



Introduction to Linux Concepts



- Presented By

--- Siva Kumar Bhuchipalli

<http://hadooptutorial.info/>

Agenda:

- What is Linux ?
- Linux Distributions
- Directory Structure
- Shell
- Linux Commands Syntax
 - ✓ File Handling Commands
 - ✓ Text Processing Commands
 - ✓ System Administration
 - ✓ Advanced Commands
- Text Editors
- Vi Editor
- Pattern Matching
- Shell Scripting
- Environment Variables

What is Linux ?

- FOSS - Free Open Source Software
- Unix-type operating system developed under the GNU General Public License
- Open source
- Popular
- Multi-user, Multitasking, Multiprocessor
- Why is it famous?
 - ✓ Linux Provides Security
 - ✓ Linux is Virus Free
 - ✓ Supports Multiple Hardware Platforms



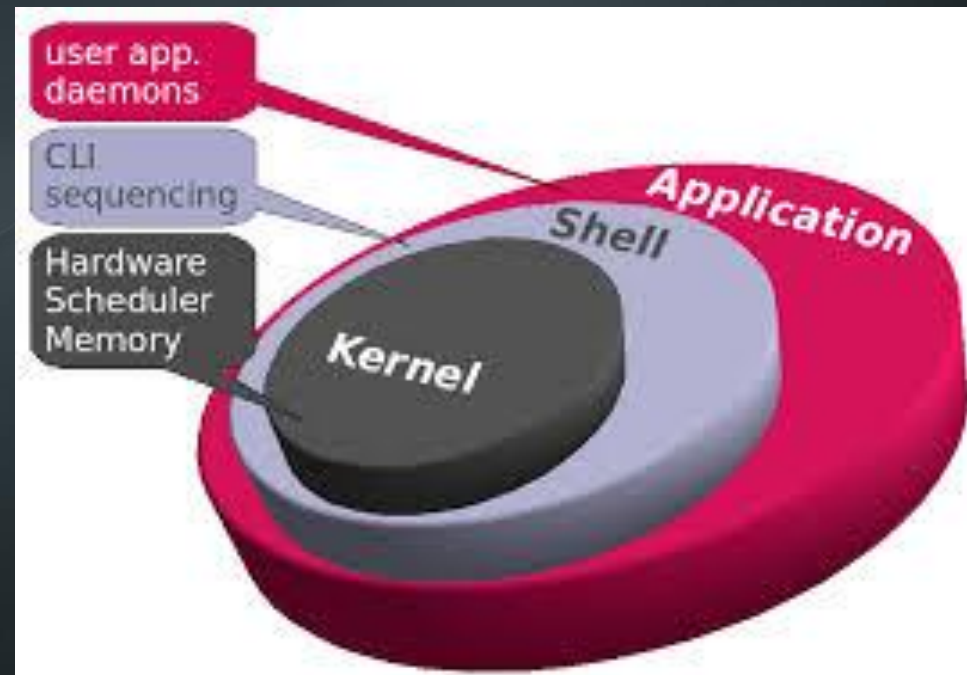
Linux Distributions

- Ubuntu
- CentOS
- Red Hat
- Fedora
- Mandrake
- Debian
- Etc...



