Lecture 12: Internal/ External Commands, Files, Utilities & Tools

INTERNAL AND EXTERNAL COMMANDS

There are two basic types of commands: the external and internal commands.

External Commands

An external command is a computer program. It is a compiled binary file. The shell searches the directories in the PATH variable to find external commands, or the full path is specified on the command line. Once the command has been located, the shell must spawn a child process for the program. Then the shell attempts to execute the program by making an exec system call. If successful, the program is loaded into memory and the UNIX operating system executes the program as a new process.

Internal Commands

An internal command is built-in to the shell code. It is a function or subroutine compiled as part of the shell program (**ksh**, **sh**, **csh**). The shell does NOT have to search the PATH variable for an internal command. No new process is spawned to execute the command. Thus no disk reading or program loading is required. The result is much faster execution.

So why not put all commands in the shell? There are multiple arguments concerning this idea. The conventional operating system concept actually leans more towards command processors and standard commands all being a part of the operating system. But UNIX tends to be more modular oriented. The modularity provides for more flexibility and better management. The more commands that are placed in the **ksh** program, the larger and less responsive it becomes. Thus the argument is fast response up to a break point versus management of programs.

The following table lists some of the less well-known internal commands.

exit n	Exit from the shell script with exit code n.
read	Read a line from the standard input. If followed by a list of variable names then the input line tokens are assigned to the variables.
readonly	When followed by a variable name marks the variable as read only.
set list	The tokens of the list are set to the positional parameters $1, $2 etc.
shift	The positional parameters are shifted "down". I.e. the value of $2 is assigned to $1 and so on.
times	Displays accumulated "u" and "s" times.
ulimit	Displays or sets various size limits.
umask	Sets the user file creation mask.
unset	When followed by a variable name list unsets the variable(s).

COMMAND EXECUTION

When you press Return to send a command to UNIX, the shell parses the command and performs the required substitutions and expansions. The following list outlines the sequence of substitutions and expansions in the order in which they are performed.

* Aliasing
* Tilde Substitution
* Command Substitution
* Variable Substitution
* Blank Interpretation
* File Name Generation
* Arithmetic Evaluation

Once the command has been expanded it is executed. If an internal or keyword command is used, the shell executes the command within the current environment (no subshell is spawned). User defined functions are included in the keyword commands. If I/O redirection is performed, a subshell is created and the commands are executed in the new environment.

If the command is an *external* type, then a process is always created and the shell attempts to execute the command via the **exec** system call. Internal and external commands are described in a following section of this module.

Searching for Commands

The shell interprets the first parameter on the command line ($0) and, based on the following criteria, locates the program to be executed.

If the command contains a / then the shell searches only the directory specified. The directories in the PATH variable are not searched for the command. If the command is not in the directory, the shell returns a "not found" message. For example,

```
cj> /usr/bin/lx
ksh:/usr/bin/lx: not found
cj> ../bin/lx
ksh:../bin/lx: not found
```

If you do not specify a directory path in the command name (no / is used), the shell searches each directory listed in the PATH variable for the command. If the command is located in one of the directories and is executable, then it is executed. If it is not found in one of the directories, the shell notifies you as follows:

```
cj> lx
ksh: lx: not found
```

Tracked Aliases

By using alias or set you can have the shell remember where a command is for later use. If a command has been tracked, the next time you attempt to execute the command the shell will not have to search the PATH directories for the command. This reduces overhead and response time in command searching.

cp

DESCRIPTION

The external **cp** command copies a file. It reads the contents of a file and creates a new file or overwrites an existing file. There are two basic formats of the **cp** command that allow you to:

* Copy one file to another
* Copy mutliple files to a directory
* Copy input from your keyboard to a file
    - Copy a file to your terminal

COMMAND FORMAT

Following is the general format of the **cp** command.

```
cp    [ -ip ]  source_file destination_file
cp    [ -ipr ] source_file_list destination_directory
cp -r [ -ip ]  source_directory destination_directory
```


Options

The following list describes the options that may be used to control how **cp** functions.

-i Interactive confirmation is required. You are prompted if the copy will overwrite an existing file. If your answer contains a **y**, the copy is performed. Otherwise, the copy is not performed.

