UNIX

What is UNIX?

UNIX is an operating system that

- Acts as an interface between the user and a computer,
- Enables the user to use the computer,
- Enables the user to start programs (email, spreadsheet, database, etc.),
- Enables to send data to printer, etc.

Comparison between Windows and UNIX

- UNIX is more powerful and complex.
- UNIX has excellent network facilities.
- UNIX supports a large company that depends on an extensive network to handle high volumes of traffic and to serve critical applications and information.
- Windows servers tend to go down – stop working properly.
- UNIX servers tend to go perfectly.
- Windows has more security bugs (problems with the way the system behaves).
- Windows does not have built-in security features.
- Windows servers have a four-processor limit.
- UNIX machines can handle many processors.
- UNIX can handle large files.

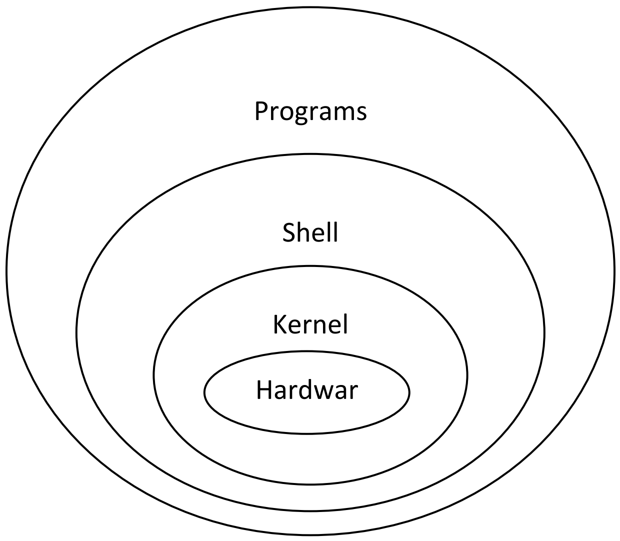
UNIX features

- Programmer's environment.
- Simple interface.
- Simple utilities that can be combined to perform powerful functions.
- Hierarchical file system.
- Simple interface to devices.
- Multiuser, multiprocessor system.
- Architecture independent and transparent to the user.

UNIX structure

- UNIX is a layered operating system.
- The innermost layer is hardware that provides the services for the operating system.
- The operating system, referred to as kernel in UNIX, interacts directly with hardware and provides services to user programs.
- User programs do not need to know anything about hardware. But, they just need to know how to interact with the kernel.
- User programs interact with the kernel through a set of standard system calls. These system calls request to be provided by the kernel.

- Such services include:



  - Accessing files (open, close, read, write, link, or execute)
  - Updating records
  - Changing ownership of a file or a directory
  - Changing to a new directory
  - Creating, suspending, or killing a process
  - Enabling access to hardware devices
  - Setting limits on system resources, etc.

Shell

- It is part of the operating system that interacts with the user.
- It provides you an interface to the UNIX system.
- It is a program that reads your command input and runs the specified commands.
- It gathers input from the user and executes programs based on that input. When the program finishes executing, it displays program's output.
- It is UNIX system's command interpreter. A command is a program that you can run.
- In UNIX there are two major types of shells:
  - The Bourne shell (includes: sh, ksh, and bash) has a prompt $.
  - The C shell (includes: csh, tsch) has a prompt %.

Logging in

- One should log in before s/he can use the system.
- To log in: type in your username and password.
  - Username is used to identify you to the system.

- Password is used to prove your identity.
- The login prompt – is an instruction that asks you to type in your username. It has the form: login as:. You then respond by typing in your username and then pressing Return key.
- The password prompt – is an instruction that asks you to type your password.
- The user prompt has the form something $ or something %. If so, the system is ready to accept commands.

Changing your password

A password should contain at least six characters.

passwd – is a command used to set/update password.

To change your password:

1. Type passwd.
2. Type your current password.
3. Type new password.
4. Type the new password again.

Logging out

It is the process of closing your account.

exit – is a command used to log out.

UNIX Command line structure

A command is a program that tells a UNIX system to do something. It has the form:

command [option(s)] [argument(s)]

Where:

- command – is the name of UNIX command such as ls, cp, cat, etc.
- options:
  - make commands operate in a different way.
  - are usually single letters and prefixed by a minus sign (-).
  - some commands allow several options to be specified after one minus (-aux), while a few commands require each option to be given its own minus sign (-a -u -x)
- arguments:
  - tell commands where to find data.
  - give the names of the files on which the command is to operate.

Note: UNIX is case sensitive.

Files and directories

Files are collections of information.