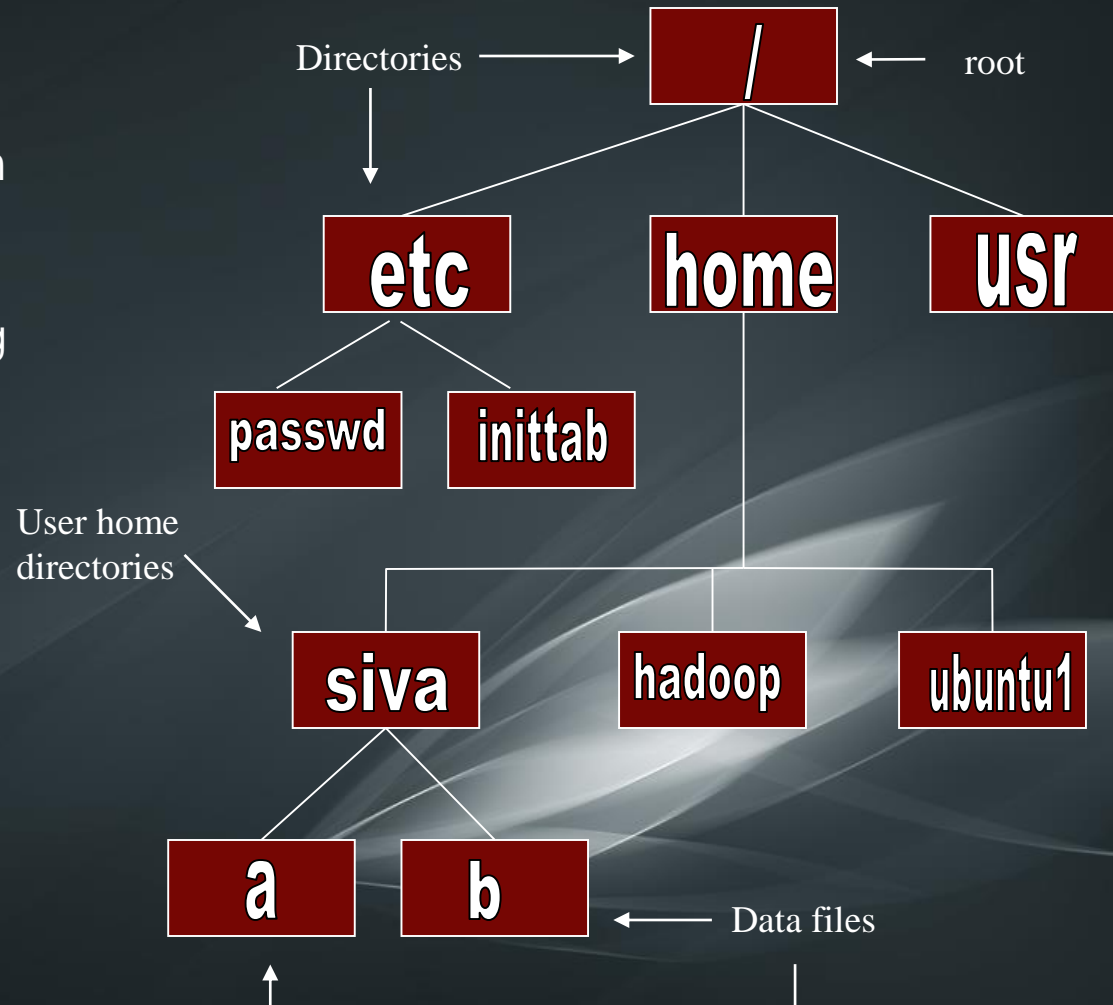
Kernel

- ❑ Core of an operating system
- ❑ Interacts with the hardware
- ❑ First program to get loaded when the system starts and runs till the session gets terminated



LFS Directory Structure

- ❑ Linux files are stored in a single rooted, hierarchical file system
- ❑ Files are stored in directories (folders)
- ❑ If you omit the leading / then path name is relative to the current working directory

