

Linux Systems

Getting started with setting up an Embedded platform

Team Emertxe



Introduction



Introduction

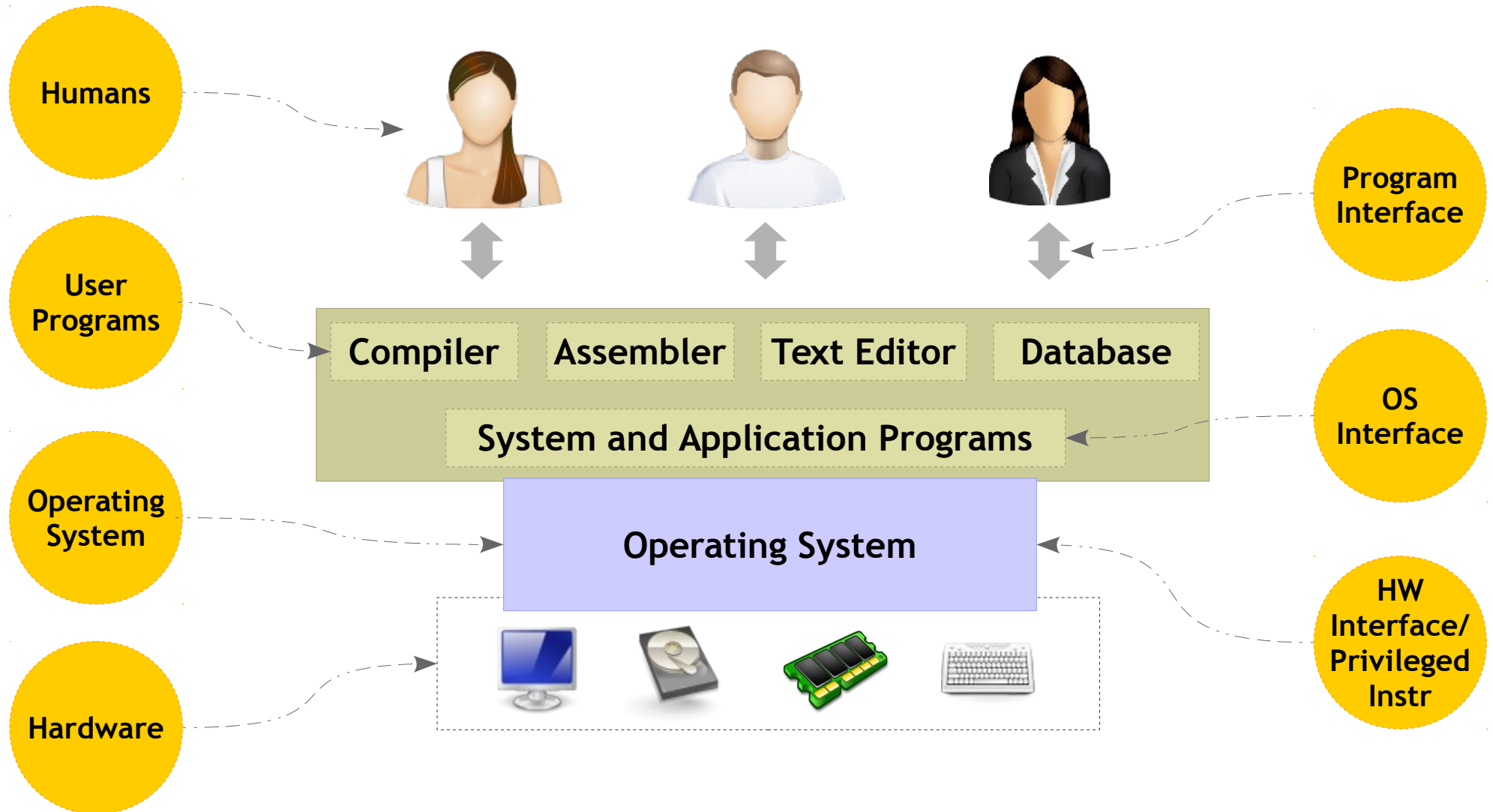
Let us ponder ...



- What exactly is an Operating System (OS)?
- Why do we need OS?
- How would the OS would look like?
- Is it possible for a team of us (in the room) to create an OS of our own?
- Is it necessary to have an OS running in a Embedded System?
- Will the OS ever stop at all?

Introduction

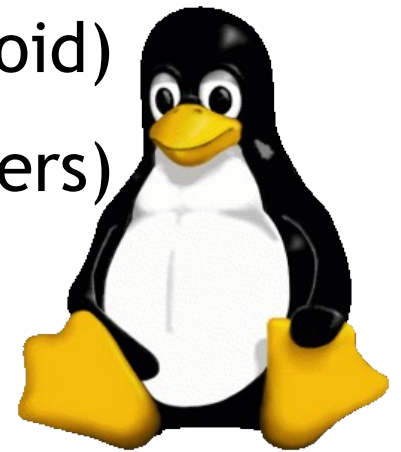
Operating System



Introduction

What is Linux?

- Linux is a free and open source operating system that is causing a revolution in the computer world
- Originally created by Linus Torvalds with the assistance of developers called community
- This operating system in only a few short years is beginning to dominate markets worldwide
- Today right from hand-held devices (ex: Android) to high end systems (ex: Stock exchange servers) use Linux

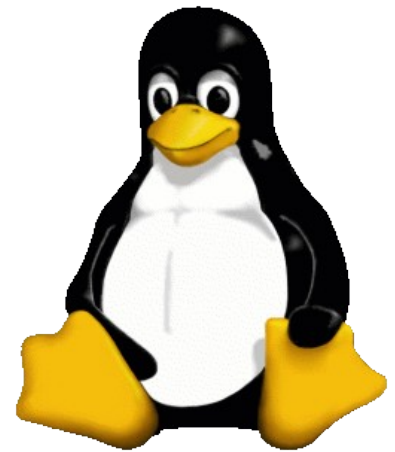


Introduction

Why use Linux?

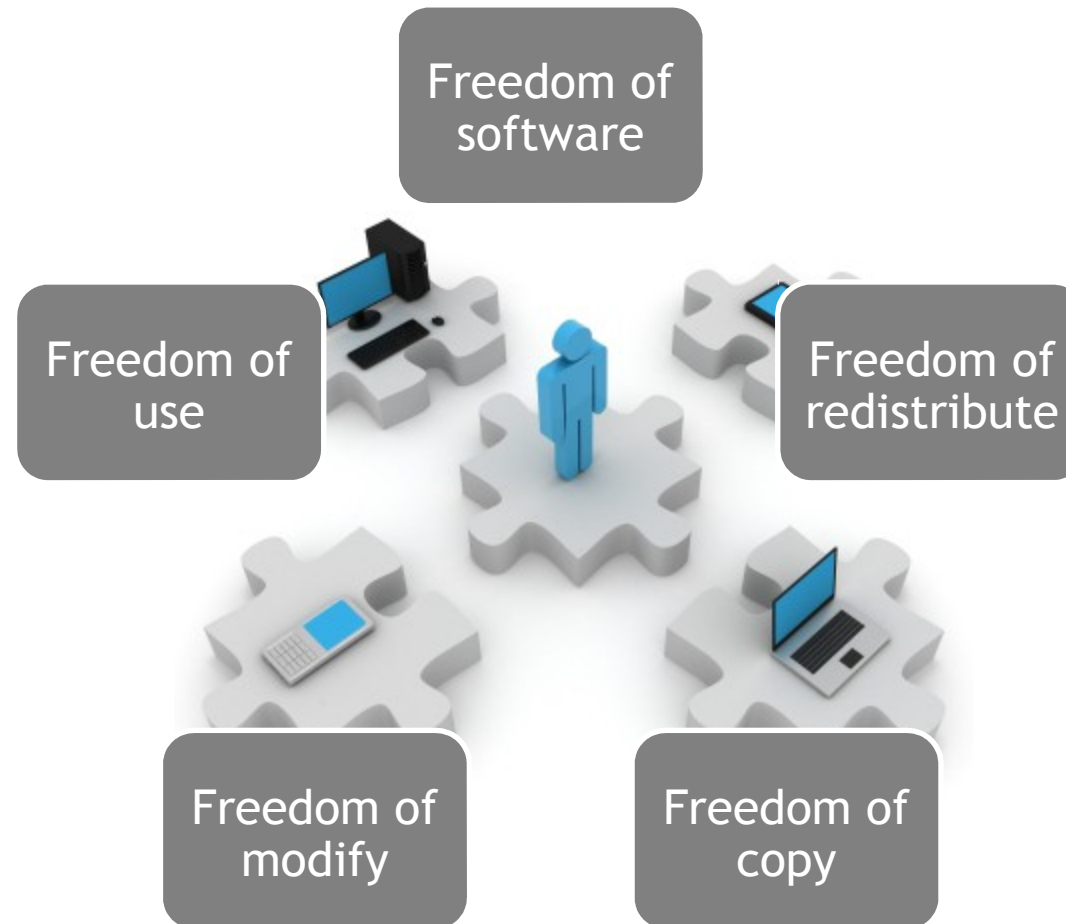
- Free & Open Source -GPL license, no cost
- Reliability -Build systems with 99.999% upstream
- Secure -Monolithic kernel offering high security
- Scalability -From mobile phone to stock market servers

The word 'Free' in Open Source should be interpreted as in 'Freedom' not as 'Free Beer'. This also explains the spirit of creating Open Source software.



