UNIX BASICS

Unix - File Management

All data in UNIX is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem.When you work with UNIX, one way or another you spend most of your time working with files. This tutorial would teach you how to create and remove files, copy and rename them, create links to them etc.

In UNIX there are three basic types of files:

1. **Ordinary Files:** An ordinary file is a file on the system that contains data, text, or program instructions. In this tutorial, you look at working with ordinary files.
2. **Directories:** Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders.
3. **Special Files:** Some special files provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Other special files are similar to aliases or shortcuts and enable you to access a single file using different names.

Listing Files:

To list the files and directories stored in the current directory. Use the following command:[amrood]$ls

Here is the sample output of the above command:

```
[amrood]$ls

bin        hosts  lib     res.03
ch07       hw1    pub     test_results
ch07.bak   hw2    res.01  users
docs       hw3    res.02  work
```

The command **ls** supports the **-1** option which would help you to get more information about the listed files:

```
[amrood]$ls -l
total 1962188

drwxrwxr-x  2 amrood amrood      4096 Dec 25 09:59 uml
-rw-rw-r--  1 amrood amrood      5341 Dec 25 08:38 uml.jpg
drwxr-xr-x  2 amrood amrood      4096 Feb 15  2006 univ
drwxr-xr-x  2 root   root        4096 Dec  9  2007 urlspedia
```

```
-rw-r--r--  1 root   root      276480 Dec  9  2007 urlspedia.tar
drwxr-xr-x  8 root   root        4096 Nov 25  2007 usr
drwxr-xr-x  2    200    300      4096 Nov 25  2007 webthumb-1.01
-rwxr-xr-x  1 root   root        3192 Nov 25  2007 webthumb.php
-rw-rw-r--  1 amrood amrood     20480 Nov 25  2007 webthumb.tar
-rw-rw-r--  1 amrood amrood      5654 Aug  9  2007 yourfile.mid
-rw-rw-r--  1 amrood amrood    166255 Aug  9  2007 yourfile.swf
drwxr-xr-x 11 amrood amrood      4096 May 29  2007 zlib-1.2.3
[amrood]$
```

Here is the information about all the listed columns:

1. First Column: represents file type and premission given on the file. Below is the description of all type of files.
2. Second Column: represents the number of memory blocks taken by the file or directory.
3. Third Column: represents owner of the file. This is the Unix user who created this file.
4. Fourth Column: represents group of the owner. Every Unux user would have an associated group.
5. Fifth Column: represents file size in bytes.
6. Sixth Column: represents date and time when this file was created or modified last time.
7. Seventh Column: represents file or directory name.

In the ls -l listing example, every file line began with a d, -, or l. These characters indicate the type of file that's listed.

Prefix	**Description**
-	Regular file, such as an ASCII text file, binary executable, or hard link.
b	Block special file. Block input/output device file such as a physical hard drive.
c	Character special file. Raw input/output device file such as a physical hard drive
d	Directory file that contains a listing of other files and directories.
l	Symbolic link file. Links on any regular file.
p	Named pipe. A mechanism for interprocess communications
s	Socket used for interprocess communication.

Meta Characters:

Meta characters have special meaning in Unix. For example ***** and **?** are metacharacters. We use ***** to match 0 or more characters, a question mark **?** matches with single character.

For Example:

```
[amrood]$ls ch*.doc
```

Displays all the files whose name start with ch and ends with .doc:

```
ch01-1.doc   ch010.doc  ch02.doc    ch03-2.doc
ch04-1.doc   ch040.doc  ch05.doc    ch06-2.doc
ch01-2.doc ch02-1.doc c
```

Here ***** works as meta character which matches with any character. If you want to display all the files ending with just **.doc** then you can use following command:

```
[amrood]$ls *.doc
```

Hidden Files:

An invisible file is one whose first character is the dot or period character (.). UNIX programs (including the shell) use most of these files to store configuration information.

Some common examples of hidden files include the files:

- **.profile:** the Bourne shell (sh) initialization script
- **.kshrc:** the Korn shell (ksh) initialization script
- **.cshrc:** the C shell (csh) initialization script
- **.rhosts:** the remote shell configuration file

To list invisible files, specify the -a option to ls:

```
[amrood]$ ls -a


.         .profile       docs     lib     test_results
..        .rhosts        hosts    pub     users
.emacs    bin            hw1      res.01  work
.exrc     ch07           hw2      res.02
.kshrc    ch07.bak       hw3      res.03
[amrood]$
```

- Single dot **.**: This represents current directory.
- Double dot **..**: This represents parent directory.

Note: I have put stars (*) just to show you the location where you would need to enter the current and new passwords otherwise at your system, it would not show you any character when you would type.

Creating Files:

You can use **vi** editor to create ordinary files on any Unix system. You simply need to give following command:

```
[amrood]$ vi filename
```

Above command would open a file with the given filename. You would need to press key **i** to come into edit mode. Once you are in edit mode you can start writing your content in the file as below:

```
This is unix file....I created it for the first time.....
I'm going to save this content in this file.
```

Once you are done, do the following steps:

- Press key **esc** to come out of edit mode.
- Press two keys **Shift + ZZ** together to come out of the file completely.

Now you would have a file created with **filemame** in the current directory.

```
[amrood]$ vi filename
[amrood]$
```

Editing Files:

You can edit an existing file using **vi** editor. We would cover this in detail in a separate tutorial. But in short, you can open existing file as follows:

```
[amrood]$ vi filename
```

Once file is opened, you can come in edit mode by pressing key **i** and then you can edit file as you like. If you want to move here and there inside a file then first you need to come out of edit mode by pressing key **esc** and then you can use following keys to move inside a file:

- **l** key to move to the right side.
- **h** key to move to the left side.
- **k** key to move up side in the file.
- **j** key to move down side in the file.

So using above keys you can position your cursor where ever you want to edit. Once you are positioned then you can use **i** key to come in edit mode. Edit the file, once you are done press **esc** and finally two keys **Shift + ZZ** together to come out of the file completely.

Display Content of a File:

You can use **cat** command to see the content of a file. Following is the simple example to see the content of above created file:

```
[amrood]$ cat filename
This is unix file....I created it for the first time.....
I'm going to save this content in this file.
[amrood]$
```

You can display line numbers by using **-b** option along with **cat** command as follows:

```
[amrood]$ cat filename –b
1   This is unix file....I created it for the first time.....
2   I'm going to save this content in this file.
[amrood]$
```

Counting Words in a File:

You can use the **wc** command to get a count of the total number of lines, words, and characters contained in a file. Following is the simple example to see the information about above created file:

```
[amrood]$ wc filename
2  19 103 filename
[amrood]$
```

Here is the detail of all the four columns:

1. First Column: represents total number of lines in the file.
2. Second Column: represents total number of words in the file.
3. Third Column: represents total number of bytes in the file. This is actual size of the file.
4. Fourth Column: represents file name.

You can give multiple files at a time to get the information about those file. Here is simple syntax:

```
[amrood]$ wc filename1 filename2 filename3
```

Copying Files:

To make a copy of a file use the **cp** command. The basic syntax of the command is:

```
[amrood]$ cp source_file destination_file
```

Following is the example to create a copy of existing file **filename**.

```
[amrood]$ cp filename copyfile
[amrood]$
```

Now you would find one more file **copyfile** in your current directory. This file would be exactly same as original file **filename**.