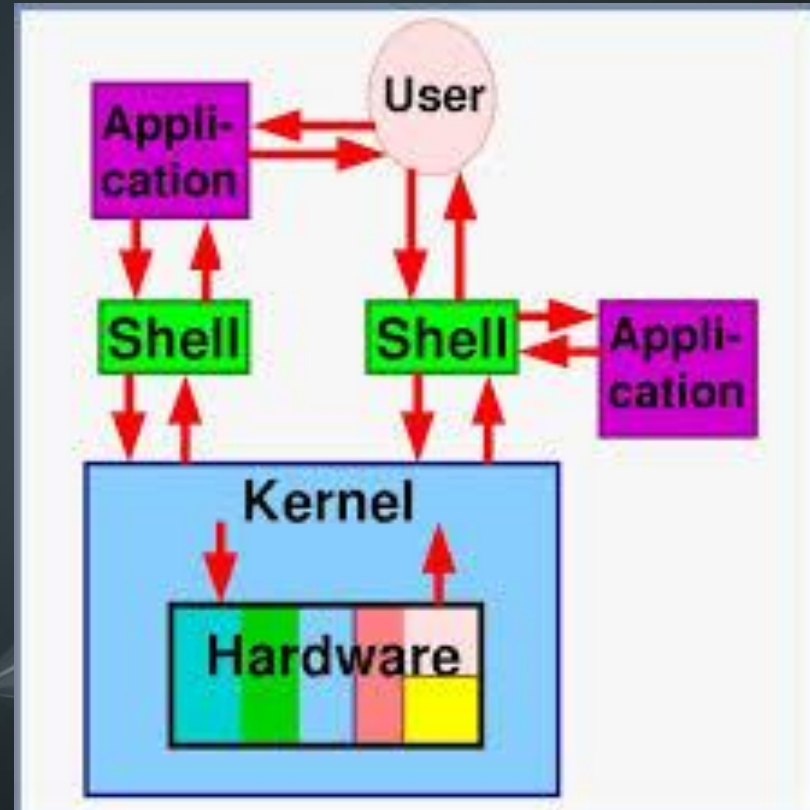


LFS Directory Structure (Contd...)

- ❑ **/bin** System binaries, including the command shell
- ❑ **/boot** Boot-up routines
- ❑ **/dev** Device files for all your peripherals
- ❑ **/etc** System configuration files
- ❑ **/home** User directories
- ❑ **/lib** Shared libraries and modules
- ❑ **/lost+found** Lost-cluster files, recovered from a disk-check
- ❑ **/mnt** Mounted file-systems
- ❑ **/opt** Optional software
- ❑ **/proc** Kernel-processes pseudo file-system
- ❑ **/root** Administrator's home directory
- ❑ **/sbin** System administration binaries
- ❑ **/usr** User-oriented software
- ❑ **/var** Various other files: mail, spooling and logging

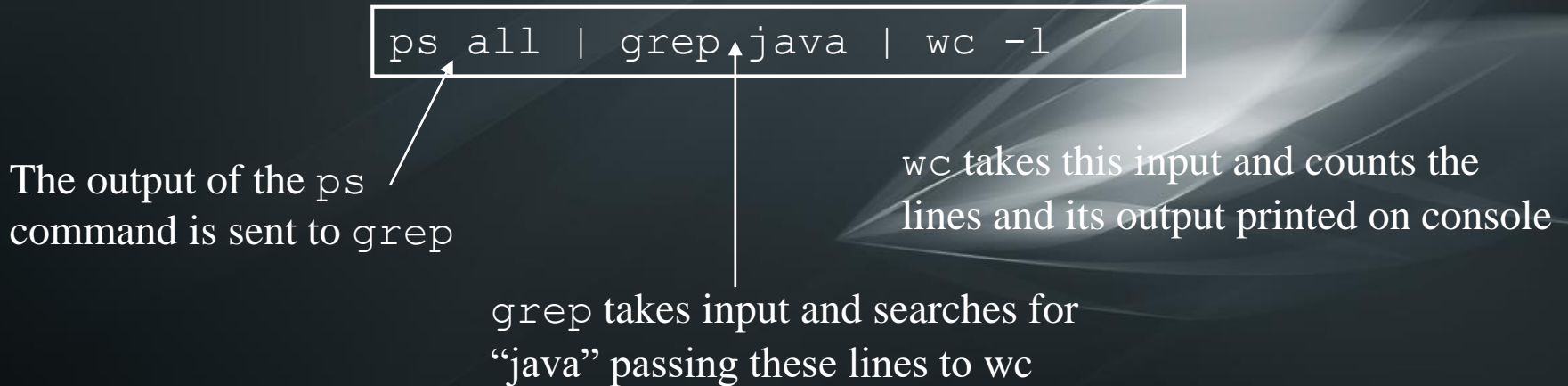
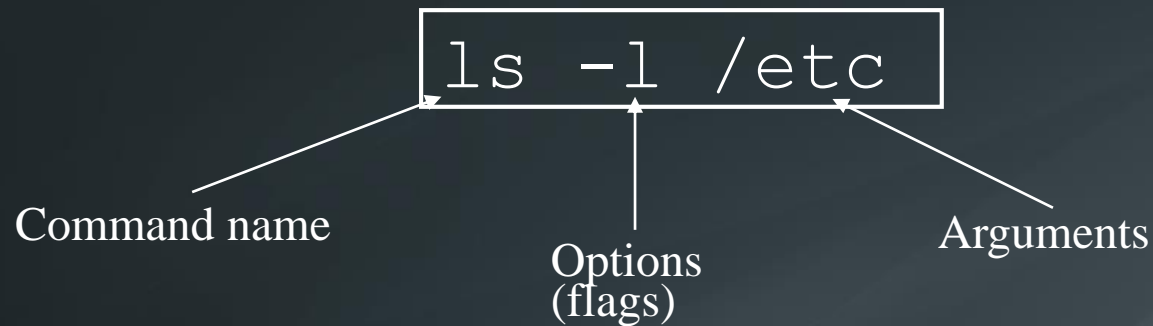
Shell