Introduction

What is Open Source?



Introduction

Open Source - How it all started?

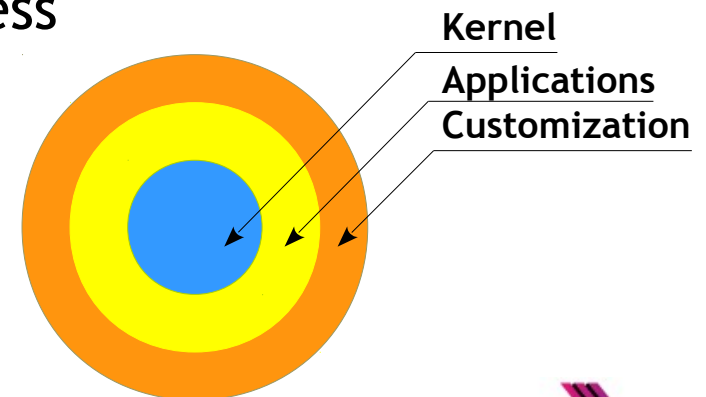
- With GNU (GNU is not UNIX)
- Richard Stallman made the initial announcement in 1983, Free Software Foundation (FSF) got formed during 1984
- Volunteer driven GNU started developing multiple projects, but making it as an operating system was always a challenge
- During 1991 a Finnish Engineer Linus Torvalds developed core OS functionality, called it as “Linux Kernel”
- Linux Kernel got licensed under GPL, which laid strong platform for the success of Open Source
- Rest is history!



Introduction

Open Source - How it evolved?

- Multiple Linux distributions started emerging around the Kernel
- Some applications became platform independent
- Community driven software development started picking up
- Initially seen as a “geek-phenomenon”, eventually turned out to be an engineering marvel
- Centered around Internet
- Building a business around open source started becoming viable
- Redhat set the initial trend in the OS business



Introduction

Open Source - Where it stands now?

OS

CANONICAL



ANDROID

Novell

Databases

EnterpriseDB®
The Enterprise PostgreSQL Company



VoltDB

Server/Cloud

openNMS



OPSCODE

Enterprise



SUGARCRM.

WIKI™

Consumer



LibreOffice®

Education



docebo®

moodle

CMS



AUTOMATTIC



eCommerce



opencart

Introduction

Open Source vs Freeware



| OSS | Freeware |
|--|---|
| <ul style="list-style-type: none">✓ Users have the right to access & modify the source codes✓ In case original programmer disappeared, users & developer group of the S/W usually keep its support to the S/W.✓ OSS usually has the strong users & developers group that manage and maintain the project | <ul style="list-style-type: none">✓ Freeware is usually distributed in a form of binary at 'Free of Charge', but does not open source codes itself.✓ Developer of freeware could abandon development at any time and then final version will be the last version of the freeware. No enhancements will be made by others.✓ Possibility of changing its licensing policy |

Introduction

GPL



- Basic rights under the GPL - access to source code, right to make derivative works
- Reciprocity/Copy-left
- Purpose is to increase amount of publicly available software and ensure compatibility
- Licensees have right to modify, use or distribute software, and to access the source code

Introduction

GPL - Issues



- Linking to GPL programs
- No explicit patent grant
- Does not discuss trademark rights
- Does not discuss duration
- Silent on sub-licensing
- Relies exclusively on license law, not contract

Introduction

Linux Properties

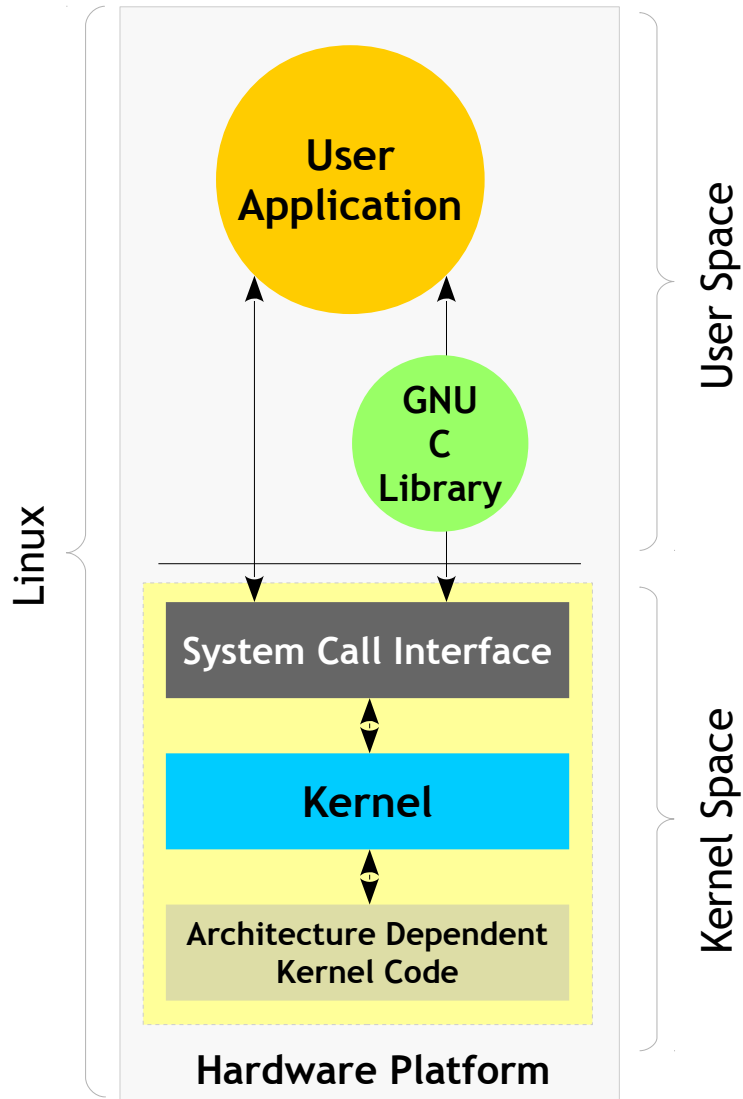


What has made Linux so popular to scale from mobile devices to powering 90% of world's super computer? Here are the key properties of Linux

- **Multitasking**
 - Ability to handle multiple tasks across single / multiple processors
- **Multi-user**
 - Have got users with different level of privileges for secured access
- **Protected Memory**
 - Clear distinction called 'user-space' and 'kernel' space thereby having protected memory access. This makes Linux Super secure comparing with other operating systems
- **Hierarchical File System**
 - Well organized file system that handles various types of files. This also makes handling various inputs very simple

