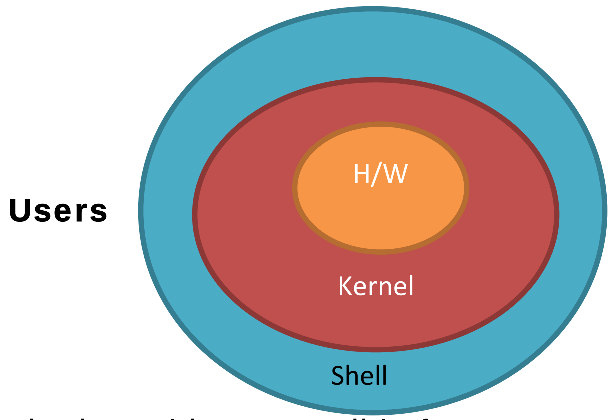
# **Unix shell and command line interface** (courtesy http://tinyurl.com/UnixShells)



The kernel is responsible for many complex tasks (translation, memory and file system management, etc). Its user interface, though much better than that of the hardware, still difficult to use for average user. Fortunately, UNIX implements another layer of abstraction that envelops the kernel. This next layer is called a shell.

The benefit of a shell, of course, is that it is built primarily for people. Time and energy have been taken to develop a user-friendly interface with a language more intelligible than either the language of the hardware or that of the kernel.

The developers of UNIX realised that several pieces which each do their one job well and know how to cooperate is a much better way to design software than to create one monolithic, all-knowing monster which is always breaking out of its cage and doing unpredictable and destructive things.

Thus, in UNIX, the hardware can focus solely on electronics, the kernel can focus on talking to the hardware and the shell can focus on talking to the user. Each piece has a limited task so it is able to that one task with efficiency and simplicity.

Of course, nothing is ever as simple as it is in theory. Though you will always use a shell when you are doing your work, "which" shell you will use will depend on the system you are working on. There are several basic command-line shells in use today and each has a slightly different interface and capabilities. Fortunately, they do share most of the same generic properties so it is fairly easy to switch between them.

In UNIX there are two major types of shells:

1. The Bourne shell. If you are using a Bourne-type shell, the default prompt is the $ character.

2. The C shell. If you are using a C-type shell, the default prompt is the % character.

There are again various subcategories for Bourne Shell which are listed as follows:

There are again various subcategories for Bourne Shell which are listed as follows:

- Bourne shell ( sh)
- Korn shell ( ksh)
- Bourne Again shell ( bash)
- POSIX shell ( sh)

The different C-type shells follow:

- C shell ( csh)
- TENEX/TOPS C shell ( tcsh)

The first UNIX shell was the Bourne Shell which is commonly abbreviated as "sh". Though this is the default shell which comes with every version of UNIX, it is the oldest and least shnazy of the shells since it has not been modified in several decades and has thus not kept up with interface advances.

The C Shell was created next at Berkeley and added several cool features like command line editing and history management. Unfortunately it was not particularly compatible with the Bourne Shell.

The Korn came next to take the good from both the C Shell and the Bourne Shell.

At any rate, by now there are quite a few standard shells as well as a host of proprietary ones. The following table lists the ones you will most often see

| Name | Abbreviation | Info |
|---|---|---|
| **Bourne Shell** | sh | The oldest of the shells which was designed by Steve Bourne. It is considered a bit primitive but very good for scripting. |
| **C Shell** | csh | The C Shell is probably the most popular. However, though it adds many nice features (like history and |

| | | |
|---|---|---|
| | | job control) unavailable in the Bourne Shell, it is quite buggy for heavy users. |
| **Korn Shell** | ksh | David Korn wrote this shell to be compatible with the Bourne Shell but include te cool features introduced by the C Shell. However, it did the C Shell one step further and introduced history editing. |
| **Bourne Again Shell** | bash | Similar to the Korn Shell but with some additional features like a built in help command. |
| **tcsh** | tsch | An extended version of the C Shell with the features introduced by ksh and bash. |

It is most likely that any UNIX system you use will have several or all of the shells installed so you can choose whichever one you are most comfortable with. The main thing to remember is that you should choose one that has the features which help you do your own special kind of work. Spend some time playing with each shell to see what they offer and how they differ.

## **Bash commands**

### **cal**

- Displays the current month
- It can also be used to display a variety of months and years.

cal month year (month is optional)

e.g., cal 2004 shows the full calendar for 2004

### **cat**

- Displays the contents of a file very easily.
- Derived from the term concatenate. The cat command concatenates the contents of a file on the disk with your screen.

cat option filename

**date**

- Displays or sets the system date and time.

date option +format

date +%D displays the date in MM/DD/YY

date +%d displays the two-digit day

date +%Y displays the four-digit year

date +%H displays the two-digit hour

date +%h displays the three-letter month

date +%T displays the time (hh:mm:ss)

date +%j displays the numeric day from 001 to 366; also called the Julian date

date +%m displays the two-digit month

**echo**

- Command used to display text on screen
- Very useful for debugging and troubleshooting shell scripts

echo “Text to display”

**history**

- Displays the list of commands you have run
- Without any options, it displays the last 1000 commands

history option

Allows you to enter a recently executed command by placing ! operator prior to the command. So, to rerun the most recently executed cal command enter:

```
!cal
```

History command displays numbers left to each command. You can run a command by its number.

```
!1031
```

**man**

- Allows you to display the manual, or help pages for a command. You run this command to see a description of a command and its options.

```
man command
```

**touch**

- Used to update the modification date and time stamp of a file.
- If a file does not exist, then touch can be used to create one.

```
touch filename
```

**who**

- Displays the list of users who are currently logged on to the system

```
who options
```

```
who –q displays a count of total logged in users
```

**Creating shell scripts**

Because shell is a command interpreter that makes use of programming capabilities, it allows you to use traditional programming concepts such as

- Make decisions based upon conditions

- Perform arithmetic operations
- Looping constructs
- Functions to perform specific tasks

A shell script performs one other function that traditional programming languages do not typically support; the shell script runs OS commands.