- ❑ Command Line Interpreter
- ❑ Bridge between kernel and the user
- ❑ Types
 - ✓ SH – Simple Shell
 - ✓ BASH – Bourne Again Shell
 - ✓ KSH – Korne Shell
 - ✓ CSH – C Shell
- ❑ Bash is the default Shell Type



Linux Command Basics

- ❑ To execute a command, type its name and arguments at the command line



Linux Commands Types

- File Handling
- Text Processing
- System Administration
- Process Management
- Archival
- Network
- File Systems
- Advanced Commands



File Handling Commands

- mkdir - creating directory
 - mkdir dirname
- rmdir – removing directory and its contents
 - rmdir dirname
- cd – Change directory
 - cd dirpath
- cp – Copying files
 - cp file1 file2
- mv – Moving or renaming files
 - mv oldfile newfile
- ls – list directory contents
 - ls –[altr] dir_path
 - ls – Lists all files in a directory
 - ls -a – Lists all files (including hidden files)
 - ls -l – Lists files in a directory along with owner information, permission etc
- ln – Creating links between files
 - ln file1 file2

File Handling commands (Contd...)

- find – search for files in a directory hierarchy
 - find . -name "*.java"
- rm – remove files
- history – prints recently used commands
- pwd – prints name of current working directory
- Viewing users, processes
 - who – List all Users
 - who am I – List the current user
 - pstree – displays all processes running in the system in tree format
 - ps – displays processes owned by the current user
- Help commands
 - man, pinfo, info (man <cmd name>)

Text Processing

- cat – concatenate files and print on the standard output
 - cat file1.txt file2.txt
- echo – display a line of text
 - echo “I like Hadoop”
- grep - prints lines matching a pattern
 - grep PATTERN FILE
- wc - prints the number of newlines, words, and bytes in files
 - wc -l → no of lines
 - wc -w → no of words
 - wc -c → no of characters



System Administration

- `chmod` – change file access permissions
 - `chmod 744 calculate.sh`
 - `$chmod u+x file.txt` – Gives execution permission to the owner of the file

Owner/user – Who creates a file

Group – Collection of users

Others – Apart from the user and the users in a group

<u>Permissions</u>	<u>Value</u>
Read	4
Write	2
Execute	1

`-rwxrwxrwx` – All permissions given for a file

`drwxrwxrwx` – All permissions given for a directory

`chmod 777 file` //gives all permission (r,w,x) for a file

- `chown` – change file owner and group
 - `chown siva myfile.txt`



Advanced Commands

- su – change user ID or become super user
 - su siva, su
- passwd – update a user's authentication tokens(s)
- su siva, su
 - passwd
- who – show who is logged on
- df (display filesystem): Displays how much disk space on every mounted partition that is occupied.
 - df -h uses MB and GB instead of blocks
- du (directory usage): Displays how much space a given directory plus all of its subdirectories uses.
- reboot – reboot the system
- poweroff – power off the system
- uname -a: Prints all information about your system
- ctrl+R - search for previously entered commands
- clear: Clear terminal screen
- export: Set an environment variable

Advanced Commands (Contd...)