Introduction

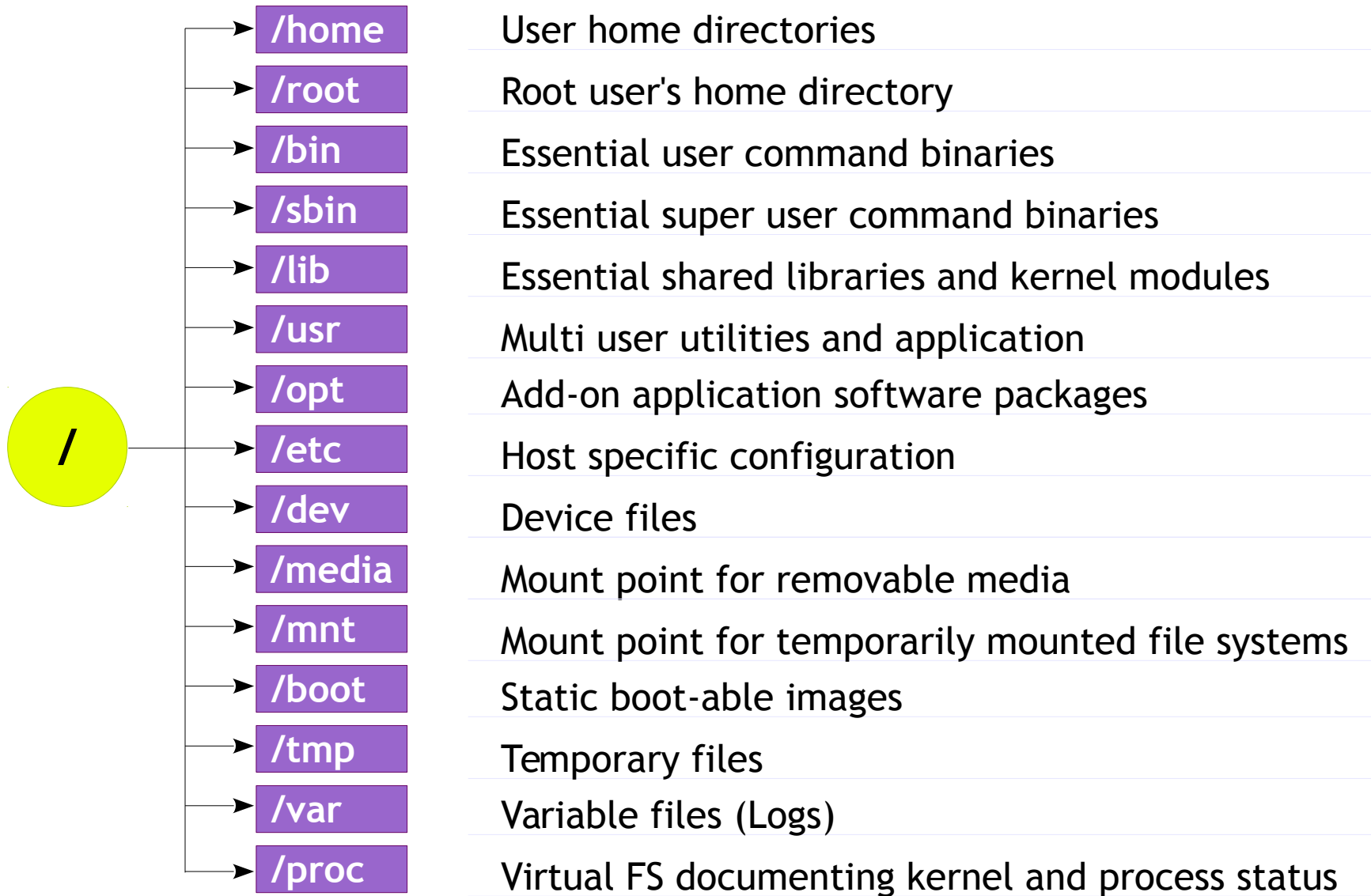
Linux Components



- **Hardware Controllers:** This subsystem is comprised of all the possible physical devices in a Linux installation - CPU, memory hardware, hard disks
- **Linux Kernel:** The kernel abstracts and mediates access to the hardware resources, including the CPU. A kernel is the core of the operating system
- **O/S Services:** These are services that are typically considered part of the operating system (e.g. windowing system, command shell)
- **User Applications:** The set of applications in use on a particular Linux system (e.g. web browser)

Introduction

Linux Directory Structure



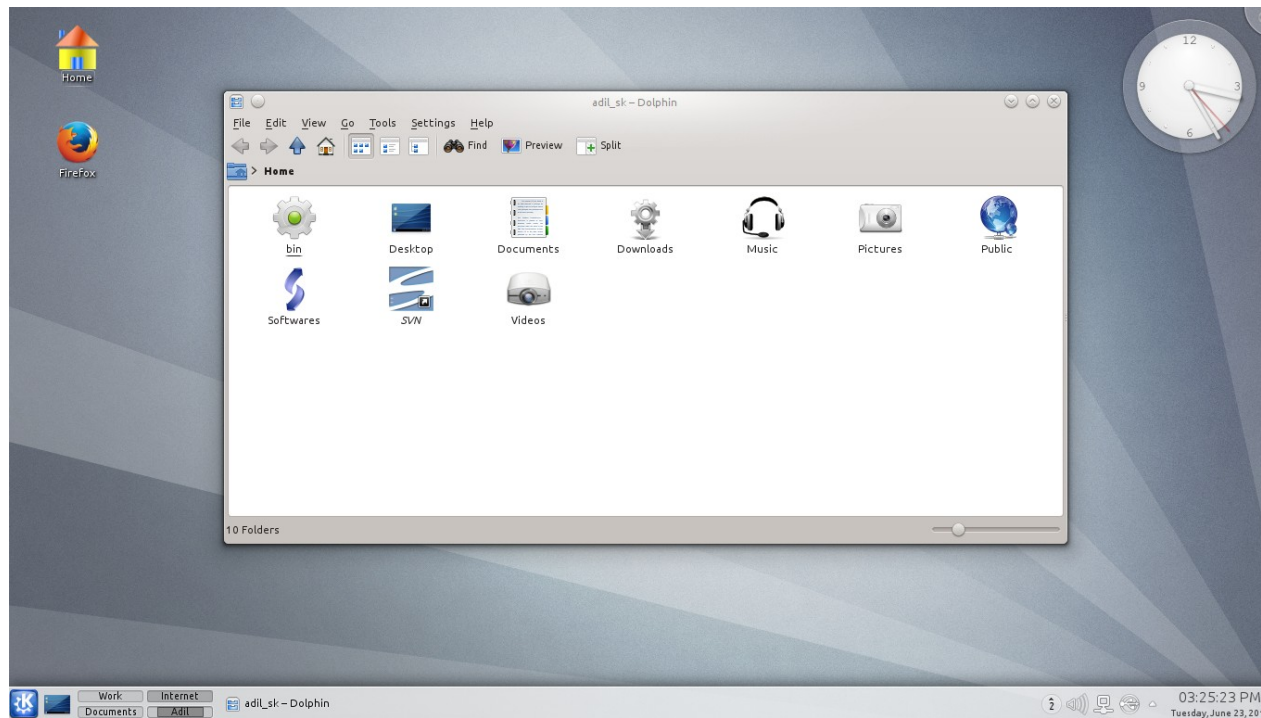
User Interfaces



User Interfaces

GUI

- In graphical mode the user will be given a GUI using which he / she will be able to use the system using mouse
- Similar to windows based system that exist in other operating systems like MS Windows & Apple MAC OS



User Interfaces

CLI



- Textual mode used to execute requested commands

```
adil_sk: bash - Konsole
File Edit View Bookmarks Settings Help
user@user-dt:~] ls
bin Desktop Documents Downloads Music Pictures Public Softwares SVN Videos
user@user-dt:~] ls -l
total 36
drwxrwxr-x 2 adil_sk adil_sk 4096 Oct 17 2014 bin
drwxr-xr-x 2 adil_sk adil_sk 4096 Nov 27 2014 Desktop
drwxr-xr-x 10 adil_sk adil_sk 4096 May 18 19:54 Documents
drwxr-xr-x 13 adil_sk adil_sk 4096 Apr 16 14:51 Downloads
drwxr-xr-x 2 adil_sk adil_sk 4096 Sep 26 2011 Music
drwxr-xr-x 9 adil_sk adil_sk 4096 Jun 23 15:34 Pictures
drwxr-xr-x 2 adil_sk adil_sk 4096 Sep 26 2011 Public
drwxr-xr-x 5 adil_sk adil_sk 4096 May 14 14:00 Softwares
lrwxrwxrwx 1 adil_sk adil_sk 15 Sep 23 2011 SVN -> /mnt/Data1/SVN/
drwxr-xr-x 6 adil_sk adil_sk 4096 Jul 8 2013 Videos
user@user-dt:~]
```

Our focus is to be in the CLI mode by executing various commands by invoking shells. We will also create programs using this environment called 'Shell scripts'

User Interfaces

The Shell - Introduction



- Shell is an application, works as a command interpreter
- Gets a command from user, gets it executed from OS
- Gives a programming environment to write scripts using interpreted language
- It has been inherited from UNIX operating system, which was predecessor to Linux

