Linux Training

PREPARED BY ENES ERTEN

Contents

File Searching

Message Digest

Process Text Streams

The Bash Shell

Linux Processes and Task Management

File Searching

- Can be done by find command

  # find [filepath] [options] [argument]

  # find [filepath] -name [filename]

  -> search for filename inside the directory

  # find [filepath] -ctime 1

  -> search for files that changed for last 1 day inside filepath

  # find [filepath] -atime 1

  -> search for files that accesed for last 1 day inside filepath

File Searching

find [filepath] -empty

-> search for empty files inside filepath

find [filepath] [option] -type [f | d] [filename]

-> search for files according to the file type f for file d for directory

find [filepath] [options] -exec [command] {}\;

-> finding file according to the options and execute command for the file

Ex: # find [filepath] -empty –type f exec rm –rf {}\;

File Globing

- * matches zero or more characters
- ? matches one characters
- [abc] matches anyone of the characters case sensitive
- [^abc] matches excepts inside the characters
- [0-9] matches 0 to 9

Ex:

```
# ls *.txt                    # ls [Pp]*.txt
# ls test.*                   # ls [Ww]ether[Rr]eport[1-5]?*
# ls ?.txt                    # ls [^W]ether[^R]eport[1-5]?*
# ls ????.txt                 # ls /dir/*
# ls test?.txt                # ls [star]*
```

Message Digest

- Message Digest is basically fingerprint of a file. It is the hash value of the contents of a file.
- md5sum: Calculates and checks a file's hash based on the MD5 algorithm.
- # md5sum [file] > test.md5

  # md5sum [file] -c test.md5
- sha256sum: Calculates and checks a file's hash based on the SHA-2 hash algorithm using 256-bits.

  # sha256sum [file] > test.sha256

  # sha256sum [file] -c test.sha256
- sha512sum: Calculates and checks a file's hash based on the SHA-2 hash algorithm using 512-bits.

  # sha256sum [file] > test.sha256

  # sha256sum [file] -c test.sha256

Process Text Streams

- sort command:

  # sort [file] # sort –n [file] -> number

  # sort –t "," -k2 [file] -> delimeter type and column to sort

  # sort –u [file]

- unique command

  # uniq [file] # uniq –c [file]

  # uniq --group [file]

- tr command

  # cat [file] | tr "," "|"

  # cat [file] | tr –d ","

  # cat [file] | tr 'A-Z' 'a-z'

Process Text Streams

- cut command:

  # cut –d [delimeter] -f [#ofColumns] [file]

  # cat [file] list.csv | tr ['delimiter'] ['delimiter'] | cut –f [#ofColumns]

- paste command

  # paste [file] [file]

  # paste –d [file] [file]

  # paste –s -d [delimiter] [file] [file]

Process Text Streams

- sed tool:

  # sed -i 's/[Str1]/[Str2]/g' [file]

  -> changes Str1 to Str2 globally (whole the file)

  If -i flag doesn't use it don't changed the file you have the redirect output to a file to save changes

- split command:

  # split [file] # split –b [#] [file]

  # split –d –n2 [file] # split –d --verbose –n[#] [file]

The Bash Shell

- A shell is referred to as the command interpreter, and it is the interface between a user and the Linux kernel. The shell accepts instructions (commands) from users (or programs),interprets them, and passes them on to the kernel for processing. The kernel utilizes all hardware and software components required for a successful processing of the instructions. When concluded, it returns the results to the shell, which then exhibits them on the screen. The shell also shows appropriate error messages, if generated. In addition, the shell delivers a customizable environment to users.
  A widely used shell by Linux users and administrators is the bash (bourne-again shell) shell.

The Bash Shell

- The bash shell is identified by the dollar sign ($) for normal users and the hash sign (#) for the root user. The bash shell is resident in the /usr/bin/bash file.
- Internal and External Commands: There is a rich collection of commands built in to the bash shell known as the internal commands. These include cd, pwd, umask, alias/unalias, history, command, . (dot), export,
exit, test, shift, set/unset, source, exec, and break. Upon invocation, these commands are executed directly by the shell without creating a new process for them. Other commands located in various directories, such as /usr/bin and /usr/sbin, are external to the shell, and the shell spawns a temporary sub-shell (child shell) to run them.

The Bash Shell

Environment Variable

A variable is a transient storage for data in memory. It retains information that is used for customizing the shell environment and referenced by many programs to function properly.