-p Preserve the characteristics of the source_file. Copy the contents, modification times, and permission modes of the source_file to the destination files.

-r Recursively copy any source directories. If a directory is given as the source file, then all of its files and subdirectories are copied. The destination must be a directory.

Arguments

The following list describes the arguments that may be passed to the **cp** command.

source_file	The existing file that will be copied
source_file_list	The name of files and/or directories to be copied to a new destination directory
destination_file	The name of the file the new copy will be named
destination_directory	The name of the directory where a copy of the files will be made
source_directory	The name of the directory from where copies are read

APPLICATIONS

The **cp** command is used to make a copy of an existing file. It provides a way to create backup copies of files. If you accidently erase the current copy you are working on, you can fall back to the last backup copy you made.

You can copy a file to a temporary work area ($HOME/tmp is a good place) to perform changes. Then when you finish with the updates, copy the file back to its orginal location. You may want to write a shell script to perform these few tasks.

You may also want to use it to back up an entire work directory to a backup directory. Even though this uses up disk space, it may be well worth the cost. The trade off depends on the importance of the work you are performing versus the time it would take to recover or recreate the files if they are lost.

TYPICAL OPERATION

In this activity you use the **cp** command to copy a file. Begin at the shell prompt.

1. List your HOME directory by typing ls and pressing Return. Notice the file named *file1* in the following display.

```
cj> ls
```

bin calendar db file1 letters stuff

2. Copy *file1* to *stuff* by typing cp file1 stuff and pressing Return. This overwrites the contents already contained in the file named *stuff.*

3. Type cp calendar letters and press Return to copy the calendar file to the letters directory.

TOOLS AND UTILITIES

Tools and utilities are also commands. The tools' files also reside in *Ibin* or *lusrlbin* directories. A user can create his own private tools, which are executable object files in his own directory. Wide ranges of tools are available, and are increasing since one can write shell command programs and create new tools. Tools and utilities are used for file processing, editing, text preparation, document formatting, pattern matching, typesetting, program development, filtering, piping, redirection, communication, etc. Tools open an input file, processes it, and produce an output accordingly. Utilities ease the job of the user, particularly in efficient system programming and application development. In fact, shell programming is a built-in utility of UNIX. The tools and utilities could be classified as follows:

1. File processing tools: *wc, pipes, filters, redirection, various sort, Ip, pr*
2. Debugging tools: *debuggers, adb, sdb*
3. Editors and language development tools: *ed, vi, Sed, Lex, YACC*
4. System development and programming tools: *m4, make, SCCS, Prof*
5. Text processing tools: *grep, cut, paste, troff, nroff, eqn, pr*
6. Communication tools: *mail, write, mesg, wall*
7. Housekeeping utilities: *cal, date, pwd, who*
8. DOS enhancements: *dosmerge, dosdir, doscp*

All the tools and utilities are not essential and very necessary for basic computer use; however, they provide additional values and enhancements to UNIX.

UNIX Utilities and Applications

- General system operation
- File management
- Text processing
- System administration
- Networking and communication
- 'C Programming tools
- Source Code Control System
- Graphics

Over two hundred Utility programs (or commands) are supplied with the **UNIX** system which support a variety of tasks such as copying files, editing text, performing calculations and developing applications. User programs can be developed using these **UNIX** utilities.

UNIX File System

- Inverted tree (or Hierarchical) file structure
- Removable file system
- Dynamic file space allocation
- Structure less files
- File sharing
- File security