User Interfaces

The Shel - Types



- Login
 - Starts after a successful login
 - It is executed under user ID during login process
 - It picks up user specific configuration and loads them
- Non Login
 - A Non login shell is started by a program without a login
 - In this case, the program just passes the name of the shell executable
 - For example, for a Bash shell it will be simply bash
 - Following are examples of Non-login shells:
 - sh
 - bash
 - ksh
 - csh

User Interfaces

The Shel - Types

- Try the following on your command prompt

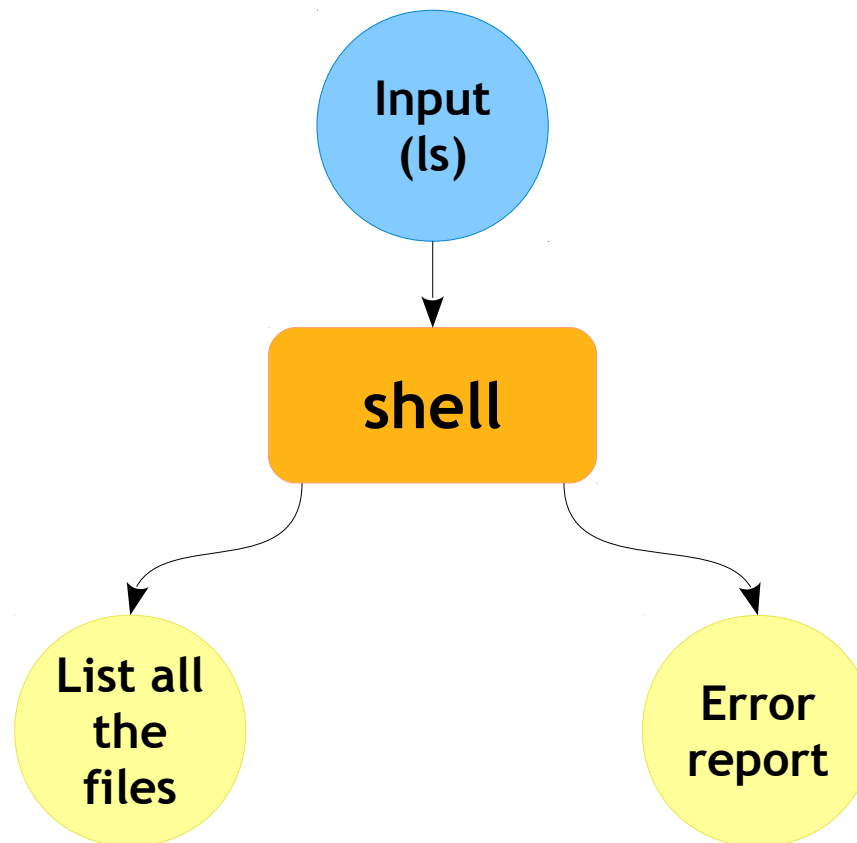
```
$ cat /etc/shells
```

```
$ echo $0
```

User Interfaces

The Shell - Invocation

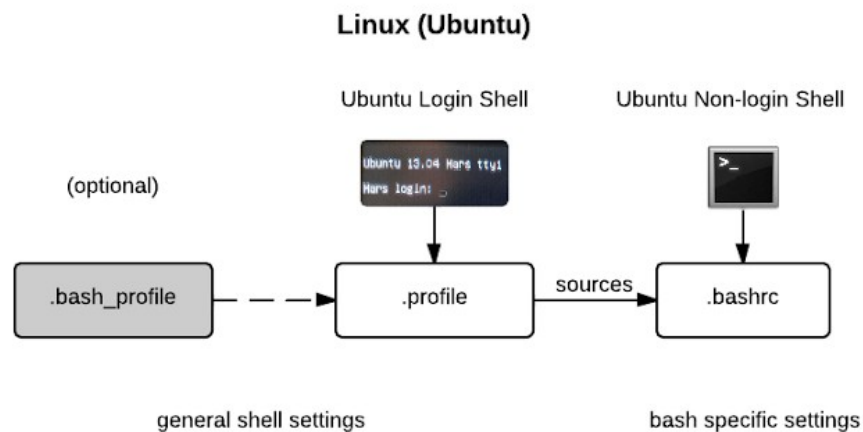
- The main task of a shell is providing a user environment



User Interfaces

The Shell - bash

- Bash - The command interpreter
 - .bash_profile (Login Shell, During login)
 - .bashrc (Non Login Shell, New instance)
 - .bash_logout (Login Shell, Logout)
 - .bash_history (Non Login Shell, Command history)



User Interfaces

The Shell - bash



- Hands on:
 - Enter `ls -a` in your home directory
 - Display contents of all files mentioned above

User Interfaces

The Shell - Environmental Variables



- Login-shell's responsibility is to set the non-login shell and it will set the environment variables
- Environment variables are set for every shell and generally at login time
- Environmental variables are set by the system.
- Environmental variables hold special values. For instance
`$ echo $SHELL`
- Environmental variables are defined in `/etc/profile`, `/etc/profile.d/` and `~/.bash_profile`.
- When a login shell exits, bash reads `~/.bash_logout`

User Interfaces

The Shell - Environmental Variables



- **env** - lists shell environment variable/value pairs

| Variable | Description |
|-----------------|--|
| SHELL | Describes the shell that will be interpreting user commands |
| TERM | This specifies the type of terminal to emulate when running the shell |
| USER | The current logged in user |
| PWD | The current working directory |
| OLDPWD | The previous working directory |
| MAIL | The path to the current user's mailbox |
| PATH | A list of directories that the system will check when looking for commands |
| HOME | The current user's home directory |
| HOSTNAME | The hostname of the computer |
| PS1 | The primary command prompt definition |

Basic Shell Commands

Basic Shell Commands

Types



- **An executable program** like all those files can have in /usr/bin.
- **A command built into the shell itself.** bash provides a number of commands internally called shell built-ins The cd command, for example, is a shell built-in
- **A shell function.** These are miniature shell scripts incorporated into the environment.
- **An alias.** Commands that you can define yourselves, built from other commands.
- To know the type, try
`$ type <command>`

Basic Shell Commands



| Command | Meaning |
|---------|---------------------------------|
| ls | List's all the files |
| pwd | Gives present working directory |
| cd | Change directory |
| man | Gives information about command |
| df | Disk free |
| du | Disk usage |
| which | Shows full path of command |

Path



- Path is the location where a particular file is located in the directory (tree) structure
- It starts with the root ('/') directory and goes into appropriate directory
- The path depends on the reference point from where you take it up:
 - **Absolute Path** : Specifies the location with reference from root directory
 - **Relative Path** : Specifies the location with reference to present working directory (pwd)
- As the name says relative path will vary depending on your pwd

Visual Editor - vi

Visual Editor - vi



- Screen-oriented text editor originally created for the Unix operating system
- The name **vi** is derived from the shortest unambiguous abbreviation for the ex command **visual**
- Improved version is called as **vim**
- To open a file

\$ vi <filename>

or

\$ vim <filename>

Visual Editor - vi



- vi opens a file in command mode to start mode.
- The power of vi comes from its 3 modes
 - **Escape mode** (Command mode)
 - Search mode
 - File mode
 - **Editing mode**
 - Insert mode
 - Append mode
 - Open mode
 - Replace mode
 - **Visual mode**

Visual Editor - vi

Cursor Movement



- You will clearly need to move the cursor around your file. You can move the cursor in command mode.
- vi has many different cursor movement commands. The four basic keys appear below
 - **k** move up one line
 - **h** line move one character to the left
 - **l** line move one character to the right
 - **j** move down one line
- Yes! Arrow keys also do work. But these makes typing faster

Visual Editor - vi

Basic Commands



- Open a file

`$ vi <file_name>`

- How to exit

`:q` -> Close with out saving.

`:wq` -> Close the file with saving.

`:q!` -> Close the file forcefully with out saving

- Already looks too complicated?
- Try by yourself, let us write a C program

Visual Editor - vi

Escape / Command Mode



- In command mode, characters you perform actions like moving the cursor, cutting or copying text, or searching for some particular text
 - Search mode
 - vi can search the entire file for a given string of text. A string is a sequence of characters. vi searches forward with the slash (/) key and string to search. To cancel the search, press ESC. You can search again by typing n (forward) or N (backward). Also, when vi reaches the end of the text, it continues searching from the beginning. This feature is called wrap scan
 - Instead of (/), you may also use question (?). That would have direction reversed
 - Now, try out. Start vi as usual and try a simple search. Type /<string> and press n and N a few times to see where the cursor goes.