Renaming Files:

To change the name of a file use the **mv** command. Its basic syntax is:

```
[amrood]$ mv old_file new_file
```

Following is the example which would rename existing file **filename** to **newfile**:

```
[amrood]$ mv filename newfile
[amrood]$
```

The **mv** command would move existing file completely into new file. So in this case you would fine only **newfile** in your current directory.

Deleting Files:

To delete an existing file use the **rm** command. Its basic syntax is:

```
[amrood]$ rm filename
```

Caution: It may be dangerous to delete a file because it may contain useful information. So be careful while using this command. It is recommended to use **-i** option along with **rm** command.
Following is the example which would completely remove existing file **filename**:

```
[amrood]$ rm filename
[amrood]$
```

You can remove multiple files at a tile as follows:

```
[amrood]$ rm filename1 filename2 filename3
[amrood]$
```

Standard Unix Streams:Under normal circumstances every Unix program has three streams (files) opened for it when it starts up:

1. **stdin :** This is referred to as *standard input* and associated file descriptor is 0. This is also represented as STDIN. Unix program would read default input from STDIN.
2. **stdout :** This is referred to as *standard output* and associated file descriptor is 1. This is also represented as STDOUT. Unix program would write default output at STDOUT
3. **stderr :** This is referred to as *standard error* and associated file descriptor is 2. This is also represented as STDERR. Unix program would write all the error message at STDERR.

Unix - File System Basics

A file system is a logical collection of files on a partition or disk. A partition is a container for information and can span an entire hard drive if desired.

Your hard drive can have various partitions which usually contains only one file system, such as one file system housing the / file system or another containing the /home file system.

One file system per partition allows for the logical maintenance and management of differing file systems.

Everything in Unix is considered to be a file, including physical devices such as DVD-ROMs, USB devices, floppy drives, and so forth.

Directory Structure:

Unix uses a hierarchical file system structure, much like an upside-down tree, with root (/) at the base of the file system and all other directories spreading from there.A UNIX filesystem is a collection of files and directories that has the following properties:

- It has a root directory (/) that contains other files and directories.
- Each file or directory is uniquely identified by its name, the directory in which it resides, and a unique identifier, typically called an inode.
- By convention, the root directory has an inode number of 2 and the lost+found directory has an inode number of 3. Inode numbers 0 and 1 are not used. File inode numbers can be seen by specifying the -i option to ls command.
- It is self contained. There are no dependencies between one filesystem and any other.

The directories have specific purposes and generally hold the same types of information for easily locating files. Following are the directories that exist on the major versions of Unix:

Directory	Description
/	This is the root directory which should contain only the directories needed at the top level of the file structure.
/bin	This is where the executable files are located. They are available to all user.
/dev	These are device drivers.
/etc	Supervisor directory commands, configuration files, disk configuration files, valid user lists, groups, ethernet, hosts, where to send critical messages.
/lib	Contains shared library files and sometimes other kernel-related files.
/boot	Contains files for booting the system.

/home	Contains the home directory for users and other accounts.
/mnt	Used to mount other temporary file systems, such as cdrom and floppy for the CD-ROM drive and floppy diskette drive, respectively
/proc	Contains all processes marked as a file by process number or other information that is dynamic to the system.
/tmp	Holds temporary files used between system boots
/usr	Used for miscellaneous purposes, or can be used by many users. Includes administrative commands, shared files, library files, and others
/var	Typically contains variable-length files such as log and print files and any other type of file that may contain a variable amount of data
/sbin	Contains binary (executable) files, usually for system administration. For example *fdisk* and *ifconfig* utlities.
/kernel	Contains kernel files

Navigating the File System:

Now that you understand the basics of the file system, you can begin navigating to the files you need. The following are commands you'll use to navigate the system:

Command	**Description**
cat filename	Displays a filename.
cd dirname	Moves you to the directory identified.
cp file1 file2	Copies one file/directory to specified location.
file filename	Identifies the file type (binary, text, etc).
find filename dir	Finds a file/directory.
head filename	Shows the beginning of a file.
less filename	Browses through a file from end or beginning.
ls dirname	Shows the contents of the directory specified.
mkdir dirname	Creates the specified directory.

more filename	Browses through a file from beginning to end.
mv file1 file2	Moves the location of or renames a file/directory.
pwd	Shows the current directory the user is in.
rm filename	Removes a file.
rmdir dirname	Removes a directory.
tail filename	Shows the end of a file.
touch filename	Creates a blank file or modifies an existing file.s attributes.
whereis filename	Shows the location of a file.
which filename	Shows the location of a file if it is in your PATH.

You can use Manpage Help to check complete syntax for each command mentioned here.

The df Command:

The first way to manage your partition space is with the df (disk free) command. The command df -k (disk free) displays the disk space usage in kilobytes, as shown below:

```
[amrood]$df -k
Filesystem      1K-blocks      Used   Available Use% Mounted on
/dev/vzfs       10485760   7836644     2649116  75% /
/devices               0         0           0   0% /devices
[amrood]$
```

Some of the directories, such as /devices, shows 0 in the kbytes, used, and avail columns as well as 0% for capacity. These are special (or virtual) file systems, and although they reside on the disk under /, by themselves they do not take up disk space.

The df -k output is generally the same on all Unix systems. Here's what it usually includes:

Column	Description
Filesystem	The physical file system name.
kbytes	Total kilobytes of space available on the storage medium.
used	Total kilobytes of space used (by files).

avail	Total kilobytes available for use.
capacity	Percentage of total space used by files.
Mounted on	What the file system is mounted on.

You can use the -h (human readable) option to display the output in a format that shows the size in easier-to-understand notation.

The du Command:

The du (disk usage) command enables you to specify directories to show disk space usage on a particular directory.

This command is helpful if you want to determine how much space a particular directory is taking. Following command would display number of blocks consumed by each directory. A single block may take either 512 Bytes or 1 Kilo Byte depending on your system.

```
[amrood]$du /etc
10     /etc/cron.d
126    /etc/default
6      /etc/dfs
...
[amrood]$
```

The -h option makes the output easier to comprehend:

```
[amrood]$du -h /etc
5k    /etc/cron.d
63k   /etc/default
3k    /etc/dfs
...
[amrood]$
```

Mounting the File System:

A file system must be mounted in order to be usable by the system. To see what is currently mounted (available for use) on your system, use this command:

```
[amrood]$ mount
/dev/vzfs on / type reiserfs (rw,usrquota,grpquota)
proc on /proc type proc (rw,nodiratime)
devpts on /dev/pts type devpts (rw)
```

```
[amrood]$
```

The /mnt directory, by Unix convention, is where temporary mounts (such as CD-ROM drives, remote network drives, and floppy drives) are located. If you need to mount a file system, you can use the mount command with the following syntax:

```
mount -t file_system_type device_to_mount directory_to_mount_to
```

For example, if you want to mount a CD-ROM to the directory /mnt/cdrom, for example, you can type:

```
[amrood]$ mount -t iso9660 /dev/cdrom /mnt/cdrom
```

This assumes that your CD-ROM device is called /dev/cdrom and that you want to mount it to /mnt/cdrom. Refer to the mount man page for more specific information or type mount -h at the command line for help information.