- head: Output the first part of file
- tail: Output the last part of files
- touch: Change file timestamps
- tr: Translate, squeeze, and/or delete characters
- ifconfig: Configure a network interface
- kill: Stop a process from running
- netstat: Networking information
- scp: Secure copy (remote file copy)
- ssh: Secure Shell client (remote login program)
- wget: Retrieve web pages or files via HTTP, HTTPS or FTP
- which: Search the user's \$path for a program file

Advanced Commands (Contd...)

- `echo `date +%Y-%M-%d``
2015-27-26
- `echo `date +%Y-%M-%D``
2015-27-10/26/15
- `echo `date +%Y``
2015
- `echo `date +%D``
10/26/15
- `echo `date +%d``
26
- `echo `date +"%m_%d_%y_%H%M%S"``
10_26_15_003100

Text Editors

- ☐ Vi
- ☐ Vim
- ☐ nano
- ☐ gEdit
- ☐ kWrite
- ☐ TextPad
- ☐ Emacs
- ☐ And more...

VI Editor

- Popular text editor
- Just type vi <<filename>> at the prompt and hit enter
- A new file will be opened
- Type the contents needed and save
- To save, press the **Esc** Key and then press **:** (colon) wq and then enter
- To quit with out saving **Esc + : + q** and then enter
- Navigation
 - Left - h
 - Down - j
 - Up - k
 - Right - l
 - Top of the screen – H (shift + h) //caps lock will not work
 - Middle of the screen – M (shift + m)
 - Bottom of the screen – L (shift + l)
 - \$ - End Key, 0 – Home Key



Pattern Matching

- grep – GNU Regular Expression Processor
 - Finds the words / patterns matching with the search and displays the line containing the patterns.
 - Search is limited to a file
- ```
grep abc hello.txt
```

Finds the occurrence of abc in hello.txt and displays the line in the screen

grep -i abc hello.txt – Ignores case. Will find Abc, ABC, aBc, aBC etc

grep -c abc hello.txt – Displays the count of matches

grep -n abc hello.txt – Displays the matching lines along with line number

grep -v abc hello.txt – Displays the lines that do not have the pattern abc

^a – Starts with a

a\$ - Ends with a

a\* - a followed by any number of characters

a..b – a separated by 2 characters and then followed by b



# Shell Scripting

- Open a file with extension .sh using any editor
- We can type any number of commands
- Save the file
- Execute the file
  - `sh file.sh`
  - `./file.sh`
- For Loop

```
for ((i=0; i<5; i++))
do
 Body of the loop
done
```
- If else condition

```
if [condn]
then
elif [condn]
then
else
fi
```



# Shell Scripting

- These are the contents of a shell script called display:

- ```
cat display
# This script displays the date, time, username and
# current directory.
echo "Date and time is:"
date
echo
echo "Your username is: `whoami` \\n"
echo "Your current directory is: \\c"
pwd
```

- Comments are represented with beginning # and are not interpreted by the shell.
- The Regular UNIX Commands will be enclosed in the backquotes (`) around the command whoami.
- The \\n is an option of the echo command that tells the shell to add an extra carriage return at the end of the line. The \\c tells the shell to stay on the same line

Passing arguments to the shell

- Arguments are passed from the command line into a shell program using the positional parameters \$1 through to \$9. Each parameter corresponds to the position of the argument on the command line.
- The positional parameter \$0 refers to the command name or name of the executable file containing the shell script.
- Only nine command line arguments can be accessed, but you can access more than nine arguments using shift command.

```
while (( $# > 0 ))      # or [ $# -gt 0 ]
do
    echo $1
    shift
done
```

- All the positional parameters can be referred to using the special parameter \$*. This is useful when passing filenames as arguments.
 - `$ cat printps`
 `a2ps $* | lpr -Pps1`
 - `$ printps elm.txt vi.ref msg`
- This processes the three files given as arguments to the command printps.

Special shell variables

- There are some variables which are set internally by the shell and which are available to the user:

`$1 - $9` these variables are the positional parameters.

`$0` the name of the command currently being executed.

`$#` the number of positional arguments given to this invocation of the shell.

`$?` the exit status of the last command executed is given as a decimal string. When a command completes successfully, it returns the exit status of 0 (zero), otherwise it returns a non-zero exit status.