Visual Editor - vi

Escape / Command Mode



– File mode

- Changing (Replacing) Text

`:%s/first/sec` - Replaces the first by second every where in the file

`:%s/orange/apple/gc` - For all lines in a file, find string "orange" and replace with string "apple" for each instance on a line

- File Interactions (edit and read)

`:e filename` - open another file without closing the current

`:r filename` - reads file named filename in place

- Editor Settings

`:set all` - display all settings of your session

Visual Editor - vi

Escape / Command Mode - Useful Shortcuts



| Command | Operation |
|---------------|--|
| G | Go to last line of the file |
| gg | Go to first line of the file |
| . | Repeat the previous command |
| Ctrl a | Increment number under the cursor by 1 |
| Ctrl x | Decrements numbers under the cursor by 1 |
| J | Joining the two adjacent lines |
| (n)gg | Move cursor to n th line |

Visual Editor - vi

Editing Mode



| Command | Mode Name | Insertion Point |
|----------|-----------|-----------------------------------|
| a | Append | just after the current character |
| A | Append | end of the current line |
| i | Insert | just before the current character |
| I | Insert | beginning of the current line |
| o | Open | new line below the current line |
| O | Open | new line above the current line |

Visual Editor - vi

Editing Mode - Editing Text



- **Deleting Text** Sometimes you will want to delete some of the text you are editing. To do so, first move the cursor so that it covers the first character of the group you want to delete, then type the desired command from the table below.

| Command | Operation |
|---------|--------------------------------------|
| dd | For deleting a line |
| (n)dd | For deleting a n lines |
| x | To delete a single character |
| D | Delete contents of line after cursor |
| dw | Delete word |
| (n)dw | Delete n words |

Visual Editor - vi

Visual Mode - Editing Text



- Visual Mode
 - Visual mode helps to visually select some text, may be seen as a sub mode of the command mode to switch from the command mode to the visual mode type one of
 - **v** - visual mode
 - **ctrl+v** - Go's to visual block mode.
 - **d** or **y** Delete or Yank selected text
 - **I** or **A** Insert or Append text in all lines (visual block only)

Shell Scripting - Basics



Shell Scripting - Basics

Programming Languages



- There are various types of programming languages, compared on various parameters
- From Embedded system engineer's view it should be seen how close or how much away from the hardware the language is
- Based on that view programming languages can be categorized into three areas:
 - Assembly language (ex: 8051)
 - Middle level language (ex: C)
 - High level / Scripting language (ex: Shell)
- Each programming language offers some benefits with some shortcomings
- Depending on the need of the situation appropriate language needs to be chosen
- This make language selection is a key criteria when it comes to building real time products!

Shell Scripting - Basics

Programming Languages - A Comparison



| Language parameter | Assembly | C | Shell |
|--------------------|----------|--------|--------|
| Speed | High | Medium | Medium |
| Portability | Low | Medium | High |
| Maintainability | Low | Medium | High |
| Size | Low | Medium | Low |
| Easy to learn | Low | Medium | High |

Shell or any scripting language is also called as 'interpreted' language as it doesn't go through compilation phase. This is to keep the language simple as the purpose is different than other languages.

Shell Scripting - Basics

Shell Script



- Any collection of shell commands can be stored in a file, which is then called as shell script
- Programming the scripts is called shell scripting
- Scripts have variables and flow control statements like other programming languages
- Shell script are interpreted, not compiled
- The shell reads commands from the script line by line and searches for those commands on the system

Shell Scripting - Basics

Shell Script - Where to use?

- System Administration
 - Automate tasks
 - Repeated tasks
- Development
 - Allows testing a limited sub-set
 - Testing tools
- Daily usage
 - Simple scripts
 - Reminders, e-mails etc...



Shell Scripting - Basics

Shell Script - Invocation



- Example:

```
$ vi hello.sh
```

and type the following inside it:

```
#!/bin/bash
```

```
echo "Hello World"
```

Then, make the script executable:

```
$ chmod 700 hello.sh
```

```
$ ./hello.sh
```

- First line tells Linux to use BASH interpreter
- Second line prints the “Hello world” into standard I/O

Shell Scripting - Basics

Variables



- Variables are a way of storing information temporarily.

NAME="Emertxe"

X=10

- A couple of conversions we need to follow
 - Variables usually appear in uppercase
 - There is no white space between the variable name and the equal sign
- Variable substitution
- Variable assignment
- Bash variables & special variables

Shell Scripting - Basics

Whitespace & Line-breaks



- Bash shell scripts are very sensitive to whitespace & line-breaks
- Because the “keywords” of this programming language are actually commands evaluated by the shell
- Need to separate arguments with whitespaces
- Likewise a line-break in the middle of a command will mislead the shell into thinking the command is incomplete.
- Example: `x=10; x = 10; x = “ok”; x=”ok”`

Shell Scripting - Basics

Special Characters



| Character | Meaning |
|-----------|--|
| ~ | The current user's home directory |
| \$ | used to access a variable (e.g. : \$HOME) |
| & | used to put a command in the background |
| * | wildcard, matching zero or more characters (e.g. : ls doc_*) |
| ? | wildcard, matching exactly one character (e.g.: ls doc_?) |
| \${#} | No of arguments passed to shell script |
| \${@} | Value of all arguments passed |
| \$0 | contains the name of the script user executed |

Shell Scripting - Basics

Quotes



- Using single quotes causes the variable name to be used literally, and no substitution will take place.

```
$var='test string'
```

```
$newvar='Value of var is $var'
```

```
echo $newvar
```

- Using double quotes to show a string of characters will allow any variables in the quotes to be resolved

```
$var="test string"
```

```
$newvar="Value of var is $var"
```

```
echo $newvar
```

Shell Scripting - Basics

Expressions

- `expr`: Evaluates simple math on the command line calculator.
- `bc`: An arbitrary precision calculator language
- Available operators: `+`, `,`, `/`, `*`, `%`

Example

```
1 #!/bin/bash
2
3 NUM1=5
4 NUM2=3
5
6 ADD=$(( ${NUM1} + ${NUM2} ))
7 SUB=$(( ${NUM1} - ${NUM2} ))
8 MUL=$(( ${NUM1} * ${NUM2} ))
9 DIV=$(( ${NUM1} / ${NUM2} ))
10 MOD=$(( ${NUM1} % ${NUM2} ))
11
12 echo "Addition of two numbers is: ${ADD}"
13 echo "Substraction of two numbers is: ${SUB}"
14 echo "Multiplication of two numbers is: ${MUL}"
15 echo "Division of two numbers is: ${DIV}"
16 echo "Modulum of two numbers is: ${MOD}"
```

Shell Scripting - Basics

Conditions - if else

- The if statement chooses between alternatives each of which may have a complex test
- The simplest form is the if-then statement

Syntax

```
if [ condition ]  
then  
    expression  
else  
    expression  
fi
```

Example

```
1 #!/bin/bash  
2  
3 NUM1=5  
4 NUM2=3  
5  
6 if [ ${NUM1} -gt ${NUM2} ]  
7 then  
8     echo "NUM1 is greater than NUM2"  
9 else  
10    echo "NUM2 is greater than NUM1"  
11 fi
```

Shell Scripting - Basics

Conditions - if else if

- Multiple if blocks can be strung together to make an elaborate set of conditional responses

Syntax

```
if [ condition_a ]
then
    condition_a is true
elif [ condition_b ]
then
    condition_b is true
else
    both false
fi
```

Example

```
1 #!/bin/bash
2
3 NUM1=5
4 NUM2=3
5
6 if [ ${NUM1} -eq ${NUM2} ]
7 then
8     echo "NUM1 is equal to NUM2"
9 elif [ ${NUM1} -gt ${NUM2} ]
10 then
11     echo "NUM1 is greater than NUM2"
12 else
13     echo "NUM1 is less than NUM2"
14 fi
```

Shell Scripting - Intermediate



Shell Scripting - Intermediate

String Tests

- String comparison, Numeric comparison, File operators and logical operators
- Comparison operations are provided below

