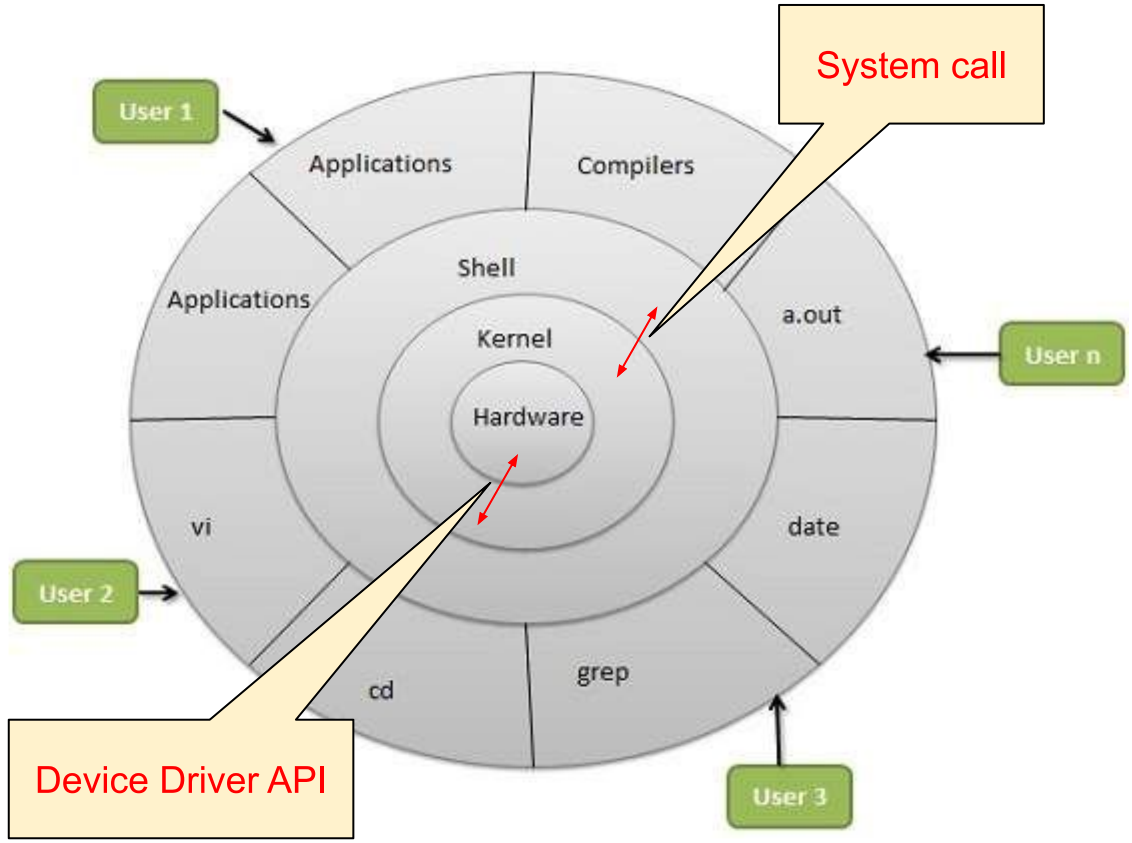
Linux Shell Tutorial

Subhrendu Chattopadhyay
https://shorturl.at/filsF
https://github.com/subhrendu1987/AOS

Types of Shell

- Bourne Shell (sh)
- C Shell (csh)
- TENEX C Shell (tcsh)
- KornShell (ksh)
- Debian Almquist Shell (dash)
- **Bourne-again SHell**



List of Important Shell Commands

- **File Operations**
  a. ls: List contents of a directory.
  b. pwd: Print the current directory.
  c. cd: Change directory.
  d. cp: Copy files or directories.
  e. mv: Move or rename files or directories.
  f. rm: Remove files or directories.
  g. touch: Create an empty file.
  h. find: Search for files or directories.
- **Information Commands**
  a. man: Display the manual page for a command.
  b. info: Display info pages.
  c. whatis: Display one-line descriptions.
- **File Viewing**
  a. cat: Display the contents of a file.
  b. less: View file contents with scrolling capability.
  c. head: Display the first part of a file.
  d. tail: Display the last part of a file.
  e. grep: Search within a file.
- **File Permissions & Ownership**
  a. chmod: Change file permissions.
  b. chown: Change file owner.
  c. chgrp: Change group ownership.
- **Pipe, Redirection, Interprocess Communication**