- Files can be of several types:

    - text files (contain ordinary text information such as reports, letters, etc)
    - data files (contain information which can only be read by the program which created them)
    - configuration files (are usually text files containing information about fonts, colors, etc.)
    - hidden files
- Files are stored in directories. Directory is a structure used to organize files.
- Filenames can have special endings. e.g. text files sometimes end in ".txt" and C program files end in ".c". Names of hidden files start with dot. Configuration files are often hidden files.

File manipulating commands

- ls, more, cat, cp, mv, rm, etc.

ls command

- lists directory contents. For example, if you type **ls** and press Enter key, the contents under the current directory will be listed.
- You can also list files as a comma separated list by using -m option.

  ```
  $ ls –m
  ```
- -F (tell file type) option causes the **ls** command to show the directories each with a '/' character appended to the directory name. Asterisk (*), if appended, shows the file is an executable file.
- -a option is used to display a list of all the files a directory including files which are hidden. Hidden files have names which begin with a dot (.) (such as .profile) and contain configuration information.
- $ ls -aF shows all files with their type.
- -l option gives long format listing.

  ```
  $ ls -l

  drwxr-xr-x  2  abebe  abebe  4096  2008-11-17  12:24  Desktop
  ```

  Column 1 – file permission flags

  Column 2 – number of links

  Column 3 – owner name

  Column 4 – group name

  Column 5 – file size

  Column 6 – date

  Column 7 – timestamp of the file or directory (created or last modified)

  Column 8 – filename

- To view the contents of other directories, specify the directory or path on the command line.

  e.g., list all files in the /usr/bin directory

  $ ls /usr/bin
- **ls** command supports wildcards and regular expressions.

  Wildcards:

?	any single letter
*	anything at all

  Examples:

  $ ls ?? – lists all files that have two-character names.

  $ ls *.txt – lists only text files in the current directory.

tree command

- lists contents of directories in tree-like format.
- -d option is used to list only directories, not files.

cat command

- cat (concatenate file) command is used to send the contents of files to screen.
- it is used for displaying short files.
- It is used for combining, creating, overwriting, or appending files.
- It is used to create a short file.
- It reads the standard input.

Examples:

1. cat is used to create short files.
   $ cat > filename.txt
   Type your text here.
   Press Ctrl + D to return to the command prompt.
   Note: > is called redirection symbol.
2. cat is used to display the contents of short files.
   $ cat filename.txt
3. cat is used to display contents of a file with line numbers on the screen using –n option.
   $ cat -n filename.txt
4. cat is used to write the contents of one file on to another file.
   $ cat filename1.txt > filename2.txt
   Note: If filename2.txt has content, it is overwritten by filename1.txt.
5. cat is used to append the contents of one file on another file using >> symbol.
   $ cat filename1.txt >> filename2.txt
   Note: filename1.txt does not overwrite filename2.txt.

more command

- The more command is one of a family of Linux commands called pagers. Pager commands let you browse through files, reading a screen or line at a time.

  $ more longfile.txt

  Note: Using more command, you can advance one screen (the current size) by tapping the spacebar and you can go backward one screen by tapping the B key.

less command

- is used to browse files.
- less is more or less like more.
- like more, less is also a pager command.
- The less command offers a number of advantages over more, for example, you can scroll backwards and forwards through text files using your cursor keys

head command

- not pager.
- reads the beginning of a file.

  $ head filename.txt
- head with -n option prints the first n lines.

  $ head -5 filename.txt – prints the first five lines of the file filename.txt.

tail command

- reads the end of a file.

  $ tail filename.txt

rm command

- used to delete a file.

  rm filename.txt – removes filename.txt.
- the rm command is also used to remove many files.

  $ rm filename1.txt filename2.txt filename3.txt – removes the three files filename1.txt, filename2.txt and filename3.txt.

mkdir command

- used to create a directory.
- can create one or several directories with a single command line.
- can also create a whole hierarchy of directories, which includes parent and children, with a single command line.

  $ mkdir directory1 – creates a single directory directory1.

  $ mkdir temp1 temp2 temp3 – creates three directories: temp1, temp2 and temp3.

  Note: To build a hierarchy of directories, use mkdir with the-p (parent) option.