There are two types of variables: local (or shell) and environment. A local variable is private to the shell in which it is created, and its value cannot be used by programs that are not started in that shell. This introduces the concept of current shell and sub-shell (or child shell). The current shell is where a program is executed, whereas a sub-shell (or child shell) is created within a shell to run a program. The value of a local variable is only available in the current shell.

The value of an environment variable is inherited from the current shell to the sub-shell during the execution of a program. In other words, the value stored in an environment variable is accessible to the program, as well as any sub- programs that it spawns during its lifecycle. Any environment variable set in a sub-shell is lost when the sub-shell terminates.

env or printenv

The Bash Shell

- Declaring local variable

  # var="variable" # echo $var

  # bash # echo $var
- Making a variable an environment variable

  # export var="variable" # echo $var

  # bash # echo $var
- Unsetting an environment variable

  # unset var

- /etc/profile file can be used to adding collecting customed environment variable
- Affecting just user environment variable enter add variable to the `~/.bash_profile` file

The Bash Shell

- Input, Output and Error redirection

Programs read input from the keyboard and write output tothe terminal window where they are initiated. Any errors, if encountered, are printed on the terminal window too. This is the default behavior. The bash handles input, output, and errors as character streams. If you do not want input to come from the keyboard or output and error to go to the terminal screen, the shell gives you the flexibility to redirect input, output, and error messages to allow programs and commands to read input from a non-default source, and forward output and errors to one or more non-default destinations.

The default (or the standard) locations for the three streams are referred to as standard input (or stdin), standard output (or stdout), and standard error (or stderr).

The Bash Shell

- stdin (<): Input redirection instructs a command to read input from an alternative source, such as a file, instead of the keyboard.

  # cat < [file]

- stdout(>): Output redirection sends the output generated by a command to an alternative destination, such as a file, instead of to the terminal window. Alternatively pipe ('|') and ('1>')can be used.

  # echo "hello,world!" > [file]

  # echo "hello, world!" >> [file] ->appends the file

- stderr(2>):Error redirection forwards any error messages generated to an alternative destination rather than to the terminal window.

  # error 2> error.txt # error 2> /dev/null

  # error > error.txt 2>&1 -> combining redirection of stdin and stdout

The Bash Shell

- tee command: Redirects stdin to a file and terminal

  # [command] > tee [file]

- xargs command: execute a command to redicted stdin

  # find [filepath] [options] > xargs –I {} [command] {}

  # find [filepath] [options] > xargs –I {} mv {} [filepath]

The Bash Shell

- history command

  # history

  # history [#] -> shows # of entries

  # ![#] -> re-execute # in history

  # !ch ->To re-execute the most recent occurrence of a command that started with a particular letter or series of letters.

  # history –d [#] -> removes from history # entry

  # !! -> re-execute the last command

The Bash Shell

- Editing at the Command Line

Ctrl+a / Home -> Moves the cursor to the beginning of the command line
Ctrl+e / End -> Moves the cursor to the end of the command line
Ctrl+u -> Erase the entire line
Ctrl+k -> Erase from the cursor to the end of the command line
Alt+f -> Moves the cursor to the right one word at a time
Alt+b -> Moves the cursor to the left one word at a time
Ctrl+f / Right arrow -> Moves the cursor to the right one character at a time
Ctrl+b / Left arrow -> Moves the cursor to the left one character at a time

The Bash Shell

- Tab Completion

Tab completion (a.k.a. command line completion) is a bash shell feature whereby typing one or more initial characters of a file, directory, or command name at the command line and then hitting the Tab key twice automatically completes the entire name. In case of multiple possibilities matching the entered characters, it completes up to the point they have in common and prints the rest of the possibilities on the screen.
You can then type one or more following characters and press Tab again to further narrow down the possibilities. When the desired name appears, press Enter to accept it and perform the action. One of the major benefits of using this feature is the time saved on typing long file, directory, or command names.

The Bash Shell

- Tilde Substitution:

Tilde substitution (or tilde expansion) is performed on words that begin with the tilde character (~). The rules to keep in mind when using the ~ are:

~ -> home directory

~+ -> current directory

~- -> previous working directory

~[USER] -> user home directory

The Bash Shell

- Alias Substitution

Alias substitution (a.k.a. command aliasing or alias) allows you to define a shortcut for a lengthy and complex command or a set of commands. Defining and using aliases saves time and saves you from typing. The shell executes the corresponding command or command set when an alias is run.

alias aliascommand='command'

unalias aliascommand

This method create a temporary alias To create a permenant alias save the alias for the ~/.bashrc file

The Bash Shell

- Quoting Mechanisms: The backslash (\), single quotation (‘’), and double quotation (“”) characters, and work by prepending a special character to the backslash, or enclosing it within single or double quotation marks.
- The backslash character (\), also referred to as the escape character in shell terminology, instructs the shell to mask the meaning of any special character that follows it.