`$$` the process number of this shell - useful for including in filenames, to make them unique.

`$!` the process id of the last command run in the background.

`$-` the current options supplied to this invocation of the shell.

`$*` a string containing all the arguments to the shell, starting at `$1`.

Reading user input

- To read standard input into a shell script use the read command. For example:

```
echo "Please enter your name:"  
read name  
echo "Welcome to Linux Class $name"
```

- This prompts the user for input, assigns this to the variable name and then displays the value of this variable to standard output. If there is more than one word in the input, each word can be assigned to a different variable. Any words left over are assigned to the last named variable. For example:

```
echo "Please enter your surname\n"  
echo "followed by your first name: \c"  
read name1 name2  
echo "Welcome to Glasgow $name2 $name1"
```

Conditional statements

- Every Unix command returns a value on exit which the shell can interrogate. This value is held in the read-only shell variable `$?`. A value of 0 (zero) signifies success; anything other than 0 (zero) signifies failure.
- The if statement uses the exit status of the given command and conditionally executes the statements following. The general syntax is:

```
if test
then
    commands          (if condition is true)
else
    commands          (if condition is false)
fi
```

- then, else and fi are shell reserved words and as such are only recognized after a newline or ; (semicolon). Make sure that you end each if construct with a fi statement. if statements may be nested:

```
if ...
then ...
else if ...
    ...
fi
fi
```

Conditional statements

- The elif statement can be used as shorthand for an else if statement. For example:

```
if ...  
then ...  
elif ...  
...  
fi
```

- The && operator - You can use the && operator to execute a command and, if it is successful, execute the next command in the list. For example:

```
cmd1 && cmd2
```

This is a terse notation for:

```
if cmd1  
then  
    cmd2  
fi
```

- The || operator - You can use the || operator to execute a command and, if it fails, execute the next command in the command list. For example:

```
cmd1 || cmd2
```

Conditional statements

- Testing for files and variables with the test command
- The shell uses a command called test to evaluate conditional expressions. Full details of this command can be found in the test manual page. For example:

```
if test ! -f $FILE
then
    if test "$WARN" = "yes"
    then
        echo "$FILE does not exist"
    fi
fi
```

- First, we test to see if the filename specified by the variable \$FILE exists and is a regular file. If it does not then we test to see if the variable \$WARN is assigned the value yes, and if it is a message that the filename does not exist is displayed.
- The case statement

```
case word in
    pattern1) command(s)
        ;;
    pattern2) command(s)
        ;;
    patternN) command(s)
        ;;
esac
```

When all the commands are executed control is passed to the first statement after the esac. Each list of commands must end with a double semi-colon (;:).

Conditional statements

- The for statement
 - The for loop notation has the general form:

```
for var in list-of-words
do
    commands
done
```
 - commands is a sequence of one or more commands separated by a newline or ; (semicolon). The reserved words do and done must be preceded by a newline or ; (semicolon). Small loops can be written on a single line. For example:

```
for var in list; do commands; done
```
 - The while and until statements
 - The while statement has the general form:

```
while command-list1
do
    command-list2
done
```
 - The commands in command-list1 are executed; and if the exit status of the last command in that list is 0 (zero), the commands in command-list2 are executed. The sequence is repeated as long as the exit status of command-list1 is 0 (zero).
 - The until statement has the general form:


```
until command-list1
do
    command-list2
done
```
 - This is identical in function to the while command except that the loop is executed as long as the exit status of command-list1 is non-zero.
- <http://hadooptutorial.info/>

Environment variables

- The set command will display all the global functions written by the user
- The env command displays only the variables and not the functions
- We can reassign values for the variables either temporarily or permanently
- Temporary
 - export varname=value at the command prompt
- Permanent
 - export varname=value in .bashrc file at the root directory

```
export JAVA_HOME=/Library/Java/Home  
export JRE_HOME
```

```
export JAVA_HOME
```

A white rectangular card is placed on top of a red envelope. The card has the words "Thank you..." written in a black, cursive script. The envelope is red and its top flap is open, showing the white card. The background is a light gray surface.

Thank you...