$ mkdir -p abc/def/ghi – creates three directories: abc, def and ghi. def inside abc and ghi inside def.

rmdir command

- is used to remove empty directories.

  rmdir abc – removes the directory abc. The directory abc must be empty. If you try to remove a directory that has a file or subdirectory, you will get an error message like this: Directory not empty.
- You can use the rm command to remove the files first.
- The rmdir command, like mkdir, also has a -p, or parent, option.
- You can use the -p option to remove directory hierarchies.
- Exmaples:
  1. $ rmdir abc/def/ghi – removes the directory ghi if ghi is empty.
  2. Suppose we have the following hierarchy of directories:

```
aa
  | __ cc
         | __ dd
                | __ ee
```

  $ rmdir aa/cc/dd/ee – removes ee since it is empty.
  $ rmdir aa/cc/dd – does not remove dd since it contains ee.
  $ rmdir -p aa/cc/dd/ee – removes the directory structure.

mv command

- The mv command is used to rename files or directories and also to move them around your file system.
- In its simplest form, mv can rename files, for example:

  $ mv file1 file2 – renames file1 by file2. The file file1 disappears, but there will now be a file called file2.

  Note: Besides renaming files, mv can also rename directories, whether empty or not, for example:

  $ mkdir -p temp/temp2/temp3 – this creates a hierarchy of directories.

  $ mv temp newtemp – renames temp by newtemp.
- mv command can also be used to move files and directories to different locations in a directory tree. The basic syntax is:

  mv source destination

  Here source is the name of the file or directory you want to move and destination is the directory where you want the file or directory to end up. For example:

  $ mv /home/ranga/names /tmp

  moves the file names located in the directory /home/ranga to the directory /tmp.

  Moving directory is exactly the same. For example,

  $ mv docs/ work/ moves the directory docs into the directory work.
- One nice feature of mv is that you can move and rename a file or directory all in one command. For example:

$ mv docs/names /tmp/names.txt

moves the file names in the directory docs to the directory /tmp and renames it with names.txt.

- As you can with cp, you can specify more than one file or directory as the source. For example:

  $ mv work/ docs/ .profile pub/

  moves the directories work and docs along with the file .profile into the directory pub. When you move multiple items you can not renames them. If you want to rename an item and move it, you must use a separate mv command for each item.

cp command

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

  cp source destination
- Here source is the name of the file that to be copied and destination is the name of the copy. For example, the following command makes a copy of the file test_results and places the copy in a file named test_results.orig:

  S cp test_results test_results.orig
- If the destination is a directory, the copy has the same name as the source but it is located in the destination directory. For example, the command:

  $ cp test_results work/ places a copy of the file test_results in the directory work by the same file name (i.e., test_results).
- $ test_results work/test_results_copy copies test_results to the file test_results_copy in the directory work.
- If more than two inputs are given, cp treats the last argument as the destination and the other files as sources. This works only if the sources are files and the destination is a directory, as in the following example:

  $ cp res.01 res.02 res.03 work/ copies the files res.01, res.02 and res.03 to the directory work by the same names.
- To copy a directory, you specify the -r option to cp. The syntax is as follows:

  cp -r source destination

  Here, source is the pathname of the directory you want to copy, and destination is where you want to place the copy. For example:

  $ cp -r docs/book /mnt/zip copies the directory book located in the docs directory to the directory /mnt/zip. It creates a new directory called book under /mnt/zip.
- cp can also be used to copy multiple directories. If cp encounters more than one source, all the source directories are copied to the destination. The destination is assumed to be the last argument. For example, the command:

  $ cp -r docs/book docs/school work/src /mnt/zip copies the directories school and book, located in the docs directory, to /mnt/zip. It also copies the directory src, located in the directory work, to /mnt/zip.

Manipulating file attributes

Owners, groups, and permissions

File ownership is an important component of UNIX that provides a secure method for storing files. Every file in UNIX has the following attributes:

- Owner permissions
- Group permissions
- Other (world) permissions

The owner's permissions determine what actions the owner of the file can perform on the file. The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file. The permissions for others indicate what action all other users can perform on the file.

You can perform the following actions on a file:

- Read
- Write
- Execute

If a user has read permission, that person can view the contents of a file. A user with write permission can change the contents of a file, whereas a user with execute permissions can run a file as a program.

Viewing Permissions

You can display the permissions of a file using the ls -l command.

For example, the following command

$ ls -l /home/ranga/.profile produces the following output:

```
-rwxr-xr-x 1 ranga users 2368 Jul 11 15:57 .profile*
```

Because the first character is a hyphen (-), you know that this is a regular file. Several characters appear after this hyphen. The first three characters indicate the permissions for the owner of the file, the next three characters indicate the permissions for the group the file is associated with, and the last three characters indicate the permissions for all other users.

The permission block for this file indicates that the user has read, write, and execute permissions, whereas members of the group users and all other users have only read and execute permissions.