Example

```
1 #!/bin/bash
2
3 echo "Enter the first string"
4 read STR1
5 echo "Enter the second string"
6 read STR2
7
8 if [ -z ${STR1} ]; then
9     echo "First string is empty"
10 else
11     echo "First string is not empty"
12 fi
13 if [ -n ${STR2} ]; then
14     echo "First string is not empty"
15 else
16     echo "First string is empty"
17 fi
18 if [ ${STR1} = ${STR2} ]; then
19     echo "Both strings are equal"
20 else
21     echo "Both strings are not equal"
22 fi
```

| Operator | Meaning |
|----------|--|
| = | Compare if two strings are equal |
| != | Compare if two strings are not equal |
| -n | Evaluate if string length is greater than zero |
| -z | Evaluate if string length is equal to zero |

Shell Scripting - Intermediate

Numeric Tests



| Operator | Meaning |
|------------|---|
| -eq | Compare if two numbers are equal |
| -ge | Compare if one number is greater than or equal to num |
| -le | Compare if one number is less than or equal to a num |
| -ne | Compare if two numbers are not equal |
| -gt | Compare if one number is greater than another number |
| -lt | Compare if one number is less than another number |

Example

```
1 #!/bin/bash
2
3 NUM1=5
4 NUM2=3
5
6 if [ ${NUM1} -gt ${NUM2} ]
7 then
8     echo "NUM1 is greater than NUM2"
9 else
10    echo "NUM2 is greater than NUM1"
11 fi
```

Shell Scripting - Intermediate

Logical Operators



Example

```
1 #!/bin/bash
2
3 echo "Enter the first number A"
4 read A
5 echo "Enter the second number B "
6 read B
7 echo "Enter the third number C "
8 read C
9
10 if [ ${A} -gt ${B} -a ${A} -gt ${C} ]; then
11     echo "A is the greatest of all"
12 elif [ ${B} -gt ${A} -a ${B} -gt ${C} ]; then
13     echo "B is the greatest of all"
14 elif [ ${C} -gt ${A} -a ${C} -gt ${B} ]; then
15     echo "C is the greatest of all"
16 else
17     echo "Invalid Input"
18 fi
```

| Operator | Meaning |
|----------|--------------------------------------|
| ! | Negate (NOT) a logical expression |
| -a | Logically AND two logical expression |
| -o | Logically OR two logical expressions |

Shell Scripting - Intermediate

Loop - for

- Sequential loop with expressions
- First arithmetic expr EXPR1 is evaluated; EXPR2 evaluated repeatedly until it evaluates to 0; Each time EXPR2 is evaluated to a non-zero value, statements are executed & EXPR3 is evaluated

Syntax

```
for ((expr1; expr2; expr3))  
do  
    Code Block  
done
```

Example

```
1 #!/bin/bash  
2  
3 for ((i=1; i<=5; i++))  
4 do  
5     echo "Loop counter is ${i}"  
6 done
```

Shell Scripting - Intermediate

Loop - while

- The structure is a looping structure. Used to execute a set of commands while a specified condition is true
- The loop terminates as soon as the condition becomes false. If condition never becomes false, loop will never exit
- Any valid conditional expression will work in the while loop.

Syntax

```
while [ condition ]  
do  
    Code Block  
done
```

Example

```
1 #!/bin/bash  
2  
3 LOOP=1  
4  
5 while [ ${LOOP} -le 5 ]  
6 do  
7     echo "Looping : ${LOOP}"  
8     LOOP=$(( ${LOOP} + 1 ))  
9 done
```

Shell Scripting - Intermediate

Case Statements

- The case statement compares the value of the variable (\$var in this case) to one or more values
- Once a match is found, the associated commands are executed and the case statement is terminated
- Used to execute statements based on specific values
- Often used in place of an if statement if there are a large number of conditions.
- Each set of statements must be ended by a pair of semicolon
- *) is used for not matched with list of values

Shell Scripting - Intermediate

Case Statements

Syntax

```
case ${VAR} in
    value_1)
        commands;
        ;;
    value_2)
        commands;
        ;;
    *)
        commands;
        ;;
esac
```

Example

```
1  #!/bin/bash
2
3  echo "Enter a number:"
4  read NUM
5
6  case ${NUM} in
7      1)
8          echo "You entered One"
9          ;;
10     2) echo "You entered Two" ;;
11     *) echo "Obey my orders please"
12         ;;
13 esac
```

Shell Scripting - Intermediate

Functions

- Writing functions can greatly simplify a program
- Improves modularity, readability and maintainability
- However speed will get slowed down
- Arguments are accessed as \$1, \$2, \$3...

Syntax

```
function name()  
{  
    <command>  
    <statements>  
    <expression>  
}
```

Example

```
1  #!/bin/bash  
2  
3  function sum()  
4  {  
5      x=`expr $1 + $2`  
6      echo $x  
7  }  
8  
9  y=`sum 5 3`  
10 echo "The sum is 5 and 3 is $y"  
11 echo "The sum is 6 and 2 is `sum 6 2`"
```


Shell Scripting - Intermediate

Arrays

- An array is a variable containing multiple values may be of same type or of different type
- There is no maximum limit to the size of an array
- Array index starts with zero

Shell Scripting - Intermediate

Arrays

Syntax

```
declare -a array_name=(element1 element2 element3)
```

Here **declare -a** is optional, arrays can be declared without that also

Example

```
1  #!/bin/bash
2
3  declare -a LINUX_DISTROS=('Debian' 'Redhat' 'Ubuntu' 'Suse' 'Fedora');
4
5  echo "Number of elements in the array: ${#LINUX_DISTROS[@]}"
6  echo "Printing elements of array in one shot: ${LINUX_DISTROS[@]}"
7  echo "Printing elements of array in one shot: ${LINUX_DISTROS[*]}"
8  echo "Printing elements of array in using a loop:"
10 for ((i = 0; i < ${#LINUX_DISTROS[@]}; i++))
11 do
12     echo ${LINUX_DISTROS[$i]}
13 done
```

Shell Scripting - Intermediate

Command Line Arguments



- Shell script can accept command-line arguments & options just like other Linux commands
- Within your shell script, you can refer to these arguments as \$1,\$2,\$3,.. & so on.
- Then the command line arguments are executed like
- Read all command line arguments and print them

Shell Scripting - Intermediate

Command Line Arguments



Example

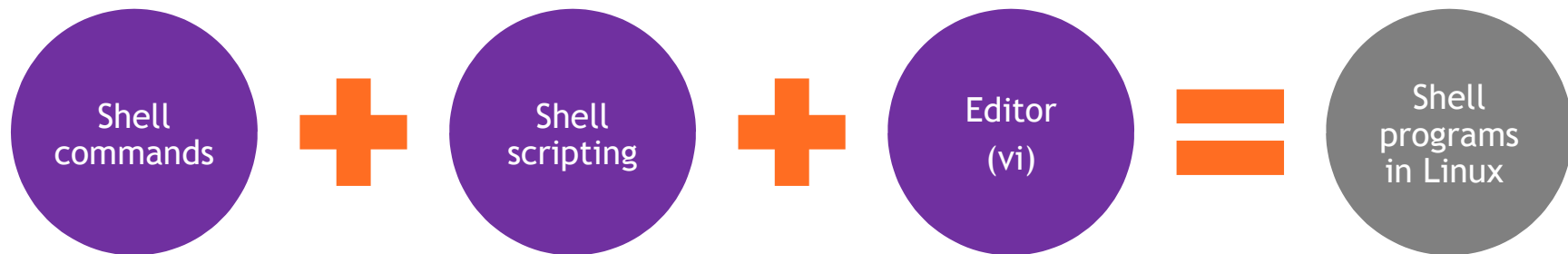
```
1  #!/bin/bash
2
3  if [ $# != 2 ]
4  then
5      echo "Usage: Pass 3 arguments"
6      exit 0
7  fi
8
9  echo "The arguments of the script you passed are:"
10 echo "Total number of arguments you passed are : $# "
11 echo "The name of the script is : $0 "
12 echo "The first argument is : $1 "
13 echo "The second argument is : $2 "
```

Linux Systems - The Bigger Picture



Linux System - A Bigger Picture

How things fit together?