  #rm /*

The Bash Shell

- Single Quotes: The single quotation marks (‘’) instructs the shell to mask the meaning of all encapsulated special characters.

  # echo '$LOGNAME'

- Double Quotes: The double quotation marks (“”) commands the shell to
  mask the meaning of all but the backslash (\), dollar sign ($), and single quotes (‘’).

  # echo " '\' "

  # echo "$SHELL"

Linux Processes and Task Management

- Process: Any program, command, or application running on the system. Every process has a unique numeric identifier and it is managed by the kernel through its entire lifespan. It may be viewed, listed, and monitored, and can be launched at a non-default priority based on the requirement and available computing resources. A process may also be re-prioritized while it is running. A process is in one of several states at any given time during its lifecycle. A process may be tied to the terminal window where it is initiated, or it may run on the system as a service. There are plenty of signals that may be passed to a process to accomplish various actions. These actions include hard killing a process, soft terminating it, running it as a background job, bringing the background job back to the foreground, and forcing it to restart with the same identifier.

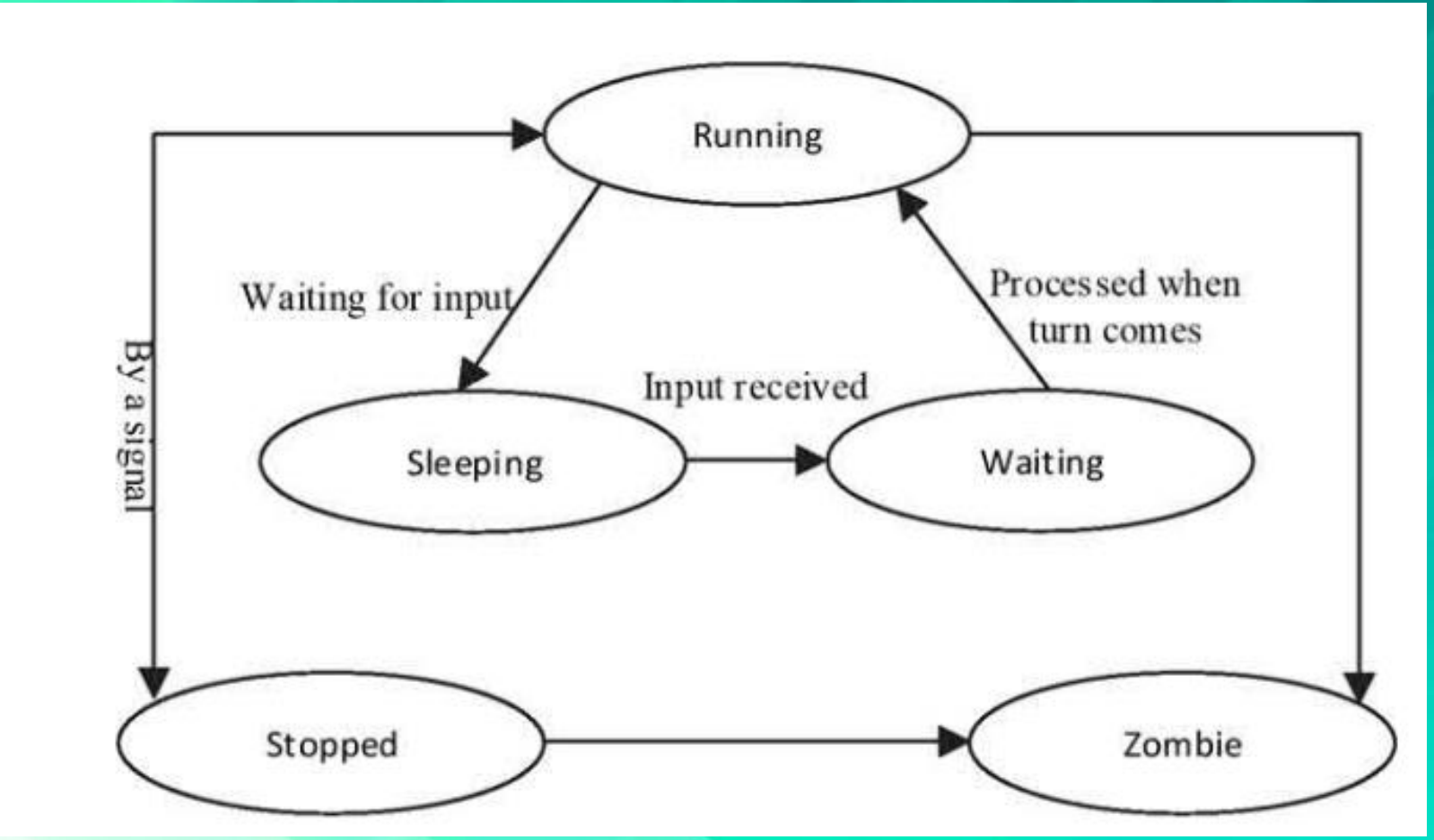
Linux Processes and Task Management

- Process and Priority: A process is a unit for provisioning system resources. It is any program, application, or command that runs on the system. A process is created in memory when a program, application, or command is initiated. Processes are organized in a hierarchical fashion. Each process has a parent process (a.k.a. a calling process) that spawns it. A single parent process may have one or many child processes and passes many of its attributes to them at the time of their creation. Each process is assigned an exclusive identification number known as the Process Identifier (PID), which is used by the kernel to manage and control the process through its lifecycle. When a process completes its lifespan or is terminated, this event is reported back to its parent process, and all the resources provisioned to it (cpu cycles, memory, etc.) are then freed and the PID is removed from the system.

Linux Processes and Task Management

- Process States: A process changes its operating state multiple times during its lifecycle. Many factors, such as load on the processor, availability of free memory, priority of the process, and response from other applications, affect how often a process jumps from one operating state to another. It may be in a non- running condition for a while or waiting for other process to feed it information so that it can continue to run.

Linux Processes and Task Management



Linux Processes and Task Management

- Process in Linux Systems

ps –eH | less

ps –u [username]

ps –ef

ps –o [colons]

ps –C [sshd]

