Download files

Project description

SFS (Symbolic File System)

A command line utility that provides a lightweight setup for organizing and backing up files

SFS stores files from a variety of sources, aka collections, that may include directories and removable media, as symbolic links to the source files. It also stores the metadata of the source files so that files can later be queried without having to plug in the source media.

An SFS is a managed directory which is initialized with the command: `sfs init`. All commands to be executed in the context of an individual SFS must be run from within the SFS directory tree. Files are added using the command `sfs add-col my_collection /path/to/source` (add collection). *SFS Files* are symlinks to source files in added collections. Foreign links and other files can also exist in an SFS but they are not managed by it and are mostly ignored.

Use Cases

- Organizing Data Across Discs

Use Cases

- **Organizing Data Across Discs**

  SFS was built with the motivation of being able to have a combined view of data stored across multiple discs, organize the data in the view and reflect changes back to source discs. This is an effortless way of organizing content across discs which is otherwise painfully slow and limited as we can operate on a limited number of discs simultaneously and inter disc transfers are very slow. Since all operations in an SFS are performed within the same disc and on symlinks instead of heavy files, they are much faster

  Note: To view the content of a file we obviously do need the source to be available. So, if there is a need of viewing file content while organizing them, the source needs to be plugged in which might or might not be appropriate for all use cases. However, SFS makes it easy to query the source of an SFS File when it is needed to be accessed

- **Backing up Files**

  While there are lots of ways to make direct backups of directories, an SFS allows you to organize the content while backing them up and potentially saving them to multiple

More results

What are the types of directories?

What is symbolic directory?

What is a symbolic file?

What is symlink directory?

Symlinks, or symbolic links, are **“virtual” files or folders which reference a physical file or folder located elsewhere**, and are an important feature built in to many operating systems, including Linux and Windows. The Windows' NTFS file system has supported symlinks since Windows Vista. 02-Dec-2016

https://blogs.windows.com › symlin...

Symlinks in Windows 10! - Windows Developer Blog

More results

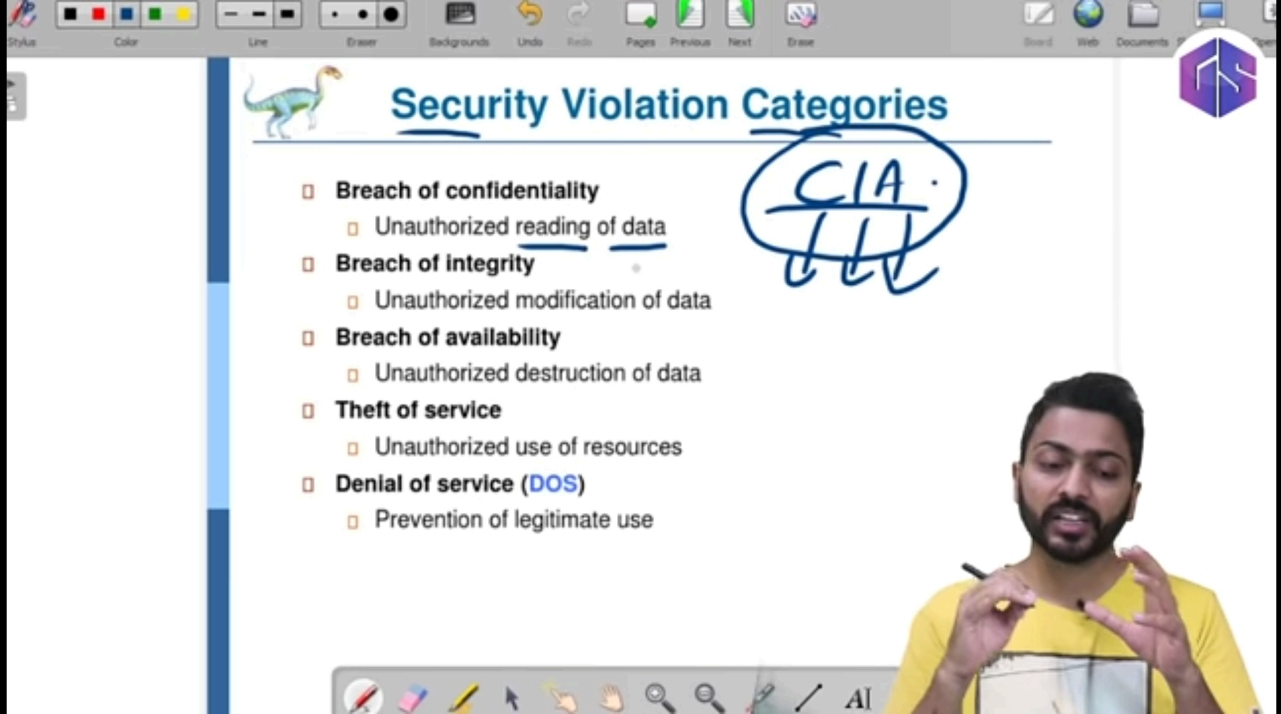
Is symbolic link a file or directory?