After mounting, you can use the cd command to navigate the newly available file system through the mountpoint you just made.

Unmounting the File System:

To unmount (remove) the file system from your system, use the **umount** command by identifying the mountpoint or device

For example, to unmount cdrom, use the following command:

```
[amrood]$ umount /dev/cdrom
```

The mount command enables you to access your file systems, but on most modern Unix systems, the automount function makes this process invisible to the user and requires no intervention.

User and Group Quotas:

User and group quotas provide the mechanisms by which the amount of space used by a single user or all users within a specific group can be limited to a value defined by the administrator.

Quotas operate around two limits that allow the user to take some action if the amount of space or number of disk blocks start to exceed the administrator defined limits:

- **Soft Limit:** If the user exceeds the limit defined, there is a grace period that allows the user to free up some space.
- **Hard Limit:** When the hard limit is reached, regardless of the grace period, no further files or blocks can be allocated.

There are a number of commands to administer quotas:

Command	**Description**
quota	Displays disk usage and limits for a user of group.
edquota	This is a quota editor. Users or Groups quota can be edited using this command.
quotacheck	Scan a filesystem for disk usage, create, check and repair quota files
setquota	This is also a command line quota editor.
quotaon	This announces to the system that disk quotas should be enabled on one or more filesystems.
quotaoff	This announces to the system that disk quotas should be disabled off one or more filesystems.
repquota	This prints a summary of the disc usage and quotas for the specified file systems

Unix - Regular Expressions with SED

A regular expression is a string that can be used to describe several sequences of characters. Regular expressions are used by several different Unix commands, including **ed, sed, awk, grep,** and, to a more limited extent, **vi**.

This tutorial would teach you how to use regular expression along with **sed**.

Here sed stands for **s**tream **ed**itor is a stream oriented editor which was created exclusively for executing scripts. Thus all the input you feed into it passes through and goes to STDOUT and it does not change the input file.

Invoking sed:

Before we start, let us take make sure you have a local copy of /etc/passwd text file to work with **sed**.

As mentioned previously, sed can be invoked by sending data through a pipe to it as follows:

```
[amrood]$ cat /etc/passwd | sed
Usage: sed [OPTION]... {script-other-script} [input-file]...
  -n, --quiet, --silent          suppress automatic printing of pattern space
  -e script, --expression=script
...............................
```

The cat command dumps the contents of /etc/passwd to sed through the pipe into sed's pattern space. The pattern space is the internal work buffer that sed uses to do its work.

The sed Genral Syntax:

Following is the general syntax for sed

```
/pattern/action
```

Here, **pattern** is a regular expression, and **action** is one of the commands given in the following table. If **pattern** is omitted, **action** is performed for every line as we have seen above.

The slash characters (/) that surround the pattern are required because they are used as delimiters.

Range	**Description**
p	Prints the line
d	Deletes the line
s/pattern1/pattern2/	Substitutes the first occurrence of pattern1 with pattern2.

Deleting All Lines with sed:

Invoke sed again, but this time tell sed to use the editing command delete line, denoted by the single letter d:

```
[amrood]$ cat /etc/passwd | sed 'd'
[amrood]$
```

Instead of invoking sed by sending a file to it through a pipe, you can instruct sed to read the data from a file, as in the following example.

The following command does exactly the same thing as the previous Try It Out, without the cat command:

```
[amrood]$ sed -e 'd' /etc/passwd
[amrood]$
```

The sed Addresses:

Sed also understands something called addresses. Addresses are either particular locations in a file or a range where a particular editing command should be applied. When sed encounters no addresses, it performs its operations on every line in the file.

The following command adds a basic address to the sed command you've been using:

```
[amrood]$ cat /etc/passwd | sed '1d' |more
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
[amrood]$
```

Notice that the number 1 is added before the delete edit command. This tells sed to perform the editing command on the first line of the file. In this example, sed will delete the first line of /etc/password and print the rest of the file.

The sed Address Ranges:

So what if you want to remove more than one line from a file? You can specify an address range with sed as follows:

```
[amrood]$ cat /etc/passwd | sed '1, 5d' |more
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
[amrood]$
```

Above command would be applied on all the lines starting from 1 through 5. So it deleted first five lines.

Try out the following address ranges:

Range	Description
'4,10d'	Lines starting from 4th till 10th are deleted
'10,4d'	Only 10th line is deleted, because sed does not work in reverse direction.
'4,+5d'	This will match line 4 in the file, delete that line, continue to delete the next five lines, and then cease its deletion and print the rest
'2,5!d'	This will deleted everything except starting from 2nd till 5th line.
'1~3d'	This deletes the first line, steps over the next three lines, and then deletes the fourth line. Sed continues applying this pattern until the end of the file.
'2~2d'	This tells sed to delete the second line, step over the next line, delete the next line, and repeat until the end of the file is reached.
'4,10p'	Lines starting from 4th till 10th are printed
'4,d'	This would generate syntax error.
',10d'	This would also generate syntax error.

Note: While using **p** action, you should use **-n** option to avoid repetition of line printing. Check the difference in betweek following two commands:

```
[amrood]$ cat /etc/passwd | sed -n '1,3p'
```

Check the above command without **-n** as follows:

```
[amrood]$ cat /etc/passwd | sed '1,3p'
```

The Substitution Command:

The substitution command, denoted by **s**, will substitute any string that you specify with any other string that you specify.

To substitute one string with another, you need to have some way of telling sed where your first string ends and the substitution string begins. This is traditionally done by bookending the two strings with the forward slash (/) character.

The following command substitutes the first occurrence on a line of the string **root** with the string **amrood**.

```
[amrood]$ cat /etc/passwd | sed 's/root/amrood/'
amrood:x:0:0:root user:/root:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
..........................
```

It is very important to note that sed substitutes only the first occurrence on a line. If the string root occurs more than once on a line only the first match will be replaced.
To tell sed to do a global substitution, add the letter **g** to the end of the command as follows:

```
[amrood]$ cat /etc/passwd | sed 's/root/amrood/g'
amrood:x:0:0:amrood user:/amrood:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
...........................
```

Substitution Flags:

There are a number of other useful flags that can be passed in addition to the g flag, and you can specify more than one at a time.

Flag	**Description**
g	Replace all matches, not just the first match.
NUMBER	Replace only NUMBERth match.
p	If substitution was made, print pattern space.
w FILENAME	If substitution was made, write result to FILENAME.
I or i	Match in a case-insensitive manner.
M or m	In addition to the normal behavior of the special regular expression characters ^ and $, this flag causes ^ to match the empty string after a newline and $ to match the empty string before a newline.