UID User ID or name of the process owner
PID Process ID of the process
PPID Process ID of the parent process
C CPU utilization for the process
STIME Process start date or time
TTY The controlling terminal the process was started on.
"Console" represents the system console and "?" represents a daemon process.
TIME Aggregated execution time for a process
CMD The command or program name

Linux Processes and Task Management

- /proc directory is pseudo file system for processes. The kernel sends data on its runtime configuration to the directory.

  # top

- Top command will give real time information about processes. ps command take snapshots from /proc directory. It will give different output on anytime.

Linux Processes and Task Management

Top command columns meaning

Columns 1 and 2: Pinpoint the process identifier (PID) and owner (USER)
Columns 3 and 4: Display the process priority (PR) and nice value (NI)
Columns 5 and 6: Depict amounts of virtual memory (VIRT) and non-swapped resident memory (RES) in use
Column 7: Shows the amount of shareable memory available to the process (SHR)
Column 8: Represents the process status (S)
Columns 9 and 10: Express the CPU (%CPU) and memory (%MEM) utilization
Column 11: Exhibits the CPU time in hundredths of a second (TIME+)
Column 12: Identifies the process name (COMMAND)

Linux Processes and Task Management

- free command

  # free –h

- pgrep command

  # pgrep [processName]

  # pgrep –a [processName]

  # pgrep –u [username]

Linux Processes and Task Management

Signals: Signals are software interrupts sent to a program to indicate that an important event has occurred. The events can vary from user requests to illegal memory access errors. Some signals, such as the interrupt signal, indicate that a user has asked the program to do something that is not in the usual flow of control.

signal	value	action	description
SIGHUB	1	terminate	Hangup detected on controlling terminal or death of controlling process
SIGKILL	9	terminate	Kill signal (killing process)
SIGTERM	15	terminate	Terminate signal (closing process)

Linux Processes and Task Management

- kill command:

  # kill [PID]

  # kill –[signalValue] [PID]

  # pkill [processName]

  # killall [processName]

- jobs command:

  # jobs –l

Linux Processes and Task Management

- Job Scheduling: Allows a user to submit a command for execution at a specified time in the future. The execution of the command could be one time or periodic based on a pre-determined time schedule.
- Job scheduling and execution is taken care of by two service daemons: atd and crond. While atd manages the jobs scheduled to run one time in the future, crond is responsible for running jobs repetitively at pre-specified times.

  /var/spool/cron and /etc/cron.d are file locations

Linux Processes and Task Management

- at command:

  # at [hh:mm(pm|am)] [mm/dd/yy]

  Enter command to execute at the specified time.