Physical files contain the actual data that is stored on the system, and a description of how data is to be presented to or received from a program. They contain only one record format, and one or more members. Records in database files can be externally or program-described.

A physical file can have a keyed sequence access path. This means that data is presented to a program in a sequence based on one or more key fields in the file.

Logical files do not contain data. They contain a description of records found in one or more physical files. A logical file is a view or representation of one or more physical files. Logical files that contain more than one format are referred to as **multi-format** logical files.

If your program processes a logical file which contains more than one record format, you can use a read by record format to set the format you wish to use.

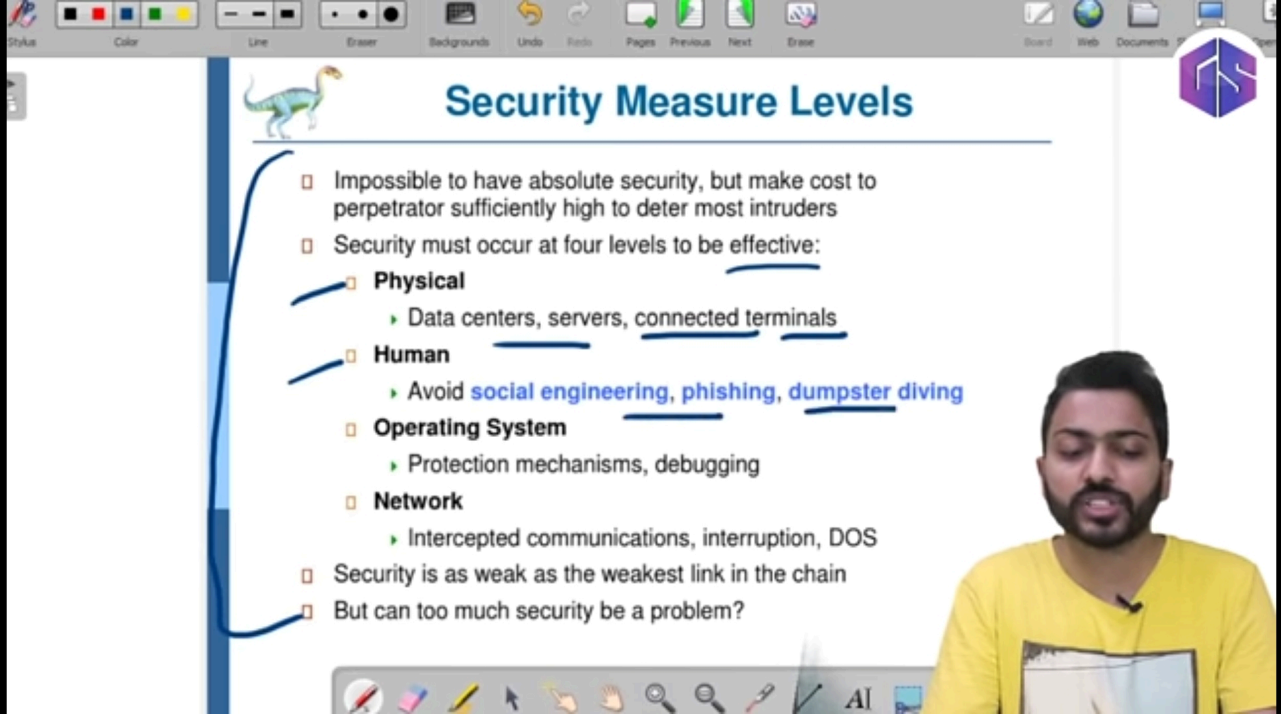

Stylus
Color
Line
Eraser
Backgrounds
Undo
Redo
Pages
Previous
Next
Erase
Board
Web
Documents
Security Violation Categories
CIA
Breach of confidentiality
Unauthorized reading of data
Breach of integrity
Unauthorized modification of data
Breach of availability
Unauthorized destruction of data
Theft of service
Unauthorized use of resources
Denial of service (DOS)
Prevention of legitimate use
SUBSCRIBE

Security Violation Methods

- **Masquerading** (breach **authentication**)
  - Pretending to be an authorized user to escalate privileges
- **Replay attack**
  - As is or with **message modification**
- **Man-in-the-middle attack**
  - Intruder sits in data flow, masquerading as sender to receiver and vice versa
- **Session hijacking**
  - Intercept an already-established session to bypass authentication

The Security Problem

- System **secure** if resources used and accessed as intended under all circumstances
  - Unachievable
- **Intruders** (**crackers**) attempt to breach security
- **Threat** is potential security violation
- **Attack** is attempt to breach security
- Attack can be accidental or malicious
- Easier to protect against accidental than malicious misuse


Stylus
Color
Line
Eraser
Backgrounds
Undo
Redo
Pages
Previous
Next
Erase
Board
Web
Documents
Security Measure Levels
Impossible to have absolute security, but make cost to perpetrator sufficiently high to deter most intruders
Security must occur at four levels to be effective:
Physical
Data centers, servers, connected terminals
Human
Avoid social engineering, phishing, dumpster diving
Operating System