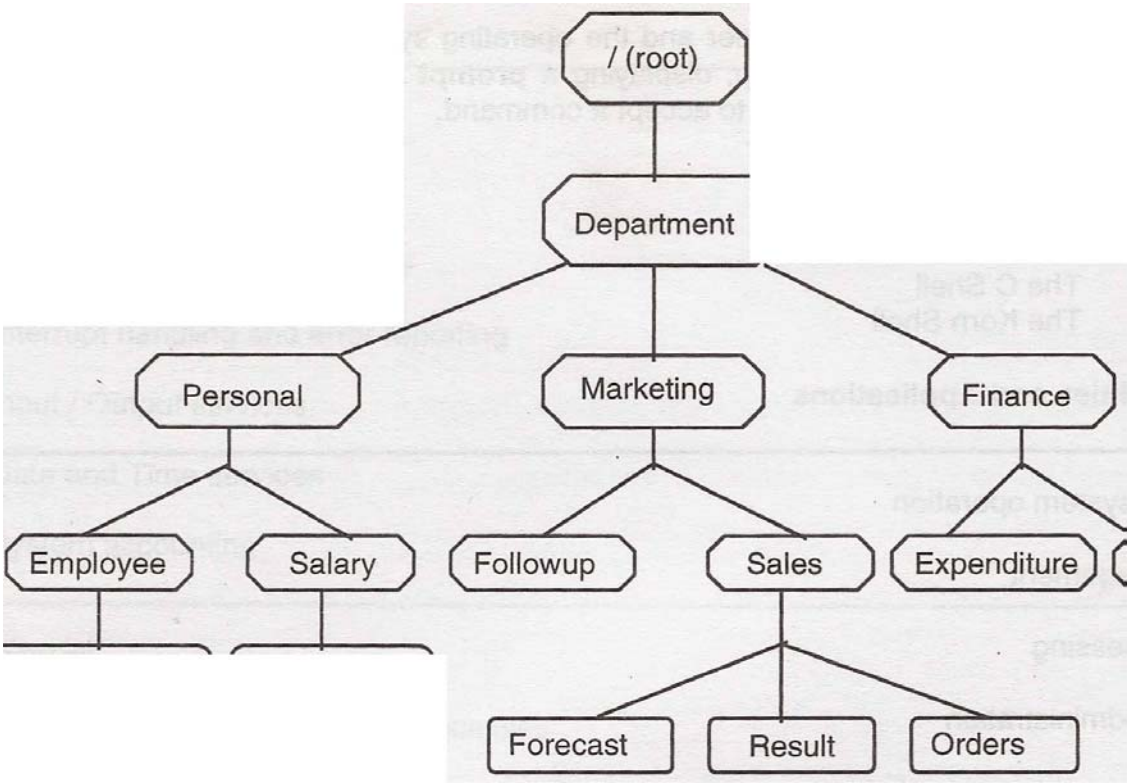

Fig: A Sample UNIX file system structure

Files are the fundamental structure to store information in the storage devices. The way a Fe ': created, accessed and modified, depends upon the operating system.

Many files can be stored on a disk. To organize these files, the **UNIX** operating system divides the disk into various logical units, where each unit can contain a group of related files.

These logical partitions are called **file systems.**

File systems are maintained on block devices such as disk or tapes.

An **Index Node** or **inode** number identifies every file in the file system.

Directories in the **UNIX** system perform the same function as a file drawer in a filing cabinet putting together related files in a common place where they can be manipulated easily and efficiently.

A directory is a way of organizing files by grouping them together. It is a special file, which contains a list of filenames present in that directory.

Hierarchical file Structure

The file system consists of all types of files and resembles an inverted tree structure.

The directories can have sub directories resulting in a large hierarchical structure.

A hierarchical file structure provides higher system performance, sheltering private data from unauthorized users and enhancing security against unintentional damage.

The main directory is at the top, called as the root, represented by "/" **(forward slash**). Note that the root directory of **DOS** is represented by a backslash,"\"
In the hierarchical structure shown in the figure on previous page, Department is a subdirectory of root. It has three sub directories, namely, Personal, Marketing and Finance. Personal in turn has two more sub directories namely, Employee and Salary.

Question: In UNIX, directories are also files. What you think will be the contents of directories?

UNIX DIRECTORY SYSTEM

Directories are central to the use of UNIX system, and the directory structure is of a great and unique strength to UNIX. The directory system of UNIX is in the form of an inverted tree. Refer

Fig. below which illustrates a UNIX directory system as an inverted tree.