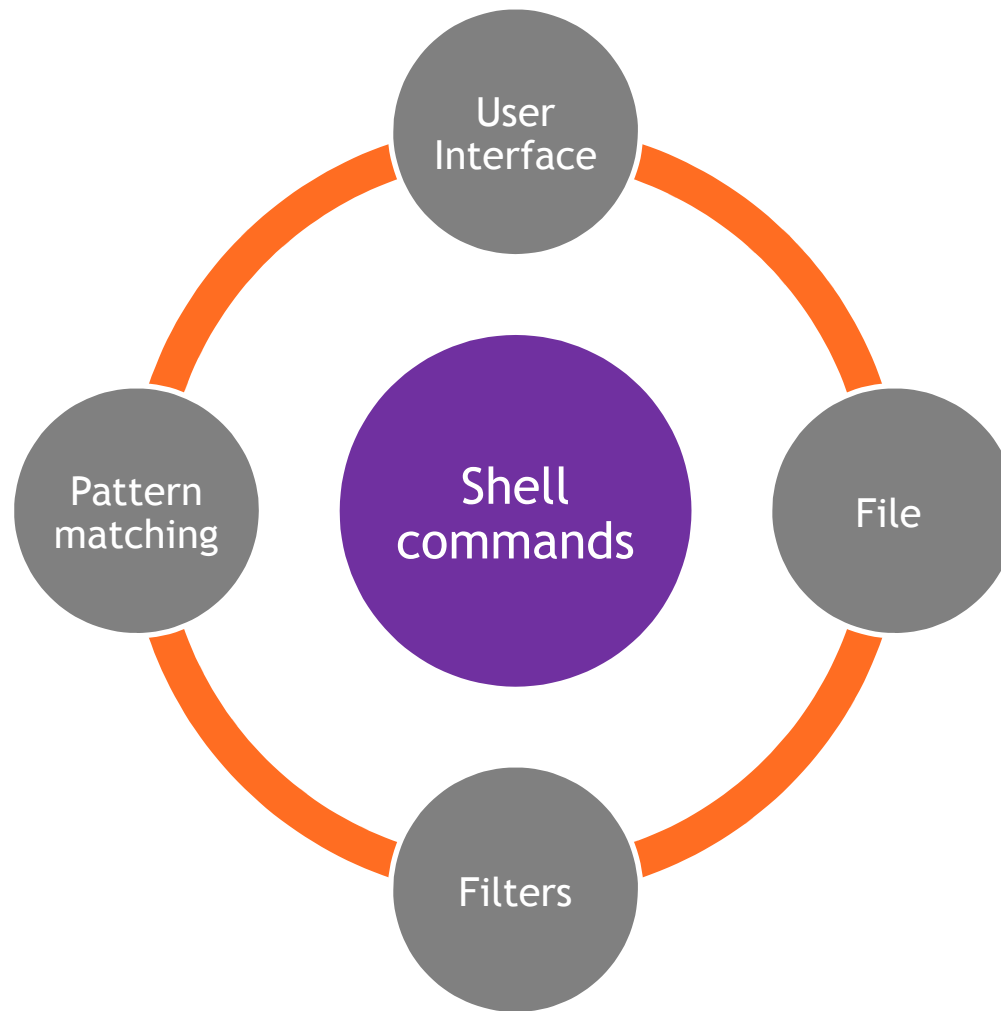


- **Shell commands:** ls, date, whoami etc.
- **Shell scripting:** Operators, Loop, Arrays etc..
- **Editor (vi):** Insertion, commands, visual mode

Now let us learn some more Linux shell commands (advanced) for making our shell scripts more powerful

Linux System - A Bigger Picture

Advanced Shell Commands



Advanced Shell Commands



Shell Commands - Advanced

User Specific Commands



- All Accesses into a Linux System are through a User
- Super user (root) will have higher privileges
- User related Shell commands

| Command | Meaning |
|------------------------------|-----------------------------------|
| <code>useradd</code> | Create user |
| <code>userdel</code> | Delete user |
| <code>su - [username]</code> | Start new shell as different user |
| <code>finger</code> | User information lookup |
| <code>passwd</code> | Change or create user password |
| <code>who</code> | To find out who is logged in |
| <code>whoami</code> | Who are you |

Shell Commands - Advanced

Remote Login and Remote Copy



- ssh is a program for logging into a remote machine and for executing commands on a remote machine.

```
$ ssh username@ipaddress
```

- scp copies files between hosts on a network.

```
$ scp filename username@ipaddress:/path/
```

Shell Commands - Advanced

File Related - Redirection



- Out put redirection (>)

```
$ ls > /tmp/outputfile
```

- Redirecting to append (>>)

```
$ ls -l >> /tmp/outputfile
```

- Redirecting the error (2>)

```
$ ls 2> /tmp/outputfile
```

Shell Commands - Advanced

File Related - Pipe



- A pipe is a form of redirection that is used in Linux operating systems to send the output of one program to another program for further processing.
- A pipe is designated in commands by the vertical bar character

```
$ ls -al /bin | less
```

Shell Commands - Advanced

File Related



- Every thing is viewed as a file in Linux. Even a *Directory* is a file.
- Basic Shell Command Set

| Command | Meaning |
|---|---|
| <code>cp <source> <dest></code> | Copy file from one to another |
| <code>mv <source> <dest></code> | Rename a file |
| <code>rm <file></code> | Remove a file |
| <code>stat</code> | File related statistics (i-node) |
| <code>ln</code> | Linking between files (-s option for soft link) |

Shell Commands - Advanced

File Related



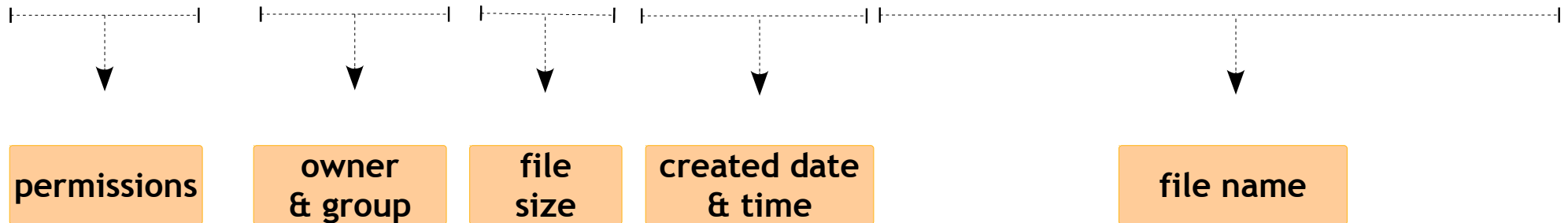
| Command | Meaning |
|-------------------------------------|---|
| <code>mkdir <dir_name></code> | Make directory |
| <code>rmdir <dir_name></code> | Remove a particular directory |
| <code>touch</code> | Change file timestamps |
| <code>wc</code> | Counts the number of lines in a file |
| <code>cat</code> | Display contents of the file in standard output |
| <code>more</code> | Display contents, navigate forward |
| <code>head</code> | Display first 10 lines of the file (-n to change) |
| <code>tail</code> | Display last 10 lines of the file (-n to change) |
| <code>sort</code> | Sort lines of text files |

Shell Commands - Advanced

File Listing



```
user@user:~] ls -l
total 12
drwxrwxr-x  2 user user    4096 Jun 23 16:48 A-Direcory
brw-r--r--  1 root root     7, 0 Jun 23 16:55 block_file
crw-r--r--  1 root root   108, 0 Jun 23 16:49 character_file
lrwxrwxrwx  1 user user     12 Jun 23 16:50 link_to_regular_file -> regular_file
prw-rw-r--  1 user user      0 Jun 23 16:50 named_pipe
-rw-rw-r--  1 user user      0 Jun 23 16:48 regular_file
-rwxrwxr-x  1 user user   7639 Jun 23 16:54 server
srwxrwxr-x  1 user user      0 Jun 23 16:55 server_socket
```



Shell Commands - Advanced

File Listing - Types



| | | | | | |
|--------------------|---|--------------------|---|------|------|
| Directory | ← | d rw-rw-r-x | 2 | user | user |
| Block | ← | b rw-r--r-- | 1 | root | root |
| Character | ← | c rw-r--r-- | 1 | root | root |
| Soft Link | ← | l rw-rw-rw- | 1 | user | user |
| FIFO (sometimes =) | ← | p rw-rw-r-- | 1 | user | user |
| Plain Text | ← | - rw-rw-r-x | 1 | user | user |
| Socket | ← | s rw-rw-r-x | 1 | user | user |

```
user@user:~] ls -l
total 12
d-rwxrwxr-x  2 user user
b-rw-r--r--  1 root root
c-rw-r--r--  1 root root
l-rwxrwxrwx  1 user user
p-rw-rw-r--  1 user user
-rwxrwxr-x  1 user user
s-rwxrwxr-x  1 user user
```