Protection mechanisms, debugging
Network
Intercepted communications, interruption, DOS
Security is as weak as the weakest link in the chain
But can too much security be a problem?
SUBSCRIBE

their advantage, and shortcomings along with the difference between the Access control list and the capability list.

Access Control List:

Access Control lists can be created by splitting the access matrix column-wise. Access Control List is the object-wise list that specifies the list of subjects that have access to a particular object along with their access right. In simple terms, the ACL. of an object defines the list of users and the operations that they can perform on that object. Each object has a security attribute that identifies its access control list. A filesystem Access Control list is a data structure containing entries that specify an individual user or group's rights to specific system objects such as programs, processes, or files. These entries are known as **access-control entries.**

Advantages of Access Control List:

- It is easy to change by removing the entry of the subject from the object's access control list.
- It is easy to review access by directly examining the access control list of objects.

Disadvantages of Access Control List:

- It imposes search overhead and results in poor efficiency as the entire access control list needs to be searched when access is made to an object.
- It requires more data storage space as data is stored object-wise and the same subject can have multiple access to multiple objects thereby consuming more storage space.

Capability Lists:

Capability lists can be created by splitting the access matrix row-wise. A capability list is a subject-wise list that specifies the list of rights the subject has for every object. Thus, the capability list of a user or a process or domain is a list of rights that it has on the various objects. A capability consists of two fields-object descriptor access rights. An object descriptor is an identifier for an object and access right indicates various operations such as read, write execute, etc. granted to an object. A capability can be given as a pair (x, r) where **x** is the name of an object and **r** is a set of privileges or rights.

Advantages of Capability List:

- It is efficient as it frequently checks the validity of an address.
- It is flexible as users are allowed to define certain parameters.
- It is simple to understand as it allows natural correspondence between subjects and objects.