Using an Alternative String Separator:

You may find yourself having to do a substitution on a string that includes the forward slash character. In this case, you can specify a different separator by providing the designated character after the s.

```
[amrood]$ cat /etc/passwd | sed 's:/root:/amrood:g'
amrood:x:0:0:amrood user:/amrood:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

In the above example we have used **:** as delimeter instead of slash / because we were trying to search /root instead of simple root.

Replacing with Empty Space:

Use an empty substitution string to delete the root string from the /etc/passwd file entirely:

```
[amrood]$ cat /etc/passwd | sed 's/root//g'
:x:0:0::/:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

Address Substitution:

If you want to substitute the string sh with the string quiet only on line 10, you can specify it as follows:

```
[amrood]$ cat /etc/passwd | sed '10s/sh/quiet/g'
root:x:0:0:root user:/root:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/quiet
```

Similarly, to do an address range substitution, you could do something like the following:

```
[amrood]$ cat /etc/passwd | sed '1,5s/sh/quiet/g'
root:x:0:0:root user:/root:/bin/quiet
daemon:x:1:1:daemon:/usr/sbin:/bin/quiet
bin:x:2:2:bin:/bin:/bin/quiet
sys:x:3:3:sys:/dev:/bin/quiet
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
```

```
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

As you can see from the output, the first five lines had the string sh changed to quiet, but the rest of the lines were left untouched.

The Matching Command:

You would use **p** option along with **-n** option to print all the matching lines as follows:

```
[amrood]$ cat testing | sed -n '/root/p'
root:x:0:0:root user:/root:/bin/sh
[root@ip-72-167-112-17 amrood]# vi testing
root:x:0:0:root user:/root:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

Using Regular Expression:

While matching pattern, you can use regular expression which provides more flexibility.
Check following example which matches all the lines starting with *daemon* and then deleting them:

```
[amrood]$ cat testing | sed '/^daemon/d'
root:x:0:0:root user:/root:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
```

```
news:x:9:9:news:/var/spool/news:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

Following is the example which would delete all the lines ending with **sh**:

```
[amrood]$ cat testing | sed '/sh$/d'
sync:x:4:65534:sync:/bin:/bin/sync
```

The following table lists four special characters that are very useful in regular expressions.

Character	**Description**
^	Matches the beginning of lines.
$	Matches the end of lines.
.	Matches any single character.
*	Matches zero or more occurrences of the previous character
[chars]	Matches any one of the characters given in chars, where chars is a sequence of characters. You can use the - character to indicate a range of characters.

Matching Characters:

Look at a few more expressions to demonstrate the use of the metacharacters. For example, the following pattern:

Expression	**Description**
/a.c/	Matches lines that contain strings such as a+c, a-c, abc, match, and a3c, whereas the pattern
/a*c/	Matches the same strings along with strings such as ace, yacc, and arctic.
/[tT]he/	Matches the string The and the:
/^$/	Matches Blank lines
/^.*$/	Matches an entire line whatever it is.
/ */	Matches one or more spaces
/^$/	Matches Blank lines

Following table shows some frequently used sets of characters:

Set	Description
[a-z]	Matches a single lowercase letter
[A-Z]	Matches a single uppercase letter
[a-zA-Z]	Matches a single letter
[0-9]	Matches a single number
[a-zA-Z0-9]	Matches a single letter or number

Character Class Keywords:

Some special keywords are commonly available to regexps, especially GNU utilities that employ regexps. These are very useful for sed regular expressions as they simplify things and enhance readability.

For example, the characters a through z as well as the characters A through Z constitute one such class of characters that has the keyword [[:alpha:]]

Using the alphabet character class keyword, this command prints only those lines in the /etc/syslog.conf file that start with a letter of the alphabet:

```
[amrood]$ cat /etc/syslog.conf | sed -n '/^[[:alpha:]]/p'
authpriv.*                /var/log/secure
mail.*                   -/var/log/maillog
cron.*                   /var/log/cron
uucp,news.crit              /var/log/spooler
local7.*                  /var/log/boot.log
```

The following table is a complete list of the available character class keywords in GNU sed.

Character Class	Description
[[:alnum:]]	Alphanumeric [a-z A-Z 0-9]
[[:alpha:]]	Alphabetic [a-z A-Z]
[[:blank:]]	Blank characters (spaces or tabs)
[[:cntrl:]]	Control characters
[[:digit:]]	Numbers [0-9]
[[:graph:]]	Any visible characters (excludes whitespace)

[[:lower:]]	Lowercase letters [a-z]
[[:print:]]	Printable characters (noncontrol characters)
[[:punct:]]	Punctuation characters
[[:space:]]	Whitespace
[[:upper:]]	Uppercase letters [A-Z]
[[:xdigit:]]	Hex digits [0-9 a-f A-F]

Aampersand Referencing:

The sed metacharacter & represents the contents of the pattern that was matched. For instance, say you have a file called phone.txt full of phone numbers, such as the following:

```
5555551212
5555551213
5555551214
6665551215
6665551216
7775551217
```

You want to make the area code (the first three digits) surrounded by parentheses for easier reading. To do this, you can use the ampersand replacement character, like so:

```
$ sed -e 's/^[[:digit:]][[:digit:]][[:digit:]]/(&)/g' phone.txt
(555)5551212
(555)5551213
(555)5551214
(666)5551215
(666)5551216
(777)5551217
```

Here in pattern part you are matching first 3 digits and then using & you are replacing those 3 digits with surrounding parentheses.

Using Multiple sed Commands:

You can use multiple sed commands in a single sed command as follows:

```
$ sed -e 'command1' -e 'command2' ... -e 'commandN' files
```

Here command1 through commandN are sed commands of the type discussed previously. These commands are applied to each of the lines in the list of files given by files.

Using the same mechanism, we can write above phone number example as follows:

```
$ sed -e 's/^[[:digit:]]\{3\}/(&)/g'  \
                    -e 's/)[[:digit:]]\{3\}/&-/g' phone.txt
(555)555-1212
(555)555-1213
(555)555-1214
(666)555-1215
(666)555-1216
(777)555-1217
```

Note: In the above example, instead of repeating the character class keyword [[:digit:]] three times, you replaced it with \{3\}, which means to match the preceding regular expression three times. Here I used \ to give line break you should remove this before running this command.

Back References:

The ampersand metacharacter is useful, but even more useful is the ability to define specific regions in a regular expressions so you can reference them in your replacement strings. By defining specific parts of a regular expression, you can then refer back to those parts with a special reference character.

To do back references, you have to first define a region and then refer back to that region. To define a region you insert backslashed parentheses around each region of interest. The first region that you surround with backslashes is then referenced by \1, the second region by \2, and so on.

Assuming phone.txt has the following text:

```
(555)555-1212
(555)555-1213
(555)555-1214
(666)555-1215
(666)555-1216
(777)555-1217
```

Now try the following command:

```
$ cat phone.txt | sed 's/\(.*)\)\(.*-\)\(.*$\)/Area \
                     code: \1 Second: \2 Third: \3/'
```

