Carl:408-532-6789:9/28/50:28000
Wallace, Janet:415-996-3253:10/6/45:70000
Waugh, Vivian:408-638-5409:2/16/73:16000
How are you,  unf19
This program will print out pertinent information about people listed in the datebook file.
Do you wish to see that information\?
n
Would you like to add any entries to the datebook file\?
y
Please enter the name in the form —> last name, first name
Nee, Victor
Please enter the phone number in the form —> XXX-YYY-ZZZZ
408-252-6677
Please enter the birthdate in the form —> MM/DD/YY
7/1/57
Please enter the salary in the form —> NNNNN
66000
The new entry is in the datebook file
There are  9  records in the datebook file
The entry you entered is:  6 th record in the datebook file
Would you like to see the new entry\?
y
```

```
The new entry is:
Name = Nee, Victor     Phone number = 408-252-6677
Birthdate = 7/1/57     Salary = 66000
csh> cat datebook
Chang, Bob:408-255-3276:7/1/57:50000
Class, George:415-345-8608:3/5/38:60000
Crane, Chuck:415-678-1234:6/6/59:45000
Daane, Jeff:415-340-2578:2/15/52:35000
Montana, Joe:408-252-6600:5/20/56:29000
Nee, Victor:408-252-6677:7/1/57:66000
Vinson, Carl:408-532-6789:9/28/50:28000
Wallace, Janet:415-996-3253:10/6/45:70000
Waugh, Vivian:408-638-5409:2/16/73:16000
csh>
```

6.0 BOURNE SHELL PROGRAMMING

This time we'll use the same building blocks to solve problems in Bourne Shell Script. The technique is the same; the differences are in the syntax and conventions.

One problem we would like to solve is as follows:

Write a Bourne Shell Script called "checkon" to see if a user is logged on and to show what processes he is running. The script should be able to handle the following cases:

[1] If the user did not provide an argument to "checkon," the output should return:

```
Incorrect number of arguments
Usage:  checkon user-id
```

[2] If an invalid user-id was provided, the output should return:

```
user-id does not exist on this system
```

[3] If the user is not logged on, the script should output the following:

```
user-id exists on this system
but not logged on at this moment
```

[4] If successful, the output should resemble the following:

```
user-id is logged on and running these processes:
. . . . . .
listing of processes
```

The source code is listed on the next two pages, followed by the execution result.

```
:
#
#
#       Name: Frank Nee                 File Name: checkon
#
#
#

#       This Bourne Shell Script is used to check if a particular user is logged on
#       The following if-stmt will check the correct number of arguments

if [ "$#" -ne 1 ]
then
        echo    "Incorrect number of arguments.          "
        echo    "Usage: checkon user-id                  "
        exit    1
fi

name = "$1"
echo user-id is: $name

#       The following if-stmt will check if a particular user exists on the system

if      grep "$name"  /etc/passwd  >        /dev/null
then
        :                                               #       means do nothing
else
        echo "$name" does not exist on this system
        exit    0
fi
```

```
#       The following if-stmt will check if a particular user is logged on

if      who | grep "$name" > /dev/null
then
        echo "$name is logged on and running these processes:"
        ps -ua | grep "$name"
else
        echo "$name" exists on        this system
        echo but not logged on at       this moment
fi
```

```
csh>    sh checkon
Incorrect number of arguments.
Usage:    checkon  user-id
csh>    sh checkon stanford
user-id is: stanford
stanford does not exist on this system
csh>    sh checkon nobody
user-id is: nobody
nobody exists on this system
but not logged on at this moment
csh>    sh checkon unf19
user-id is: unf19
unf19 is logged on and running these processes:
unf19     3155    6.8    0.1     38    16    p1    S    0:00    sh checkon unf19
unf19     3179    3.0    0.1     21    13    p1    S    0:00    grep unf19
unf19     3178    1.0    0.7    299   189    p1    R    0:00    ps –ua
unf19     3078    0.3    0.1     37    14    p9    S    0:00    script nee–lab5
unf19     3080    0.1    0.2    130    34    p1    S    0:00    –h –i (csh)
unf19     3079    0.0    0.1     53    16    p9    S    0:00    script nee–lab5
unf19     2740    0.0    0.2    138    40    p9    I    0:01    –csh (csh)
csh>
```

7.0 BOOK REVIEW AND OPEN DISCUSSION

[1] The UNIX Programming Environment
Brian W. Kernighan and Rob Pike
Prentice Hall, Inc.
Definitive standard for UNIX operating system.

[2] The C Programming Language
Brian W. Kernighan and Dennis M. Ritchie
Prentice Hall, Inc.
Definitive standard for C Programming Language.

[3] The Design of the UNIX Operating System
Maurice J. Bach
Prentice Hall, Inc.
Describes the internal algorithms and structures that form the basis of the operating system (called the kernel) and their relationship to the programmer interface.

[4] UNIX System Programming
K. Haviland and B. Salama
Addison-Wesley
A book that describes low-level programming.

[5] Advanced UNIX—A Programmer's Guide
Stephen Prata
The Waite Group
Discussion of system calls, UNIX-C interface, etc.

[6] A Practical Guide to the UNIX System
Mark Sobell
The Benjamin/Cummings Publishing Company
A book that covers both C-Shell and Bourne Shell.