Limitations of Capability Lists:

- It is difficult to deallocate memory that is not currently in use.
- It is difficult to change access rights once assigned to subjects.
- It has complicated control of the propagation of various access rights.
- It is difficult to review the access provided to various subjects.

Difference between the Access Control list and Capability list:

Sr. No	Access Control Lists	Capability Lists
1.	It is defined object-wise (resources).	It is defined subject-wise (users, processes, and procedures).
2.	It lists the various subjects along with the rights of an object.	It lists the various objects along with the rights permitted on them for a subject.
3.	Each object (resource) has a list of pairs of the form <subject, access rights>	Each subject (user, process procedure) has a list of pairs of the form <object, access rights>
4.	It would be tedious to have separate listings for each object (user), therefore, they are grouped into classes. For example, in UNIX, there are three classes self, group, and anybody else.	Here capabilities are the names of the objects. The objects not referred to in at capability list cannot be ever named.
5.	The default is: Everyone should be able to access a file.	The default is: No one should be able to access a file unless they have been given a capability.
6.	Access lists are simple and are used in almost all file systems.	Capabilities are used in systems that need to be very secure as they prohibit sharing of information unless access is given to a subject.

Access Control Matrix

Alice	rwx	r--	r--
Bob	r--	r--	rwx
Charlie	r--	rwx	r--
	A	B	C

Access Control List (ACL)

Alice	rwx	r--	r--
Bob	r--	r--	rwx
Charlie	r--	rwx	r--
	A	B	C

Access Control List

Capability List

Alice	rwx	r--	r--
Bob	r--	r--	rwx
Charlie	r--	rwx	r--
	A	B	C

Access Control Matrix, ACL, and Capability List

When a user tries to access a resource, the operating system can check the Access Control List of the resource to determine whether the user should be granted access.

When a user process wants to access a resource, the operating system can check the capability list to determine whether the process should be granted access.

Operating Systems

Subject Code : 20A05402T

UNIT – 5 Protection and System Security

Implementation of the Access Matrix

- Methods Of Implementing The Access Matrix
  - Global Table
  - Access Lists for Objects
  - Capability Lists for Domains
  - A Lock–Key Mechanism

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

1. Global Table

- The simplest implementation of the access matrix is a global table.
- It consisting of a set of ordered triples <domain, object, rights-set>.
- Whenever an operation *M* is executed on an object *Oj* within domain *Di*, the global table is searched for a
- triple <*Di* , *Oj* , *Rk*>, with $M \in Rk$.
- If this triple is found, the operation is allowed to continue; otherwise, an exception (or error) condition is raised.

Drawbacks of Global Table

- The table is usually large and thus cannot be kept in main memory, so additional I/O is needed.
- Virtual memory techniques are often used for managing this table.
- It is difficult to take advantage of special groupings of objects or domains.
- For example, if everyone can read a particular object, this object must have a separate entry in every domain.

2. Access Lists for Objects

- Each column in the access matrix can be implemented as an access list for one object, the empty entries can be discarded.
- The resulting list for each object consists of ordered pairs <domain, rights-set>,
- which define all domains with a nonempty set of access rights for that object.
- This approach can be extended easily to define a list, and a ***default*** set of access rights.

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

3. Capability Lists for Domains

- A **capability list** for a domain is a list of objects together with the operations allowed on those objects.
- An object is represented by its physical name or address, called a **capability**.
- To execute operation *M* on object *Oj*, the process executes the operation *M*, specifying the capability (or pointer) for object *Oj* as a parameter.
- The **possession** of the capability means that access is allowed.

4. A Lock–Key Mechanism

- The **lock–key scheme** is a compromise between access lists and capability lists.
- Each object has a list of unique bit patterns, called **locks**.
- Each domain has a list of unique bit patterns, called **keys**.
- A process executing in a domain can access an object, only if that domain has a key, that matches one of the locks of the object.
- As with capability lists, the list of keys for a domain must be managed by the operating system on behalf of the domain.
- Users are not allowed to examine or modify the list of keys (or locks) directly.