```
Area code: (555) Second: 555- Third: 1212
Area code: (555) Second: 555- Third: 1213
Area code: (555) Second: 555- Third: 1214
Area code: (666) Second: 555- Third: 1215
Area code: (666) Second: 555- Third: 1216
Area code: (777) Second: 555- Third: 1217
```

Note: In the above example each regular expression inside the parenthesis would be back referenced by \1, \2 and so on. Here I used \ to give line break you should remove this before running this command.

Unix - User Administration

There are three types of accounts on a Unix system:

1. **Root account:** This is also called superuser and would have complete and unfettered control of the system. A superuser can run any commands without any restriction. This user should be assumed as a system administrator.
2. **System accounts:** System accounts are those needed for the operation of system-specific components for example mail accounts and the sshd accounts. These accounts are usually needed for some specific function on your system, and any modifications to them could adversely affect the system.
3. **User accounts:** User accounts provide interactive access to the system for users and groups of users. General users are typically assigned to these accounts and usually have limited access to critical system files and directories.

Unix supports a concept of *Group Account* which logically groups a number of accounts. Every account would be a part of any group account. Unix groups plays important role in handling file permissions and process management.

Managing Users and Groups:

There are three main user administration files:

1. **/etc/passwd:** Keeps user account and password information. This file holds the majority of information about accounts on the Unix system.
2. **/etc/shadow:** Holds the encrypted password of the corresponding account. Not all the system support this file.
3. **/etc/group:** This file contains the group information for each account.
4. **/etc/gshadow:** This file contains secure group account information.

Check all the above files using **cat** command.

Following are commands available on the majority of Unix systems to create and manage accounts and groups:

Command	**Description**
useradd	Adds accounts to the system.
usermod	Modifies account attributes.
userdel	Deletes accounts from the system.
groupadd	Adds groups to the system.
groupmod	Modifies group attributes.
groupdel	Removes groups from the system.

You can use Manpage Help to check complete syntax for each command mentioned here.

Create a Group

You would need to create groups before creating any account otherwise you would have to use existing groups at your system. You would have all the groups listed in */etc/groups* file.

All the default groups would be system account specific groups and it is not recommended to use them for ordinary accounts. So following is the syntax to create a new group account:

```
 groupadd [-g gid [-o]] [-r] [-f] groupname
```

Here is the detail of the parameters:

Option	**Description**
-g GID	The numerical value of the group's ID.
-o	This option permits to add group with non-unique GID
-r	This flag instructs groupadd to add a system account
-f	This option causes to just exit with success status if the specified group already exists. With -g, if specified GID already exists, other (unique) GID is chosen
groupname	Actaul group name to be created.

If you do not specify any parameter then system would use default values.

Following example would create *developers* group with default values, which is very much acceptable for most of the administrators.

```
[amrood]$ groupadd developers
```

Modify a Group:

To modify a group, use the **groupmod** syntax:

```
[amrood]$ groupmod -n new_modified_group_name old_group_name
```

To change the developers_2 group name to developer, type:

```
[amrood]$ groupmod -n developer developer_2
```

Here is how you would change the financial GID to 545:

```
[amrood]$ groupmod -g 545 developer
```

Delete a Group:

To delete an existing group, all you need are the groupdel command and the group name. To delete the financial group, the command is:

```
[amrood]$ groupdel developer
```

This removes only the group, not any files associated with that group. The files are still accessible by their owners.

Create an Account

Let us see how to create a new account on your Unix system. Following is the syntax to create a user's account:

```
useradd -d homedir -g groupname -m -s shell -u userid accountname
```

Here is the detail of the parameters:

Option	**Description**
-d homedir	Specifies home directory for the account.
-g groupname	Specifies a group account for this account.
-m	Creates the home directory if it doesn't exist.
-s shell	Specifies the default shell for this account.
-u userid	You can specify a user id for this account.
accountname	Actual account name to be created

If you do not specify any parameter then system would use default values. The useradd command modifies the /etc/passwd, /etc/shadow, and /etc/group files and creates a home directory.
Following is the example which would create an account *mcmohd* setting its home directory to */home/mcmohd* and group as *developers*. This user would have Korn Shell assigned to it.

```
[amrood]$ useradd -d /home/mcmohd -g developers -s /bin/ksh mcmohd
```

Before issuing above command, make sure you already have *developers* group created using *groupadd* command.
Once an account is created you can set its password using the **passwd** command as follows:

```
[amrood]$ passwd mcmohd20
Changing password for user mcmohd20.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

When you type *passwd accountname*, it gives you option to change the password provided you are super user otherwise you would be able to change just your password using the same command but without specifying your account name.

Modify an Account:

The **usermod** command enables you to make changes to an existing account from the command line. It uses the same arguments as the useradd command, plus the -l argument, which allows you to change the account name.
For example, to change the account name *mcmohd* to *mcmohd20* and to change home directory accordingly, you would need to issue following command:

```
[amrood]$ usermod -d /home/mcmohd20 -m -l mcmohd mcmohd20
```

Delete an Account:

The **userdel** command can be used to delete an existing user. This is a very dangerous command if not used with caution.
There is only one argument or option available for the command: .r, for removing the account's home directory and mail file.
For example, to remove account *mcmohd20*, you would need to issue following command:

```
[amrood]$ userdel -r mcmohd20
```

If you want to keep her home directory for backup purposes, omit the -r option. You can remove the home directory as needed at a later time.

Unix - System Performance

The purpose of this tutorial is to introduce the performance analyst to some of the free tools available to monitor and manage performance on UNIX systems, and to provide a guideline on how to diagnose and fix performance problems in Unix environment.

UNIX has following major resource types that need to be monitored and tuned:

- **CPU**
- **Memory**
- **Disk space**
- **Communications lines**
- **I/O Time**
- **Network Time**
- **Applications programs**

Peformance Components:

There are following major five component where total system time goes:

Component	**Description**
User state CPU	The actual amount of time the CPU spends running the users program in the user state. It includes time spent executing library calls, but does not include time spent in the kernel on its behalf.
System state CPU	This is the amount of time the CPU spends in the system state on behalf of this program. All I/O routines require kernel services. The programmer can affect this value by the use of blocking for I/O transfers.
I/O Time and Network Time	These are the amount of time spent moving data and servicing I/O requests
Virtual Memory Performance	This includes context switching and swapping.
Application Program	Time spent running other programs - when the system is not servicing this application because another application currently has the CPU.

Peformance Tools:

Unix provides following important tools to measure and fine tune Unix system performance:

Command	**Description**
nice/renice	Run a program with modified scheduling priority
netstat	Print network connections, routing tables, interface statistics, masquerade connections, and multicast memberships
time	Time a simple command or give resource usage
uptime	System Load Average
ps	Report a snapshot of the current processes.
vmstat	Report virtual memory statistics
gprof	Display call graph profile data
prof	Process Profiling
top	Display system tasks

Unix - System Logging

Unix systems have a very flexible and powerful logging system, which enables you to record almost anything you can imagine and then manipulate the logs to retrieve the information you require.

Many versions of UNIX provide a general-purpose logging facility called *syslog*. Individual programs that need to have information logged send the information to syslog.

Unix *syslog* is a host-configurable, uniform system logging facility. The system uses a centralized system logging process that runs the program **/etc/syslogd** or **/etc/syslog**.

The operation of the system logger is quite straightforward. Programs send their log entries to *syslogd*, which consults the configuration file /etc/syslogd.conf or /etc/syslog and, when a match is found, writes the log message to the desired log file.

There are four basic syslog terms that you should understand:

Term	Description
Facility	The identifier used to describe the application or process that submitted the log message. Examples are mail, kernel, and ftp.
Priority	An indicator of the importance of the message. Levels are defined within syslog as guidelines, from debugging information to critical events.
Selector	A combination of one or more facilities and levels. When an incoming event matches a selector, an action is performed.
Action	What happens to an incoming message that matches a selector. Actions can write the message to a log file, echo the message to a console or other device, write the message to a logged in user, or send the message along to another syslog server.