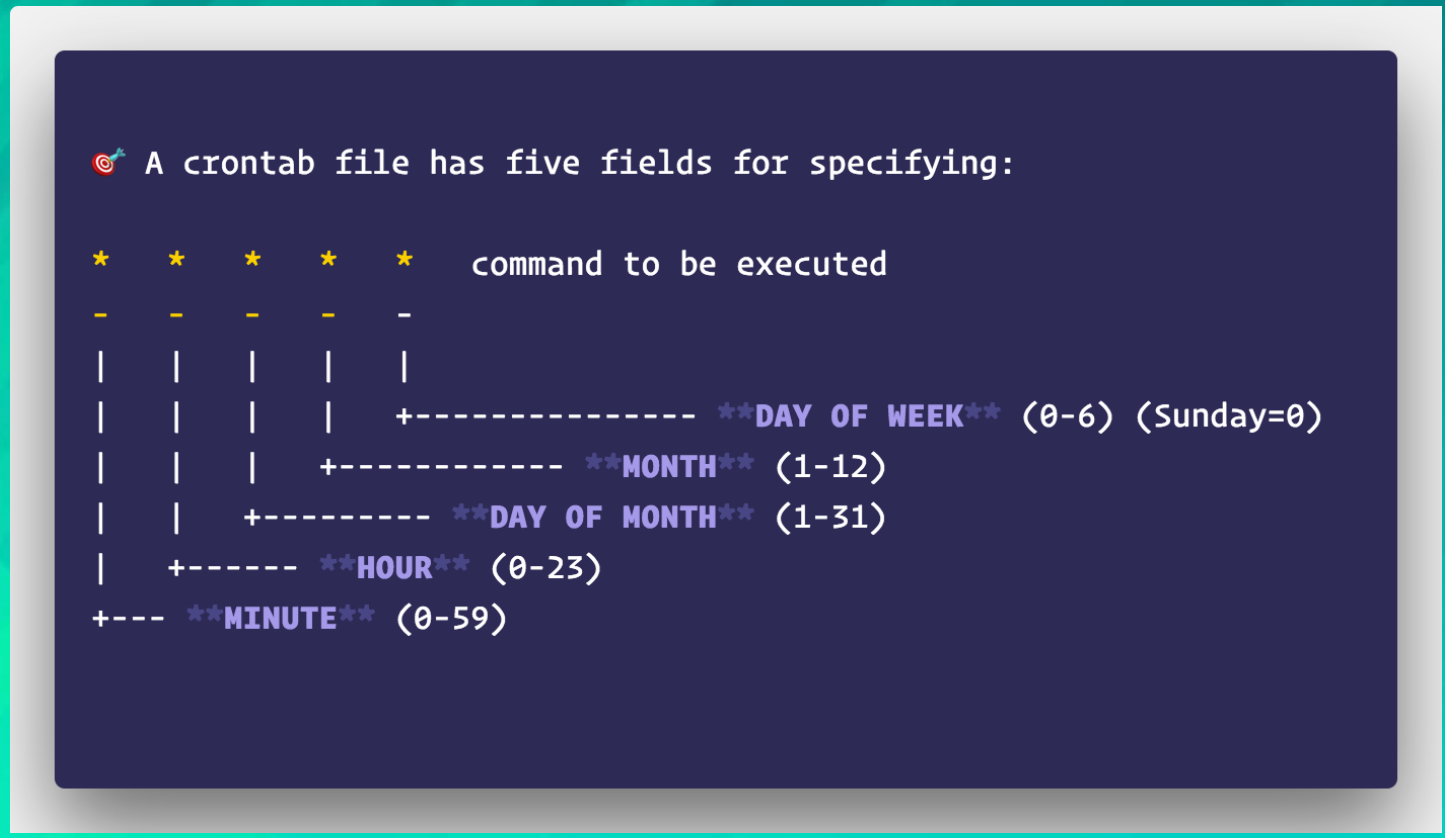
  # at –l -> list scheduled jobes

  # at –c [jobID] -> check scheduled job script

  # at –d [jobID] -> delete scheduled job

Linux Processes and Task Management

- crontab:
- https://crontab.guru/#



Linux Processes and Task Management

- crontab:

  # crontab –e -> it will open a file and add the cron table and command save and quit from the file

  # crontab –l -> list the scheduled jobs

  # crontab –r -> remove all scheduled jobs

  Modifying or deleting a scheduled job # crontab -e