Difference between Private key and Public key

Cryptography is the science of secret writing with the intention of keeping the data secret. Cryptography is classified into symmetric cryptography, asymmetric cryptography, and hashing.

Private Key:

In the Private key, the same key (secret key) is used for encryption and decryption. In this key is symmetric because the only key is copied or shared by another party to decrypt the cipher text. It is faster than public-key cryptography.

Public Key:

In a Public key, two keys are used one key is used for encryption and another key is used for decryption. One key (public key) is used to encrypt the plain text to convert it into cipher text and another key (private key) is used by the receiver to decrypt the cipher text to read the message. Now, we see the difference between them:

Difference between Private Key and Public Key:

S.NO	Private Key	Public Key
1.	The private key is faster than the public key.	It is slower than a private key.
2.	In this, the same key (secret key) and algorithm are used to encrypt and decrypt the message.	In public-key cryptography, two keys are used, one key is used for encryption, and the other is used for decryption.
3.	In private key cryptography, the key is kept a secret.	In public-key cryptography, one of the two keys is kept a secret.
4.	The private key is **Symmetrical** because there is only one key that is called a secret key.	The public key is **Asymmetrical** because there are two types of keys: private and public keys.
5.	In this cryptography, the sender and receiver need to share the same key.	In this cryptography, the sender and receiver do not need to share the same key.
6.	In this cryptography, the key is private.	In this cryptography, the public key can be public and a private key is private.
7.	It is an efficient technology.	It is an inefficient technology.
8.	It is used for large amounts of text.	It is used for only short messages.
9.	There is the possibility of losing the key that renders the systems void.	There is less possibility of key loss, as the key is held publicly.
10.	The private key is to be shared between two parties.	The public key can be used by anyone.
11.	The Performance testing checks the reliability, scalability, and speed of the system.	The Load testing checks the sustainability of the system.
12.	The private key is used in algorithms such as AES 128, AES 192 and AES 256.	The public key is used in algorithms such as RSA, DSA, etc.
13.	The private key is kept secret.	The public key is widely distributed.
14.	It is used to protect disk drives and other data storage devices.	It is used to secure web sessions and emails.
15.	The recipient's private key decrypts the message.	The recipient's public key encrypts the message.
16.	If the private key is the locking key, then the system can be used to verify documents sent by the holder of the private key.	If the public key is the locking key, then it can be used to send private communication.

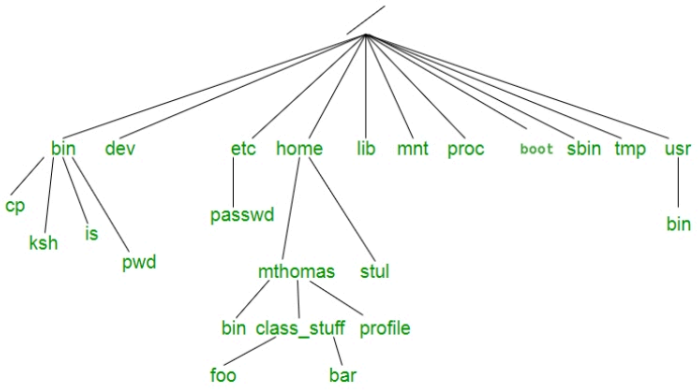
Article Tags : Computer Networks | Difference Between | GATE CS

Recommended Articles

1. Difference between Private and Public IP addresses
2. Difference between Public and Private in C++ with Example
3. Difference Between Public Cloud and Private Cloud
4. Know your public and private IP addresses
5. Public vs Private Access Modifiers in Java
6. Public vs Protected vs Package vs Private Access Modifier in Java
7. Public Key Encryption

Unix File System

Unix file system is a logical method of **organizing and storing** large amounts of information in a way that makes it easy to manage. A file is a smallest unit in which the information is stored. Unix file system has several important features. 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 file system. Files in Unix System are organized into multi-level hierarchy structure known as a directory tree. At the very top of the file system is a directory called "root" which is represented by a "/". All other files are "descendants" of root.