1. File Operations : Example

```
$ ls
$ pwd
$ mkdir
$ cd
$ cp
$ mv
$ rm
$ touch
$ find
```

2. Information Commands : Example

```
$ man ls
LS(1)                User Commands                LS(1)
NAME
        ls - list directory contents
SYNOPSIS
        ls [OPTION]... [FILE]...
DESCRIPTION
        List  information about the FILEs (the current directory by default).
Sort entries alphabetically if none of
        -cftuvSUX nor --sort is specified.
        Mandatory arguments to long options are mandatory for short options
too.
        -a, --all
        do not ignore entries starting with .
…
```

```
$ info ls
10.1 'ls': List directory contents
==================================

The 'ls' program lists information about files (of any type, including
directories).  Options and file arguments can be intermixed arbitrarily,
as usual.
…
```

```
$ whatis ls
ls (1)          - list directory contents
```

3.1 File Viewing : Example

```
$ cat <filename>
$ less <filename>
$ head <filename>
$ tail <filename>
$ grep <filename>
```

3.2 Redirection: Example

```
$ cat /var/log/auth.log > selected.txt
$ cat /var/log/auth.log.1 >> selected.txt
$ ls /abc > text.txt; cat text.txt
$ ls /abc 2> text.txt; cat text.txt
$ <CMD> &> output_and_error.log
$ (ls /; ls /abc) > all.log 2>&1 ; cat all.log
```

stdout (Standard Output): This is the default output stream where a program writes its output data. It's represented as 1 (though you usually don't need to explicitly specify it).

stderr (Standard Error): This is where a program writes its error messages. It's represented as 2.

3.1 File Viewing : Example

```
$ cat <filename>
$ less <filename>
$ head <filename>
$ tail <filename>
$ grep <filename>
```

3.2 Redirection: Example

```
$ cat /var/log/auth.log > selected.txt
...
```

3.3 Pipe: Example

```
$ echo "Hello, World!" | grep "World"   # Un-named pipe
$ mkfifo my_pipe                         # Named pipe
$ echo "Hello, World!" > my_pipe &
$ cat < my_pipe
```

4.1 Compilation: Example

```
$ mkdir sample; cd sample
```

```
$ nano util.h
#ifndef UTIL_H
#define UTIL_H

/* Function to add two integers */
int add(int a, int b);

/* Function to subtract two integers */
int subtract(int a, int b);

#endif // UTIL_H
```

```
$ nano util.c
#include "util.h"

int add(int a, int b) {
        return a + b;
}

int subtract(int a, int b) {
        return a - b;
}
```

```
$ nano main.c
#include <stdio.h>
#include "util.h"

int main() {
        int x = 5, y = 3;
        int sum, diff;
        sum = add(x, y);
        diff = subtract(x, y);
        printf("Sum: %d\n", sum);
        printf("Difference: %d\n", diff);
        return 0;
}
```

```
$ gcc main.c util.c -o my_program
```

4.2 Compilation with Make : Example

```
$ nano Makefile
# This is a comment
# Variables
CXX = gcc                # GCC compiler
CXXFLAGS = -Wall -g      # C++ flags
# Targets
all: my_program
my_program: main.o util.o
	$(CXX) $(CXXFLAGS) -o my_program main.o util.o
main.o: main.c util.h
	$(CXX) $(CXXFLAGS) -c main.c
utility.o: util.c util.h
	$(CXX) $(CXXFLAGS) -c util.c

clean:
	rm -f *.o my_program
```

Variables: Variables are defined using the = symbol. Once defined, you can use them in your rules by wrapping their names in $().

Targets: Targets define how to produce one or more output files from a set of input files. In the example, ***all*** is a target that depends on ***my_program***. ***my_program*** is a target that depends on ***main.o*** and ***util.o***.

Commands: Commands are specified after the dependencies for a target. They must be indented by a ***tab***, *not spaces*.

Clean: The clean target is a conventional target in makefiles, which is used to clean up any files that can be regenerated

5. File Permissions & Ownership : Example

```
$ ls -al <file/folder name>
```

-rwxrwxr-x <Hardlinks> <Owner> <Group> <Size> <Mod Time> <file/dir name>

- The 1st character indicates the type of file: -: Regular file; d: Directory; l: Symbolic link
- Next 9 characters represent permissions

"rwx" → Read, Write, Execute

Partitioned into 3 sections

- Owner permission : "rwx------"
- Group permission : "---rwx---"
- Others permission : "------rwx"

Octal representation

```
rwx  → 111     → octal 7
rw-  → 110     → octal 6
r-x  → 101     → octal 5
r--  → 100     → octal 4  …
```