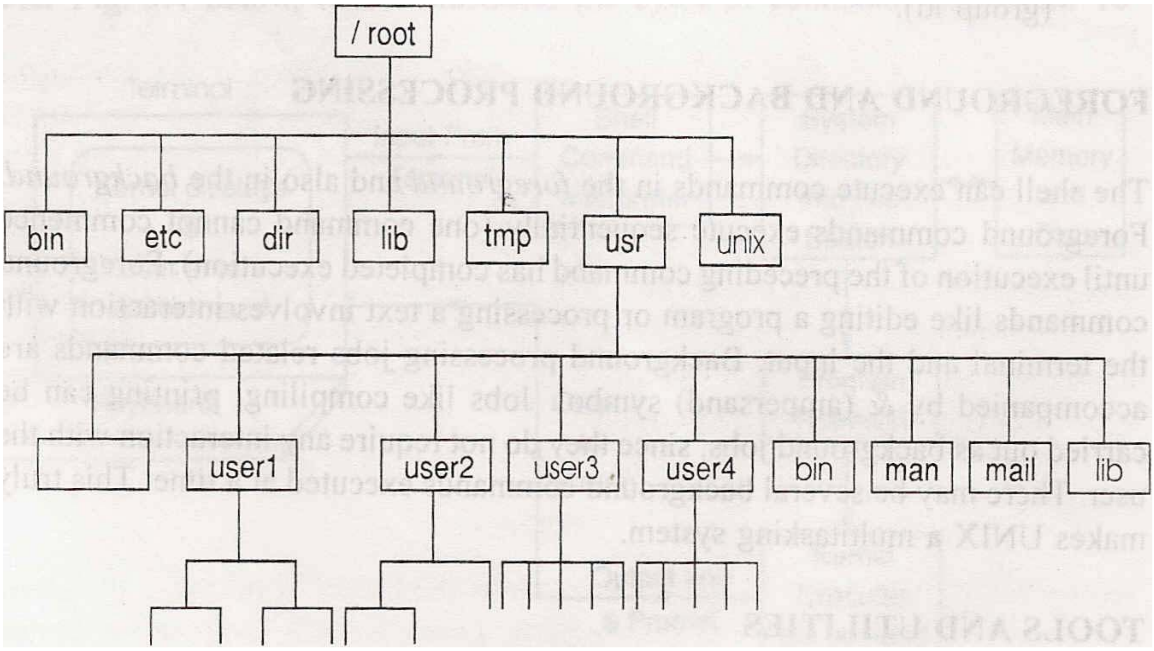

Fig: UNIX system directories

The directory at the topmost position is called *the root* since the structure appears as an inverted tree. It is the source from which all directories emerge. Under root, there are several system directories as the branches. The system directories are known as *bin, dev, lib, etc, temp, UNIX, usr.*

The *usr* directory has further system subdirectories. These are called as subdirectories of *usr* directory. Also under *usr,* directories for various users with their login name are located, with one directory allocated for each user. Whenever the user creates a directory for storing his files, they are stored in his login directory, called home directory.

Many users can create a number of subdirectories and files, which they can easily locate and use. For normal use, the user needs to be aware only of his own private universe of files and directories, and need not even know that any other user exists or is working in the system. The details of system directories and their content are shown in Table below and the structure in Fig.

below. Generally, there is no limit to the number of subdirectories one can create under his/her directory. They are restricted to the disc space allocation provided by the system administrator.

TABLE: Table of System Directories, the Files under them, and their Functions

/	root
/bin	Most commonly used UNIX commands and tools compilers, assemblers, editors, program development tools
/dev	Devices including terminals. Contains special files that represent peripheral devices such as the console, printer, floppy drives, tape discs, terminals, etc.
/etc	System Administrative files and commands contains system programs, data files of system administration, password files, login, etc.
/lib	Library functions for C program and languages, system calls, I/O routines
/tmp	Temporary files and work space for any user, using editors, compilers and assemblers
/unix	UNIX kernel
/usr	All user accounts, mail commands
/usr/bin	Less used utility program
/usr/man	Online user manual and information
/usr/mail	All the mails
/usr/lib	Library files
/usr/dict	Wordlists, spell checker
/user1 */user2*	Users' home directories

Question: Explain the term *home* directory in UNIX, where it is created and why?

ASSIGNMENT: Request the system administrator to enter a suitable command to display on the monitor, the subdirectories under root.

Review Questions:

- Differentiate between Internal and External commands used in UNIX.
- What are UNIX utilities?
- What type of structure does the UNIX file system have?
- Explain the UNIX file/directory structure.