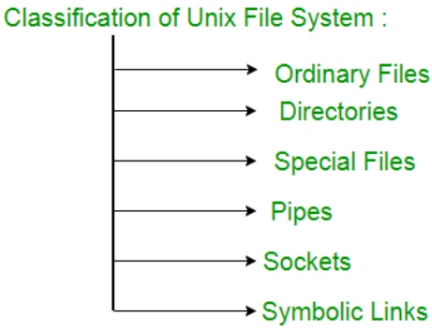


Directories or Files and their description –

- **/ :** The slash / character alone denotes the root of the filesystem tree.
- **/bin :** Stands for "binaries" and contains certain fundamental utilities, such as ls or cp, which are generally needed by all users.
- **/boot :** Contains all the files that are required for successful booting process.
- **/dev :** Stands for "devices". Contains file representations of peripheral devices and pseudo-devices.
- **/etc :** Contains system-wide configuration files and system databases. Originally also contained "dangerous maintenance utilities" such as init,but these have typically been moved to /sbin or elsewhere.
- **/home :** Contains the home directories for the users.
- **/lib :** Contains system libraries, and some critical files such as kernel modules or device drivers.
- **/media :** Default mount point for removable devices, such as USB sticks, media players, etc.
- **/mnt :** Stands for "mount". Contains filesystem mount points. These are used, for example, if the system uses multiple hard disks or hard disk partitions. It is also often used for remote (network) filesystems, CD-ROM/DVD drives, and so on.
- **/proc :** procfs virtual filesystem showing information about processes as files.
- **/root :** The home directory for the superuser "root" – that is, the system administrator. This account's home directory is usually on the initial filesystem, and hence not in /home (which may be a mount point for another filesystem) in case specific maintenance needs to be performed, during which other filesystems are not available. Such a case could occur, for example, if a hard disk drive suffers physical failures and cannot be properly mounted.
- **/tmp :** A place for temporary files. Many systems clear this directory upon startup; it might have tmpfs mounted atop it, in which case its contents do not survive a reboot, or it might be explicitly cleared by a startup script at boot time.
- **/usr :** Originally the directory holding user home directories,its use has changed. It now holds executables, libraries, and shared resources that are not system critical, like the X Window System, KDE, Perl, etc. However, on some Unix systems, some user accounts may still have a home directory that is a direct subdirectory of /usr, such as the default as in Minix. (on modern systems, these user accounts are often related to server or system use, and not directly used by a person).
- **/usr/bin :** This directory stores all binary programs distributed with the operating system not residing in /bin, /sbin or (rarely) /etc.

reboots.

Types of Unix files – The UNIX files system contains several different types of files :



1. Ordinary files – An ordinary file is a file on the system that contains data, text, or program instructions.

- Used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.
- Always located within/under a directory file.
- Do not contain other files.
- In long-format output of ls -l, this type of file is specified by the "-" symbol.

2. Directories – Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders. A directory file contains an entry for every file and subdirectory that it houses. If you have 10 files in a directory, there will be 10 entries in the directory. Each entry has two components. (1) The Filename (2) A unique identification number for the file or directory (called the inode number)

- Branching points in the hierarchical tree.
- Used to organize groups of files.
- May contain ordinary files, special files or other directories.
- Never contain "real" information which you would work with (such as text). Basically, just used for organizing files.
- All files are descendants of the root directory, (named /) located at the top of the tree.

In long-format output of ls –l , this type of file is specified by the "d" symbol. **3. Special Files** – Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Output (I/O) operations. **Device or special files** are used for device Input/Output(I/O) on UNIX and Linux systems. They appear in a file system just like an ordinary file or a directory. On UNIX systems there are two flavors of special files for each device, character special files and block special files :

- When a character special file is used for device Input/Output(I/O), data is transferred one character at a time. This type of access is called raw device access.
- When a block special file is used for device Input/Output(I/O), data is transferred in large fixed-size blocks. This type of access is called block device access.