Three basic permissions that can be granted or denied on a file are read, write, and execute. These permissions are defined in Table 1.

After the permissions block, the owner and the group are listed. For this file, the owner is ranga and the group is users.

Letter Permission Definition

Table 1: Letter permission

r	read	The user can view the contents of the file.
w	write	The user can alter the contents of the file.
x	execute	The user can run the file, which is likely a program. For directories, the execute permission must be set in order for users to access the directory.

Directory Permissions

The read permission on a directory enables users to use the ls command to view files and their attributes that are located in the directory.

The write permission on a directory is the permission to watch out for because it lets a user add and also remove files from the directory.

A directory that grants a user only execute permission will not enable the user to view the contents of the directory or add or delete any files from the directory, but it will let the user run executable files located in the directory.

Changing File and Directory Permissions

You can change file and directory permissions with the chmod command. The basic syntax is as follows:

chmod expression files

Here, expression is a statement of how to change the permissions. This expression can be one of the following types:

- Symbolic
- Octal

The symbolic expression method uses letters to alter the permissions, and the octal expression method uses numbers. The numbers in the octal (base 8) method are numbers ranging from 0 to 7.

Symbolic Method

The symbolic expression has the syntax of

(who)(action)(permission)

Table 2 shows the possible values for who, Table 3, shows the possible actions, and Table 4, shows the possible permissions settings. Using these three reference tables, you can build an expression.

Table 2: Who

Letter	Represents
u	Owner
g	Group
o	Other
a	All

Table 3: Actions

Symbol	Represents
+	Adding permissions to the file
-	Removing permission from the file
=	Explicitly set the file permissions

Table 4: Permissions

Letter	Represents
r	Read
w	Write
x	Execute

Now look at a few examples of using chmod:

1. To give the "world" read access to all files in a directory, you can use one of the following commands:

   ```
   $ chmod a=r *
   ```

   or

   ```
   $ chmod guo=r *
   ```

2. To stop anyone except the owner of the file .profile from writing to it, try this:

   ```
   $ chmod go-w .profile
   ```

3. To deny access to the files in your home directory, you can try the following:

   ```
   $ cd; chmod go= *
   ```

   or

   ```
   $ cd; chmod go-rwx *
   ```

4. When specifying the users part or the permissions part, the order in which you give the letters is irrelevant. Thus these commands are equivalent:

   ```
   $ chmod guo+rx *
   $ chmod uog+xr *
   ```

5. If you need to apply more than one set of permissions to a file or files, use a comma separated list. For example

   ```
   $ chmod go-w, a+x a.out
   ```

   removes the groups and "world" write permission on a.out and adds the execute permission for everyone.

Octal Method

By changing permissions with an octal expression, you can only explicitly set file permissions. This method uses a single number to assign the desired permission to each of the three categories of users (owner, group, and others).

The values of the individual permissions are the following:

- Read permission has a value of 4
- Write permission has a value of 2
- Execute permission has a value of 1

Adding the values of the permissions that you want to grant will give you a number between 0 and 7. This number will be used to specify the permissions for the owner, group, and finally the others category.

Examples:

1. In order to set the “world” read access to all files in a directory, do this:
   ```
   $ chmod 444 *
   ```
2. To stop anyone except the owner of the file .profile from writing to it, do this:
   ```
   chmod 600 .profile
   ```

Key combinations in Bash

Table 5 shows several special key combinations that allow you to do things easier and faster in Bash.

Table 5: Key combinations and their functions

Ctrl+A	Move cursor to the beginning of the command line.
Ctrl+C	End a running program and return the command prompt.
Ctrl+D	Log out of the current shell session, equal to typing exit or logout.
Ctrl+E	Move cursor to the end of the command line.
Ctrl+H	Generate backspace character.
Ctrl+K	Delete characters from the cursor’s position to the end of command line.
Ctrl+L	Clear this terminal.
Ctrl+R	Search command history.
Ctrl+U	Delete all characters at the command line.
Ctrl+Z	Suspend a program.
Left Arrow and Right Arrow	Move the cursor one place to the left or right on the command line, so that you can insert characters at other places than just at the beginning and the end.
Up Arrow and Down Arrow	Browse history. Go to the line that you want to repeat, eventually edit details, and press Enter to save time.
Shift+PageUp and Shift+PageDown	Browse terminal buffer (to see text that has "scrolled off" the screen).
Tab	Command or filename completion; when multiple choices are possible, the system will either signal with an audio or visual bell, or, if too many choices are possible, asks you if you want to see them all.
Tab Tab	Shows file or command completion possibilities.