Syslog Facilities:

Here are the available facilities for the selector. Not all facilities are present on all versions of UNIX.

Facility	Description
Auth	Activity related to requesting name and password (getty, su, login)
Authpriv	Same as auth but logged to a file that can only be read by selected users
Console	Used to capture messages that would generally be directed to the system console
cron	Messages from the cron system scheduler
daemon	System daemon catch-all
ftp	Messages relating to the ftp daemon
kern	Kernel messages
local0.local7	Local facilities defined per site
lpr	Messages from the line printing system

mail	Messages relating to the mail system
mark	Pseudo event used to generate timestamps in log files
news	Messages relating to network news protocol (nntp)
ntp	Messages relating to network time protocol
user	Regular user processes
uucp	UUCP subsystem

Syslog Priorities:

The syslog priorities are summarized in the following table:

Priority	**Description**
emerg	Emergency condition, such as an imminent system crash, usually broadcast to all users
alert	Condition that should be corrected immediately, such as a corrupted system database
crit	Critical condition, such as a hardware error
err	Ordinary error
warning	Warning
notice	Condition that is not an error, but possibly should be handled in a special way
info	Informational message
debug	Messages that are used when debugging programs
none	Pseudo level used to specify not to log messages.

The combination of facilities and levels enables you to be discerning about what is logged and where that information goes.

As each program sends its messages dutifully to the system logger, the logger makes decisions on what to keep track of and what to discard based on the levels defined in the selector.

When you specify a level, the system will keep track of everything at that level and higher.

The /etc/syslog.conf file:

The /etc/syslog.conf file controls where messages are logged. A typical syslog.conf file might look like this:

```
*.err;kern.debug;auth.notice /dev/console
daemon,auth.notice          /var/log/messages
lpr.info                    /var/log/lpr.log
mail.*                      /var/log/mail.log
ftp.*                       /var/log/ftp.log
auth.*                      @prep.ai.mit.edu
auth.*                      root,amrood
netinfo.err                 /var/log/netinfo.log
install.*                   /var/log/install.log
*.emerg                     *
*.alert                     |program_name
mark.*                      /dev/console
```

Each line of the file contains two parts:

- A message selector that specifies which kind of messages to log. For example, all error messages or all debugging messages from the kernel.
- An action field that says what should be done with the message. For example, put it in a file or send the message to a user's terminal.

Following are the notable points for the above configuration:

- Message selectors have two parts: a facility and a priority. For example, *kern.debug* selects all debug messages (the priority) generated by the kernel (the facility).
- Message selectetor *kern.debug* selects all priorities that are greater than debug.
- An asterisk in place of either the facility or the priority indicates "all." For example, *.debug means all debug messages, while kern.* means all messages generated by the kernel.
- You can also use commas to specify multiple facilities. Two or more selectors can be grouped together by using a semicolon.

Logging Actions:

The action field specifies one of five actions:

1. Log message to a file or a device. For example, /var/log/lpr.log or /dev/console.
2. Send a message to a user. You can specify multiple usernames by separating them with commas (e.g., root, amrood).
3. Send a message to all users. In this case, the action field consists of an asterisk (e.g., *).
4. Pipe the message to a program. In this case, the program is specified after the UNIX pipe

symbol (|).

5. Send the message to the syslog on another host. In this case, the action field consists of a hostname, preceded by an at sign (e.g., @tutorialspoint.com)

The logger Command:

UNIX provides the **logger** command, which is an extremely useful command to deal with system logging. The **logger** command sends logging messages to the syslogd daemon, and consequently provokes system logging.

This means we can check from the command line at any time the **syslogd** daemon and its configuration. The logger command provides a method for adding one-line entries to the system log file from the command line.

The format of the command is:

```
logger [-i] [-f file] [-p priority] [-t tag] [message]...
```

Here is the detail of the parameters:

Option	**Description**
-f filename	Use the contents of file filename as the message to log.
-i	Log the process ID of the logger process with each line.
-p priority	Enter the message with the specified priority (specified selector entry); the message priority can be specified numerically, or as a facility.priority pair. The default priority is user.notice.
-t tag	Mark each line added to the log with the specified tag.
message	The string arguments whose contents are concatenated together in the specified order, separated by the space

You can use Manpage Help to check complete syntax for this command.

Log Rotation:

Log files have the propensity to grow very fast and consume large amounts of disk space. To enable log rotations, most distributions use tools such as *newsyslog* or *logrotate*.

These tools should be called on a frequent time interval using the cron daemon. Check the man pages for *newsyslog* or *logrotate* for more details.

Important Log Locations

All the system applications create their log files in */var/log* and its sub-directories. Here are few important applications and their coressponding log directories:

Application	**Directory**
Httpd	/var/log/httpd
Samba	/var/log/samba
cron	/var/log/
mail	/var/log/
mysql	/var/log/

Unix - Signals and Traps

Signals are software interrupts sent to a program to indicate that an important event has occurred. The events can vary from user requests to illegal memory access errors. Some signals, such as the interrupt signal, indicate that a user has asked the program to do something that is not in the usual flow of control.

The following are some of the more common signals you might encounter and want to use in your programs:

Signal Name	**Signal Number**	**Description**
SIGHUP	1	Hang up detected on controlling terminal or death of controlling process
SIGINT	2	Issued if the user sends an interrupt signal (Ctrl + C).
SIGQUIT	3	Issued if the user sends a quit signal (Ctrl + D).
SIGFPE	8	Issued if an illegal mathematical operation is attempted
SIGKILL	9	If a process gets this signal it must quit immediately and will not perform any clean-up operations
SIGALRM	14	Alarm Clock signal (used for timers)
SIGTERM	15	Software termination signal (sent by kill by default).

List of Signals:

There is an easy way to list down all the signals supported by your system. Just issue **kill -l** command and it would display all the supported signals:

```
[amrood]$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

The actual list of signals varies between Solaris, HP-UX, and Linux.

Default Actions:

Every signal has a default action associated with it. The default action for a signal is the action that a script or program performs when it receives a signal.

Some of the possible default actions are:

- Terminate the process.
- Ignore the signal.
- Dump core. This creates a file called core containing the memory image of the process when it received the signal.
- Stop the process.
- Continue a stopped process.

Sending Signals:

There are several methods of delivering signals to a program or script. One of the most common is for a user to type CONTROL-C or the INTERRUPT key while a script is executing.

When you press the *Ctrl+C* key a SIGINT is sent to the script and as per defined default action script terminates.

The other common method for delivering signals is to use the kill command whose syntax is as follows:

```
[amrood]$ kill -signal pid
```

Here **signal** is either the number or name of the signal to deliver and **pid** is the process ID that the signal should be sent to. For Example:

```
[amrood]$ kill -1 1001
```

Sends the HUP or hang-up signal to the program that is running with process ID 1001. To send a kill signal to the same process use the folloing command:

```
[amrood]$ kill -9 1001
```

This would kill the process running with process ID 1001.

Trapping Signals:

When you press the *Ctrl+C* or Break key at your terminal during execution of a shell program, normally that program is immediately terminated, and your command prompt returned. This may not always be desirable. For instance, you may end up leaving a bunch of temporary files that won't get cleaned up.

Trapping these signals is quite easy, and the trap command has the following syntax:

```
$ trap commands signals
```

Here *command* can be any valid Unix command, or even a user-defined function, and signal can be a list of any number of signals you want to trap.

There are three common uses for trap in shell scripts:

1. Clean up temporary files
2. Ignore signals

Cleaning Up Temporary Files:

As an example of the trap command, the following shows how you can remove some files and then exit if someone tries to abort the program from the terminal:

```
$ trap "rm -f $WORKDIR/work1$$ $WORKDIR/dataout$$; exit" 2
```

From the point in the shell program that this trap is executed, the two files *work1$$* and *dataout$$* will be automatically removed if signal number 2 is received by the program.

So if the user interrupts execution of the program after this trap is executed, you can be assured that these two files will be cleaned up. The **exit** command that follows the **rm** is necessary because without it execution would continue in the program at the point that it left off when the signal was received.

Signal number 1 is generated for hangup: Either someone intentionally hangs up the line or the line gets accidentally disconnected.

You can modify the preceding trap to also remove the two specified files in this case by adding signal number 1 to the list of signals:

```
$ trap "rm $WORKDIR/work1$$ $WORKDIR/dataout$$; exit" 1 2
```

Now these files will be removed if the line gets hung up or if the *Ctrl+C* key gets pressed.

The commands specified to trap must be enclosed in quotes if they contain more than one command. Also note that the shell scans the command line at the time that the trap command gets executed and also again when one of the listed signals is received.

So in the preceding example, the value of WORKDIR and $$ will be substituted at the time that the trap command is executed. If you wanted this substitution to occur at the time that either signal 1 or 2 was received you can put the commands inside single quotes:

```
$ trap 'rm $WORKDIR/work1$$ $WORKDIR/dataout$$; exit' 1 2
```

Ignoring Signals:

If the command listed for trap is null, the specified signal will be ignored when received. For example, the command:

```
$ trap '' 2
```

Specifies that the interrupt signal is to be ignored. You might want to ignore certain signals when performing some operation that you don't want interrupted. You can specify multiple signals to be ignored as follows:

```
$ trap '' 1 2 3 15
```

Note that the first argument must be specified for a signal to be ignored and is not equivalent to writing the following, which has a separate meaning of its own:

```
$ trap  2
```

If you ignore a signal, all subshells also ignore that signal. However, if you specify an action to be taken on receipt of a signal, all subshells will still take the default action on receipt of that signal.

Resetting Traps:

After you've changed the default action to be taken on receipt of a signal, you can change it back again with trap if you simply omit the first argument; so

```
$ trap 1 2
```

resets the action to be taken on receipt of signals 1 or 2 back to the default.

Unix - Useful Commands

This quick guide lists commands, including a syntax and brief description. For more detail, use:

```
$man command
```

Files and Directories:

These commands allow you to create directories and handle files.

Command	**Description**
cat	Display File Contents
cd	Changes Directory to dirname
chgrp	change file group
chmod	Changing Permissions
cp	Copy source file into destination
file	Determine file type
find	Find files
grep	Search files for regular expressions.
head	Display first few lines of a file
ln	Create softlink on oldname
ls	Display information about file type.
mkdir	Create a new directory dirname

more	Display data in paginated form.
mv	Move (Rename) a oldname to newname.
pwd	Print current working directory.
rm	Remove (Delete) filename
rmdir	Delete an existing directory provided it is empty.
tail	Prints last few lines in a file.
touch	Update access and modification time of a file.

Manipulating data:

The contents of files can be compared and altered with the following commands.

Command	**Description**
awk	Pattern scanning and processing language
cmp	Compare the contents of two files
comm	Compare sorted data
cut	Cut out selected fields of each line of a file
diff	Differential file comparator
expand	Expand tabs to spaces
join	Join files on some common field
perl	Data manipulation language
sed	Stream text editor
sort	Sort file data
split	Split file into smaller files
tr	Translate characters
uniq	Report repeated lines in a file
wc	Count words, lines, and characters

vi	Opens vi text editor
vim	Opens vim text editor
fmt	Simple text formatter
spell	Check text for spelling error
ispell	Check text for spelling error
ispell	Check text for spelling error
emacs	GNU project Emacs
ex, edit	Line editor
emacs	GNU project Emacs
emacs	GNU project Emacs

Compressed Files:

Files may be compressed to save space. Compressed files can be created and examined:

Command	**Description**
compress	Compress files
gunzip	Uncompress gzipped files
gzip	GNU alternative compression method
uncompress	Uncompress files
unzip	List, test and extract compressed files in a ZIP archive
zcat	Cat a compressed file
zcmp	Compare compressed files
zdiff	Compare compressed files
zmore	File perusal filter for crt viewing of compressed text

Getting Information:

Various Unix manuals and documentation are available on-line. The following Shell commands give information:

Command	Description
apropos	Locate commands by keyword lookup
info	Displays command information pages online
man	Displays manual pages online
whatis	Search the whatis database for complete words.
yelp	GNOME help viewer

Network Communication:

These following commands are used to send and receive files from a local UNIX hosts to the remote host around the world.

Command	Description
ftp	File transfer program
rcp	Remote file copy
rlogin	Remote login to a UNIX host
rsh	Remote shell
tftp	Trivial file transfer program
telnet	Make terminal connection to another host
ssh	Secure shell terminal or command connection
scp	Secure shell remote file copy
sftp	secure shell file transfer program

Some of these commands may be restricted at your computer for security reasons.

Messages between Users:

The UNIX systems support on-screen messages to other users and world-wide electronic mail:

Command	Description
evolution	GUI mail handling tool on Linux
mail	Simple send or read mail program

mesg	Permit or deny messages
parcel	Send files to another user
pine	Vdu-based mail utility
talk	Talk to another user
write	Write message to another user

Programming Utilities:

The following programming tools and languages are available based on what you have installed on your Unix.

Command	**Description**
dbx	Sun debugger
gdb	GNU debugger
make	Maintain program groups and compile programs.
nm	Print program's name list
size	Print program's sizes
strip	Remove symbol table and relocation bits
cb	C program beautifier
cc	ANSI C compiler for Suns SPARC systems
ctrace	C program debugger
gcc	GNU ANSI C Compiler
indent	Indent and format C program source
bc	Interactive arithmetic language processor
gcl	GNU Common Lisp

perl	General purpose language
php	Web page embedded language
py	Python language interpreter
asp	Web page embedded language
CC	C++ compiler for Suns SPARC systems
g++	GNU C++ Compiler
javac	JAVA compiler
appletvieweir	JAVA applet viewer
netbeans	Java integrated development environment on Linux
sqlplus	Run the Oracle SQL interpreter
sqlldr	Run the Oracle SQL data loader
mysql	Run the mysql SQL interpreter

Misc Commands:

These commands list or alter information about the system:

Command	**Description**
chfn	Change your finger information
chgrp	Change the group ownership of a file
chown	Change owner
date	Print the date
determin	Automatically find terminal type
du	Print amount of disk usage
echo	Echo arguments to the standard options
exit	Quit the system
finger	Print information about logged-in users

groupadd	Create a user group
groups	Show group memberships
homequota	Show quota and file usage
iostat	Report I/O statistics
kill	Send a signal to a process
last	Show last logins of users
logout	log off UNIX
lun	List user names or login ID
netstat	Show network status
passwd	Change user password
passwd	Change your login password
printenv	Display value of a shell variable
ps	Display the status of current processes
ps	Print process status statistics
quota -v	Display disk usage and limits
reset	Reset terminal mode
script	Keep script of terminal session
script	Save the output of a command or process
setenv	Set environment variables
stty	Set terminal options
time	Time a command
top	Display all system processes
tset	Set terminal mode
tty	Print current terminal name

umask	Show the permissions that are given to view files by default
uname	Display name of the current system
uptime	Get the system up time
useradd	Create a user account
users	Print names of logged in users
vmstat	Report virtual memory statistics
w	Show what logged in users are doing
who	List logged in users

Unix - Shell Builtin Functions

The most of the part of this tutorial covered Bourne Shell but this page list down all the mathematical builti-in functions available in **Korn** Shell.

The Korn shell provides access to the standard set of mathematical functions. They are called using C function call syntax.

Function	**Description**
abs	Absolute value
log	Natural logarithm
acos	Arc cosine
sin	Sine
asin	Arc sine
sinh	Hyperbolic sine
cos	Cosine
sqrt	Square root
cosh	Hyperbolic cosine
tan	Tangent

exp	Exponential function
tanh	Hyperbolic tangent
int	Integer part of floating-point number

Unix - Basic Utilities

So far you must have got some idea about Unix OS and nature of its basic commands. This tutorial would cover few very basic but important Unix utilities which you would use in your day to day life.

Printing Files:

Before you print a file on a UNIX system, you may want to reformat it to adjust the margins, highlight some words, and so on. Most files can also be printed without reformatting, but the raw printout may not look quite as nice.

Many versions of UNIX include two powerful text formatters, **nroff** and **troff**. They are not covered in this tutorial but you would quit a lot material on the net for these utilities.

The pr Command:

The **pr** command does minor formatting of files on the terminal screen or for a printer. For example, if you have a long list of names in a file, you can format it onscreen into two or more columns.

Here is the syntax of **pr** command:

```
pr option(s) filename(s)
```

The **pr** changes the format of the file only on the screen or on the printed copy; it doesn't modify the original file. Following table lists some pr options:

Option	**Description**
-k	Produces k columns of output
-d	Double-spaces the output (not on all pr versions).
-h "header"	Takes the next item as a report header.
-t	Eliminates printing of header and top/bottom margins.
-l PAGE_LENGTH	Set the page length to PAGE_LENGTH (66) lines. Default number of lines of text 56.

-o MARGIN	Offset each line with MARGIN (zero) spaces.
-w PAGE_WIDTH	Set page width to PAGE_WIDTH (72) characters for multiple text-column output only.

Before using **pr**, here are the contents of a sample file named food

```
[amrood]$cat food
Sweet Tooth
Bangkok Wok
Mandalay
Afghani Cuisine
Isle of Java
Big Apple Deli
Sushi and Sashimi
Tio Pepe's Peppers
........
[amrood]$
```

Let's use **pr** command to make a two-column report with the header *Restaurants*:

```
[amrood]$pr -2 -h "Restaurants" food
Nov  7  9:58 1997  Restaurants   Page 1

Sweet Tooth            Isle of Java
Bangkok Wok              Big Apple Deli
Mandalay               Sushi and Sashimi
Afghani Cuisine          Tio Pepe's Peppers
........
[amrood]$
```

The lp and lpr Commands:

The command **lp** or **lpr** prints a file onto paper as opposed to the screen display. Once you are ready with formatting using **pr** command, you can use any of these commands to print your file on printer connected with your computer.

Your system administrator has probably set up a default printer at your site. To print a file named food on the default printer, use the lp or lpr command, as in this example:

```
[amrood]$lp  food
request id is laserp-525  (1 file)
[amrood]$
```

The lp command shows an ID that you can use to cancel the print job or check its status.

- If you are using lp command, you can use -n**Num** option to print Num number of copies. Along with the command lpr, you can use **-Num** for the same.
- If there are multiple printers connected with the shared network, then you can choose a printer using -d**printer** option along with lp command and for the same purpose you can use -P**printer** option along with lpr command. Here printer is the printer name.

The lpstat and lpq Commands:

The lpstat command shows what's in the printer queue: request IDs, owners, file sizes, when the jobs were sent for printing, and the status of the requests.

Use lpstat -o if you want to see all output requests rather than just your own. Requests are shown in the order they'll be printed:

```
[amrood]$lpstat -o
laserp-573  john  128865  Nov 7  11:27  on laserp
laserp-574  grace  82744  Nov 7  11:28
laserp-575  john   23347  Nov 7  11:35
[amrood]$
```

The **lpq** gives slightly different information than lpstat -o:

```
[amrood]$lpq
laserp is ready and printing
Rank   Owner      Job  Files                  Total Size
active john       573  report.ps             128865 bytes
1st    grace      574  ch03.ps ch04.ps        82744 bytes
2nd    john       575  standard input         23347 bytes
[amrood]$
```

Here the first line displays the printer status. If the printer is disabled or out of paper, you may see different messages on this first line.

The cancel and lprm Commands:

The **cancel** terminates a printing request from the lp command. The **lprm** terminates lpr requests. You can specify either the ID of the request (displayed by lp or lpq) or the name of the printer.

```
[amrood]$cancel laserp-575
request "laserp-575" cancelled
[amrood]$
```

To cancel whatever request is currently printing, regardless of its ID, simply enter cancel and the printer name:

```
[amrood]$cancel laserp
request "laserp-573" cancelled
[amrood]$
```

The lprm command will cancel the active job if it belongs to you. Otherwise, you can give job numbers as arguments, or use a dash (-) to remove all of your jobs:

```
[amrood]$lprm 575
dfA575diamond dequeued
cfA575diamond dequeued
[amrood]$
```

The lprm command tells you the actual filenames removed from the printer queue.

Sending Email:

You use the Unix mail command to send and receive mail. Here is the syntax to send an email:

```
[amrood]$mail [-s subject] [-c cc-addr] [-b bcc-addr] to-addr
```

Here are important options related to mail command:

Option	**Description**
-s	Specify subject on command line.
-c	Send carbon copies to list of users. List should be a comma-separated list of names.
-b	Send blind carbon copies to list. List should be a comma-separated list of names.

Following is the example to send a test message to amrood@gmail.com.

```
[amrood]$mail -s "Test Message" admin@yahoo.com
```

You are then expected to type in your message, followed by an "control-D" at the beginning of a line. To stop simply type dot (.) as follows:

```
Hi,

This is a test
.
Cc:
```

You can send a complete file using a redirect < operator as follows:

```
[amrood]$mail -s "Report 05/06/07" admin@yahoo.com < demo.txt
```

To check incoming email at your Unix system you simply type email as follows:

```
[amrood]$mail
no email
```