For terminal devices, it's one character at a time. For disk devices though, raw access means reading or writing in whole chunks of data – blocks, which are native to your disk.

- In long-format output of ls -l, character special files are marked by the "c" symbol.
- In long-format output of ls -l, block special files are marked by the "b" symbol.

4. Pipes – UNIX allows you to link commands together using a pipe. The pipe acts a temporary file which only exists to hold data from one command until it is read by another.A Unix pipe provides a one-way flow of data.The output or result of the first command sequence is used as the input to the second command sequence. To make a pipe, put a vertical bar (|) on the command line between two commands.For example:

More Detail

A **Shell** provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

Shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

Shell Prompt

The prompt, **$**, which is called the **command prompt**, is issued by the shell. While the prompt is displayed, you can type a command.

Shell reads your input after you press **Enter**. It determines the command you want executed by looking at the first word of your input. A word is an unbroken set of characters. Spaces and tabs separate words.

A **Shell** provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

Shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

Shell Prompt

The prompt, **$**, which is called the **command prompt**, is issued by the shell. While the prompt is displayed, you can type a command.

Shell reads your input after you press **Enter**. It determines the command you want executed by looking at the first word of your input. A word is an unbroken set of characters. Spaces and tabs separate words.

Following is a simple example of the **date** command, which displays the current date and time −

```
$date
Thu Jun 25 08:30:19 MST 2009
```

You can customize your command prompt using the environment variable PS1 explained in the Environment tutorial.

Shell Types

In Unix, there are two major types of shells −

- **Bourne shell** − If you are using a Bourne-type shell, the **$** character is the default prompt.
- **C shell** − If you are using a C-type shell, the % character is the default prompt.

The Bourne Shell has the following subcategories −

- 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 original Unix shell was written in the mid-1970s by Stephen R. Bourne while he was at the AT&T Bell Labs in New Jersey.

Bourne shell was the first shell to appear on Unix systems, thus it is referred to as "the shell".

Unix System Calls

System calls in Unix are used for file system control, process control, interprocess communication etc. Access to the Unix kernel is only available through these system calls. Generally, system calls are similar to function calls, the only difference is that they remove the control from the user process.

There are around 80 system calls in the Unix interface currently. Details about some of the important ones are given as follows -

System Call	Description
access()	This checks if a calling process has access to the required file
chdir()	The chdir command changes the current directory of the system
chmod()	The mode of a file can be changed using this command
chown()	This changes the ownership of a particular file
kill()	This system call sends kill signal to one or more processes
link()	A new file name is linked to an existing file using link system call.

system calls. In general, system calls are available as assembly language instructions. They are also included in the manuals used by the assembly level programmers.

Unix System Calls

System calls in Unix are used for file system control, process control, interprocess communication etc. Access to the Unix kernel is only available through these system calls. Generally, system calls are similar to function calls, the only difference is that they remove the control from the user process.

There are around 80 system calls in the Unix interface currently. Details about some of the important ones are given as follows -

System Call	Description
access()	This checks if a calling process has access to the required file
chdir()	The chdir command changes the current directory of the system
chmod()	The mode of a file can be changed using this command
chown()	This changes the ownership of a particular file
kill()	This system call sends kill signal to one or more processes
link()	A new file name is linked to an existing file using link system call.
open()	This opens a file for the reading or writing process
pause()	The pause call suspends a file until a particular signal occurs.
stime()	This system call sets the correct time.
times()	Gets the parent and child process times
alarm()	The alarm system call sets the alarm clock of a process
fork()	A new process is created using this command
chroot()	This changes the root directory of a file.
exit()	The exit system call is used to exit a process.

Windows System Calls

System calls in Windows are used for file system control, process control, interprocess communication, main memory management, I/O device handling, security etc. The programs interact with the Windows operating system using the system calls. Since system calls are the only way to access the kernel, all the programs requiring resources must use system calls.