File permission

```
rwxr-xr-x   111 101 101     755
rw-r--r--   110 100 100     644
rwx------   111 000 000     700
```

```
$ chmod <Mode> <file/dir name>
$ chown <owner>:<group> <filename>
$ chgrp <group> <filename>
```

Diff and Patch: Example

```
$ diff -u main.c main_v2.c > diff.patch
$ patch main.c < diff.patch     # Apply
$ patch -R main.c < diff.patch  # Revert
```

```
$ diff main.c main_v2.c
8c8
<     diff = subtract(x, y);
---
>     //diff = subtract(x, y);
$ diff -u main.c main_v2.c
--- main.c    2023-08-29 09:34:50.651132095 +0530
+++ main_v2.c    2023-08-30 07:49:45.264289946 +0530
@@ -5,7 +5,7 @@
     int x = 5, y = 3;
     int sum, diff;
     sum = add(x, y);
-    diff = subtract(x, y);
+    //diff = subtract(x, y);
     printf("Sum: %d\n", sum);
     printf("Difference: %d\n", diff);
     return 0;
```

Find & Grep: Example

```
#Find all .txt files in the current directory and its subdirectories
$ find . -name "*.txt"
Find all directories named data in the current directory and its
subdirectories
$ find . -type d -name "data"
Find all regular files larger than 100 MB in the /home/user directory
$ find /home/user -type f -size +100M
Find all empty directories
$ find . -type d -empty
$ grep "<Search String>" <filename>
Case insensitive search
$ grep -i "<Search String>" <filename>
```

```
#Display the line numbers of the matches using the -n option
$ grep -n "<Search String>" <filename>
Use the -v option to display lines that do NOT match the pattern
$ grep -v "<Search String>" <filename>
```

Shell Script: Example

```
$nano greet.sh
```

```
#!/bin/bash
# This is a comment: Script to greet the user
# Ask the user for their name
read -p "What's your name? " user_name
# Print a personalized greeting
echo "Hello, $user_name!"
# Print the current date and time
echo "Current date and time: $(date)"
# End of script
```

```
$ chmod +x greet.sh
$ ./greet.sh # or bash greet.sh
```

Shell Script: Example

```
$nano loopExample.sh
```

```
#!/bin/bash
#Basic numeric range:
echo -e "\033[33;44m Basic numeric range: \033[0m"
for i in {1..5}
do
    echo $i
done
```

Shell Script: Example

```
$nano loopExample.sh
```

```
#!/bin/bash
…
##############################################
#Using a variable for the range:
echo -e "\033[33;44m Using a variable for the range: \033[0m"
end=5
for i in $(seq 1 $end)
do
      echo $i
done
…
# Try other examples from the git repo
```

https://github.com/subhrendu1987/AOS/blob/main/ShellTutorial/loopExample.sh

Shell Script: Example

```
$nano loopExample.sh
```

```
#!/bin/bash
…
##################################################
#Looping over an array:
echo -e "\033[33;44m Looping over an array: \033[0m"
fruits=("apple" "banana" "cherry")
for fruit in "${fruits[@]}"
do
	echo $fruit
done
```

List of Important Shell Commands

- Text Processing
  a. wc: Count words, lines, and characters.
  b. awk: Text processing tool.
  c. sed: Stream editor for text manipulation.
  d. sort: Sort lines in text files.
  e. cut: Remove sections from lines of files.
- Networking
  a. ping: Check connectivity to a host.
  b. netstat: Network statistics.
  c. ifconfig or ip: Display or configure network interfaces.
  d. ssh: Secure shell for remote login.
  e. scp: Secure copy files between hosts.
- Disk Usage
  a. df: Disk space usage of file systems.
  b. du: Estimate file and directory space usage.
- Archiving and Compression
  a. tar: Create or extract tar archives.
  b. gzip: Compress files.
  c. gunzip: Decompress files.
  d. zip: Create zip archives.
  e. unzip: Extract files from zip archives.
- Process Management
  a. ps: Display active processes.
  b. top: Display system processes in real-time.
  c. kill: Terminate processes.
  d. bg: Run processes in the background.
  e. fg: Bring a background process to the foreground.
- System Information
  a. uname: Display system information.
  b. hostname: Show or set the system's host name.
  c. free: Display memory usage.
  d. uptime: Show system uptime and load.
- Environment & Variables
  a. env: Display environment variables.
  b. export: Set environment variables.
  c. set: Display or set shell options.
  d. echo: Display a message or variable value.
- Shell Scripting
  a. #!: Shebang to indicate script interpreter.
  b. $?: Exit status of the last executed command.
  c. $#: Number of arguments passed to the script.
  d. $*: All arguments passed to the script.
  e. $$: Process ID of the script.