Create an execute a simple script

```
cat > script1.sh

echo “Welcome to my shell script”

echo “This is one of my first shell scripts”

echo “Shell script programming is a toll for helping users automate their tasks”

ctrl+D

bash script1.sh
```

```
cat script2.sh

cal

date

who

ctrl+D

bash script2.sh
```

**Linux System Directories**

/ Root directory. Normally, no files are placed here.

/bin Binary directory, where many GNU user-level utilities are stored

/boot Boot directory, where boot files are stored. Hold Linux kernel

/dev Contains device files for all the devices in the Linux file system

/etc Holds configuration files. These are the files that various programs read to know what they are supposed to do.

/home Default location which contains user directories; when user logs in to a Linux system, their current directory is /home/username.

/lib Contains files and executable programs used by the system

/mnt Contains mounted drives

/root The Linux superuser, or root's home directory

/sbin system binary directory, where many GNU admin-level utilities are stored for use by the root user

/tmp Used for temporary file and directory storage

/usr Contains other subdirectories for applications, such as X11 (for X Windows), HTML files, library files, and games. Two of the most widely used subdirectories within /usr are usr/bin and /usr/sbin. These contain directories and binary executables

/var Contains both files and directories; typically, the type of files in /var vary in size such as log files for various processes located in /var/log. System log file is stored at /var/log/messages.

## Navigating the Tree Hierarchy

User's home directory is represented by a variable called $HOME

Directories on the same level are called sibling directories.

A child directory is one that is contained inside another directory called parent directory.

## Changing directory locations

cd directory

Full path specifies the complete path from root. It always begins from the root directory.

Partial path specifies a certain point in the directory path, which is simply the name of the directory path, which is simply the name of the directory that you would like to change to relative to your current location. It never begins at root.

cd / changes to root directory

cd .. changes to the parent directory (one level up)

cd ../sibling directory changes to a sibling directory via the parent directory

cd ../../directory changes directory locations across multiple locations (in this case two level up)

## Directory commands

pwd (print working directory) prints the name of the current directory. Lets you know where in the tree hierarchy you are.

ls displays both files and directories in the current directory. When using ls –l option (called long listing) shows more information such as type of file, permissions, links, owner, group, size in bytes, date and time, and name

*Possible types of files:*

| | |
|---|---|
| - | A normal file |
| d | A directory |
| b | A block device, one which communicates by sending entire blocks of data (e.g., HD, USB drive) |
| s | A socket connection for communication |
| c | Character device, one in which the driver communicates by sending and receiving single characters (bytes or octets) (e.g., serial ports, parallel ports, sound card, etc.) |

*Permission column:*

r Indicates that the file or directory can be read w Indicates that the file or directory can be written to

x Indicates that the file can be executed; shell scripts may need the x permission. An x permission for a directory indicates that you can take a directory listing of the directory to see its subdirectories and files

-Indicates the permission is not granted

*Link counts:*

Shows the number of links or shortcut a file has. A file has 1 count and directory has 2 counts to start with.

ls –a option shows hidden files as well (hidden files start with a dot)

When you type ls –a you will notice a dot (.) and dot dot (..). The single dot represents current directory. The two dots represents the parent directory. Every directory has these two directories. That is why cd .. works.

ls –al shows the long listing with the hidden files

mkdir creates or makes a directory

rmdir deletes or removes a directory (only deletes empty directory)

**Understanding File Management Commands**

cp Creates a copy of an existing file

cp option source destination

if you use the –i option it will prompt the user before copy the file if the destination exists (to avoid overwriting)

You can use cp with both the full path and partial path

mv allows you to move or rename a file or directory.

mv source destination (again can be used with –i option for interactive mode)

mv source destination in same directory causes renaming of the file

The only time the source and destination can have the same name is when you are moving it to another directory. You cannot rename a file to its same name.

sort is used to sort data in ascending or descending order.

sort filename

sort –r filename (to sort in descending order or reverse order)

cut allows you to strip text out of files and display the cut text on the screen or redirect to another file. It does not alter the original file.

cut options filename

The –d option is used to identify the delimiter of the file. The –f option is used to identify which field you want based on the delimiter.

1001,King,Dog

First field is 1001, second field is King, and third field is Dog.

cut –d, -f1 filename would display the first field

cut –d –f1,3 filename would display the first and third field

paste command pastes or merges data from one file to another.

paste filename_one filename_two

rm command is used to delete files permanently.

rm filename

rm –f to forcibly delete a file

rm –r to recursively delete files in a directory

rm –fr to recursively delete files in a directory without prompting

uniq used to find duplicate lines from a sorted file.

uniq filename

uniq –c filename tells the number of duplicates