Details about some of the important system calls in Windows are given as follows -

System Call	Description
CreateProcess()	A new process is created using this command
ExitProcess()	This system call is used to exit a process.
CreateFile()	A file is created or opened using this system call.
ReadFile()	Data is read from the file using this system call.

into several categories. The functions in each category and their purposes are listed in Standard I/O Functions.

Standard I/O Functions

Function	Purpose
Control Functions	control basic access to files
`fopen+`	opens a file
`afopen*+`	opens a file with system-dependent options
`freopen+`	reopens a file
`afreopen*+`	reopens a file with system-dependent options
`tmpfile`	creates and opens a temporary file
`tmpnam`	generates a unique filename
`fflush`	writes any buffered output data
`afflush+`	forces any buffered output data to be written immediately
`fclose+`	closes a file
`setbuf+`	changes stream buffering
`setvbuf+`	changes stream buffering
Character I/O Functions	read or write single characters
`fgetc`	reads a character
`getc`	reads a character (macro version)
`ungetc`	pushes back a previously read character
`getchar`	reads a character from `stdin`
`fputc`	writes a character
`putc`	writes a character (macro version)
`putchar`	writes a character to `stdout`
String I/O Functions	read or write character strings
`fgets`	reads a line into a string
`gets`	reads a line from `stdin` into a string
`fputs`	writes a string
`puts`	writes a line to `stdout`
Array I/O Functions	read or write arrays or objects of any data type
`fread`	reads on or more data elements
`fwrite`	writes one or more data elements
Record I/O Functions	read or write entire funtions
`afread*`	reads a record
`afread0*`	reads a record (possibly length 0)
`afreadh*`	reads the initial part of a record
`afwrite*`	writes a record
`afwrite0*`	writes a record (possibly length 0)
`afwriteh*`	writes the initial part of a record
Formatted I/O Functions	easily read or write formatted data
`fprintf`	writes one or more formatted items
`printf`	writes one or more formatted items to `stdout`
`sprintf`	formats items into a string
`snprintf*`	formats items into a string (with maximum length)
`fscanf`	reads one or more formatted items
`scanf`	reads one or more formatted items from `stdin`
`sscanf`	obtains formatted data from a string
`vfprintf`	writes formatted data to a file
`vprintf`	writes formatted data to a standard output string
`vsprintf`	writes formatted data to a string
`vsnprintf`	writes formatted data to a string (with maximum length)
File Positioning Functions	interrogate and change the file position
`fseek`	positions a file
`fsetpos`	positions a file
`rewind`	positions a file to the first byte
`ftell`	returns current file position for `fseek`
`fgetpos`	returns current file position for `fsetpos`

A shell script is a computer program designed to be run by the Unix/Linux shell which could be one of the following:

- The Bourne Shell
- The C Shell
- The Korn Shell
- The GNU Bourne-Again Shell

A shell is a command-line interpreter and typical operations performed by shell scripts include file manipulation, program execution, and printing text.

Extended Shell Scripts

Shell scripts have several required constructs that tell the shell environment what to do and when to do it. Of course, most scripts are more complex than the above one.

The shell is, after all, a real programming language, complete with variables, control structures, and so forth. No matter how complicated a script gets, it is still just a list of commands executed sequentially.

The following script uses the **read** command which takes the input from the keyboard and assigns it as the value of the variable PERSON and finally prints it on STDOUT.

```
#!/bin/sh

# Author : Zara Ali
# Copyright (c) Tutorialspoint.com
# Script follows here:

echo "What is your name?"
read PERSON
echo "Hello, $PERSON"
```

Here is a sample run of the script −

```
$./test.sh
What is your name?
Zara Ali
Hello, Zara Ali
$
```

Subsequent part of this tutorial will cover Unix/Linux Shell Scripting in detail.

Previous Page Print Page Next Page

Filters

1- cat :- Show line by line
2- head :- Display 1st n lines.
3- tail :- " last n lines
4- sort :-
5- uniq :- removes duplicates lines
6- wc :- no of lines, words and characters in data
7- grep :- search a particular information from a text file
8- tac
9- sed :- stream editor.
10 nl :- number of lines in text data