Shell Commands - Advanced

File Listing - Permission



```
user@user:~] ls -l
total 1
-rwxrwxr-x 2 user user
```

Value used to Set

→ Execute

001 - 1

→ Write

010 - 2

→ Read

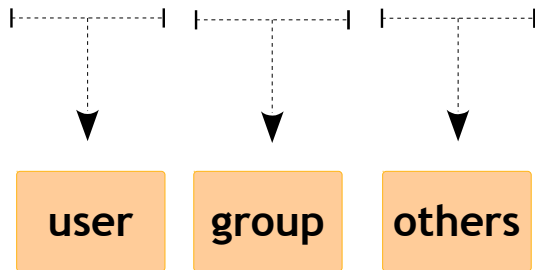
100 - 4

Shell Commands - Advanced

File Listing



```
user@user:~] ls -l  
total 1  
-rwxrwxrwx  2 user user
```



Shell Commands - Advanced

File Permissions



- **chmod** - Change file permissions
- **chown** - Change file owner
- **chmod** [ug+r, 746] file.txt
- **chown -R** user:group [filename | dir]

Shell Commands - Advanced

find



Syntax

```
find <where-to-look> <criteria> <what-to-do>
```

| Find options | Meaning |
|--|---|
| <code>find . -print</code> | Find all files from current directory & sub-directory & print their path |
| <code>find . -name *.sh</code> | Find all files that are having *.sh extension starting from current directory |
| <code>find / -type d -name tech</code> | Search for directories with name 'tech' from the 'root' directory |
| <code>find . -type f -empty</code> | Find all files that are empty in current directory |

Shell Commands - Advanced

cut

Syntax

`cut <option> <file>`

Option '-c' character

Option '-d' delimiter and much more

| cut options | Meaning |
|--|---|
| <code>cut -c3 <file></code> | Outputs third character of every line of <file> |
| <code>cut -c1-3 <file></code> | Outputs the first three characters of every line of <file> |
| <code>cut -d':' -f1 /etc/passwd</code> | Outputs the first field of the file /etc/passwd, where fields are delimited by a colon (':'). The first field of /etc/passwd is the username, so this will output every username in the passwd file |
| <code>cut -d':' -f1,6 /etc/passwd</code> | Output first and sixth fields of /etc/passwd |

Shell Commands - Advanced

split



Syntax

```
split <option> <file> <newfile>
```

Option '-b' bytes

Option '-l' lines and much more

| split options | Meaning |
|--|--|
| <code>split -b <file> <newfile></code> | Split 'b' bytes from 'file' and put them into newfileaa, newfileab, newfileac etc.. |
| <code>split -l <file> <newfile></code> | Split <file> into 'l' number of lines and put them into newfileaa, newfileab, newfileac etc. |

There are three more file related commands as follows:

- **cmp** - Compares two files and stops where difference is found
- **diff** - Reports differences between two files
- **uniq** - Reports/Filters repeated pattern in a file

Let us read through the man pages and understand them (Self-study)

Shell Commands - Advanced

tr



Syntax

```
tr <options> <input>
```

| tr options | Meaning |
|---------------------------|---------------------------------------|
| tr -d <char> | Delete occurrences of given character |
| tr -s <char> | Squeeze repetition characters |
| :upper: | Upper case characters |
| :lower: | Lower case characters |
| :space: | Space character |
| :digit: | Numerical numbers |

Shell Commands - Advanced

tr



Delete all occurrences of character 'h'

```
$ echo hello how are you | tr -d h
```

ello ow are you

Replace all spaces with tabs

```
$ echo "whats up with you guys" | tr [:space:] '\t'
```

whats up with you guys

Remove all numbers from the input

```
$ echo "my age is 99" | tr -d [:digit:]
```

my age is

Replace all lower case with upper case

```
$ tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

hello

HELLO

Shell Commands - Advanced

File Compression



- Compression is needed to conserve the disk space
- When there is a need to send large files as an attachment via the email, it is good practice to compress first
- Compression & Decompression utilities - `gzip` & `gunzip(.gz)`
- The degree of compression depends on
 - The type of the file
 - Its size
 - Compression program used
- Example
 - Html files compress more
 - GIF & JPEG image files compress very less, as they are already in compressed form

Shell Commands - Advanced

File Compression - Flow



- **Recursive compression and de-compression (-r option), will come handy**
- **gzip -r <directory> : To compress whole directory**
- **gunzip -r <directory> : To de-compress whole directory**

Shell Commands - Advanced

File Compression - Example



Compression

| Input | Program | Output |
|-------------|------------------|----------------|
| file | gzip file | file.gz |
| sample.html | gzip sample.html | sample.html.gz |

De-compression

| Input | Program | Output |
|----------------|-----------------------|-------------|
| file.gz | gunzip file.gz | file |
| sample.html.gz | gunzip sample.html.gz | sample.html |

Shell Commands - Advanced

File Archival



- Used for creating disk archive that contains a group of files or an entire directory structure
- An archive file is a collection of files and directories that are stored in one file
- Archive file is not compressed, it uses the same amount of disk space as all the individual files and directories
- In case of compression, compressed file occupies lesser space
- Combination of archival & compression also can be done

- **File archival is achieved using 'tar' with the following commands:**
- **tar -cvf <archive name> <file-names>**
- **tar -xvf <archive name>**

Shell Commands - Advanced

File Archival - Flow



Shell Commands - Advanced

Regular Expressions



- Regular expressions = search (and replace/modify/remove) pattern
- In theoretical computer science regular expressions are called as regex or regexp
- It is a sequence of characters that forms a search pattern using some special characters
- Popular applications in Linux (Vi editor, Grep, Sed, Lex & Yacc etc..) extensively use regular expressions
- Extensively used in compiler design and implementation
- Our idea is to understand them from Linux commands

Shell Commands - Advanced

Regular Expressions



- Each character in a regular expression is either understood to be a meta-character with its special meaning
- Or a regular character with its literal meaning
- Together they form a pattern. Some popular & most frequently used examples are provided below

| Meta-character | Meaning |
|----------------|-------------------------|
| ? | Zero or one occurrence |
| * | Zero or more occurrence |
| + | One or more occurrence |

Shell Commands - Advanced

Pattern Matching - grep



- Get Regular Expression And Print (GREP)
- Grep is pattern matching tool used to search the name input file

Syntax

```
grep <reg-exp> <file>
```

Option

Meaning

```
grep a* <file>
```

Search for lines starting with name 'a' in <file>

```
grep -x <pattern> <file>
```

Exactly match for <pattern> in <file>

```
grep -v <pattern> <file>
```

Print non-matching of <pattern> in <file>

Shell Commands - Advanced

Pattern Matching - sed



- Stream Editor (SED)
- Sed perform basic text transformations on an input stream
- It can be a file, or input from a pipe

Syntax

```
sed <reg-exp> <file>
```

| Option | Meaning |
|--|--|
| <code>sed -n '1,5p' <file></code> | Print line numbers ranging from 1-5 in the given input file |
| <code>sed '1,5d' <file> > out</code> | Delete line number ranging from 1-5 in the given input file and re-direct into another file called out |
| <code>sed 's/<old_string>/<new_string>/' <file></code> | Replace <old_string> with <new_string> in input <file> |

Stay Connected



About us: Emertxe is India's one of the top IT finishing schools & self learning kits provider. Our primary focus is on Embedded with diversification focus on Java, Oracle and Android areas

Emertxe Information Technologies,

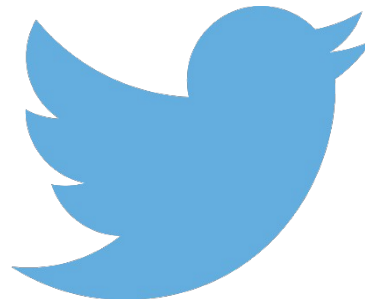
No-1, 9th Cross, 5th Main,
Jayamahal Extension,
Bangalore, Karnataka 560046

T: +91 80 6562 9666

E: training@emertxe.com



<https://www.facebook.com/Emertxe>



<https://twitter.com/EmertxeTweet>



<https://www.slideshare.net/EmertxeSlides>

Thank You