# Linux Basics and Shell Scripting Chapter - 3

Presentation from Uplatz

Contact Us: https://training.uplatz.com/
Email: info@uplatz.com
Phone: +44 – 7836 212635

# 3. Linux Utilities

- Disk utilities (du, df, dd, hexdump)
- Process utilities (ps, kill, xkill, ulimit, sleep, bg, fg, jobs, top)
- Text processing utilities (find, locate, grep)
- Miscellaneous commands (history, tee, free, alias)
- Compressing and archiving  (for backup and restore) utilities (tar, gzip, gunzip, bzip2, bzcat, lzma, unlzma)

# Measuring disk usage

- Show the total size on disk of files or directories (**d**isk **u**sage)

Caution: different from file size!

```
$ du –h dir1 dir2 file1 file2
```

-h: returns size on disk of the given file, in human readable format: K (kilobytes), M (megabytes) or G(gigabytes). Without –h, du returns the raw number of disk blocks used by the file (hard to read).

Note that the –h option only exists in GNU du.

```
$ du –sh <dir>
```

-s: returns the sum of disk usage of all the files in the given directory.

# Measuring disk space

- Returns the size, total space and free space of the current partition.
- $ df –h <dir> (disk free)

Similarly, the –h option only exists in GNU df.

- Example
- $ df –h .

```
Filesystem   Size    Used  Avail  Use% Mounted on
/dev/hda5    9.2G  7.1G  1.8G  81%   /
```

- $ df –h

Returns disk space information for all filesystems (partitions) available in the system. When errors happen, useful to look for full filesystems.

# Special devices

Device files with a special behavior or contents

- /dev/null
  - The data sink! Discards all data written to this file. Useful to get rid of unwanted output, typically log information:
    - $ mplayer black_adder_4th.avi &> /dev/null
- /dev/zero
  - Reads from this file always return \0 characters. Useful to create a file filled with zeros:
    - $ dd if=/dev/zero of=disk.img bs=1k count=2048
- /dev/random
  - Returns random bytes when read. Mainly used by cryptographic programs. Uses interrupts from some device drivers as sources of true randomness ("entropy"). Reads can be blocked until enough entropy is gathered.

  ```
  $ hexdump –C –n 8 /dev/random
  ```

  - Check entropy_avail and poolsize (under /proc/sys/kernel/random) to keep an eye on the system's entropy pool.
- /dev/urandom
  - For programs for which pseudo random numbers are fine. Always generates random bytes, even if not enough entropy is available.
- /dev/full
  - Mimics a full device. Useful to check that your application properly handles this kind of situation.
- See man null or man zero or man random  or man full for details

# Full control on tasks

- Since the beginning, Unix supports true preemptive multitasking.
- Ability to run many tasks in parallel, and abort them even if they corrupt their own state and data.
- Ability to choose which programs you run.
- Ability to choose which input your programs takes, and where their output goes.

# Processes

"Everything in Unix is a file
Everything in Unix that is not a file is a process"

Processes

- Instances of a running programs.
- Several instances of the same program can run at the same time
- Each program in unix runs as a process
- Processes can create other processes
- The process that creates another one is called parent process
- The process that is created is called child process
  - Ex: The shell you login to is a process by itself
- Processes can interact with the files, processes and other resources
- Each process opens 3 files by default
  - Standard input
  - Standard ouput and
  - Standard error
- Data associated to processes:
  - Open files, allocated memory, stack, process id, parent, priority, state …

## Process

- Process is a running instance of the program.
- Every process holds an unique process id (PID).
- PID usually ranges from 0 to 32767
- Each process holds the parent process (PPID).

## Process Characteristics and Commands related to processes

- Process characteristics
  - Each process will be identified by a unique ID called ProcessID or PID
  - Each process belongs to a user and is denoted by UID
  - Each process is executed from a file called command
  - Each process consumes resources
    - CPU time
    - Memory
- Commands related to processes
- Process listing

```
$ ps –u <username>
$ ps –l
$ ps –f
$ ps –aef
$ ps -e
```

- Usage limits for all processes that are spawned by this session

```
$ ulimit -a
```

# Running jobs in background

- Same usage throughout all the shells
- Useful
  - For command line jobs which output can be examined later, especially for time consuming ones.
  - To start graphical applications from the command line and then continue with the mouse.
- Staring a task: add & at the end of your line:

```
$ find_prince_charming --cute --clever --rich  &
```

# Sleep Command

- Introduces delay for required number of seconds

```
$ sleep 50
```

# Switching processes

- Make process to switch from foreground to background and vice versa

```
$ sleep 123
^Z
[1]+ Stopped sleep 123

$ sleep 234
^Z
[2]+ Stopped sleep 234

$ fg
sleep 234
^Z
[2]+ Stopped sleep 234

$ fg 1
sleep 123
^Z
[1]+ Stopped sleep 123

$ bg 1
[1]+ sleep 123 &

$ bg
[2]+ sleep 234 &

$ fg 2
sleep 234
^Z
[2]+ Stopped sleep 234

$ fg 1
sleep 123
^Z
[1]+ Stopped
```

# Background job control

$ jobs

- Returns the list of background jobs from the same shell

```
[1]-  Running ~/bin/find_meaning_of_life --without-god &
[2]+ Running make mistakes &
```

$ fg

$ fg %<n>

- Puts the last/nth background job in foreground mode

- Moving the current task in background mode:

[Ctrl] Z

bg

$ kill %<n>

- Aborts the nth job.

Uplatz

# Job control example

```
$ jobs
[1]- Running ~/bin/find_meaning_of_list --without-god&
[2]+ Running make mistakes &
$ fg
make mistakes
$ [Ctrl] Z
[2]+ Stopped make mistakes

$bg
[2]+ make mistakes &
$ kill %1
[1]+ Terminated ~/bin/find_meaning_of_life --without-
god
```

## Listing all processes

…whatever shell, script or process they are started from

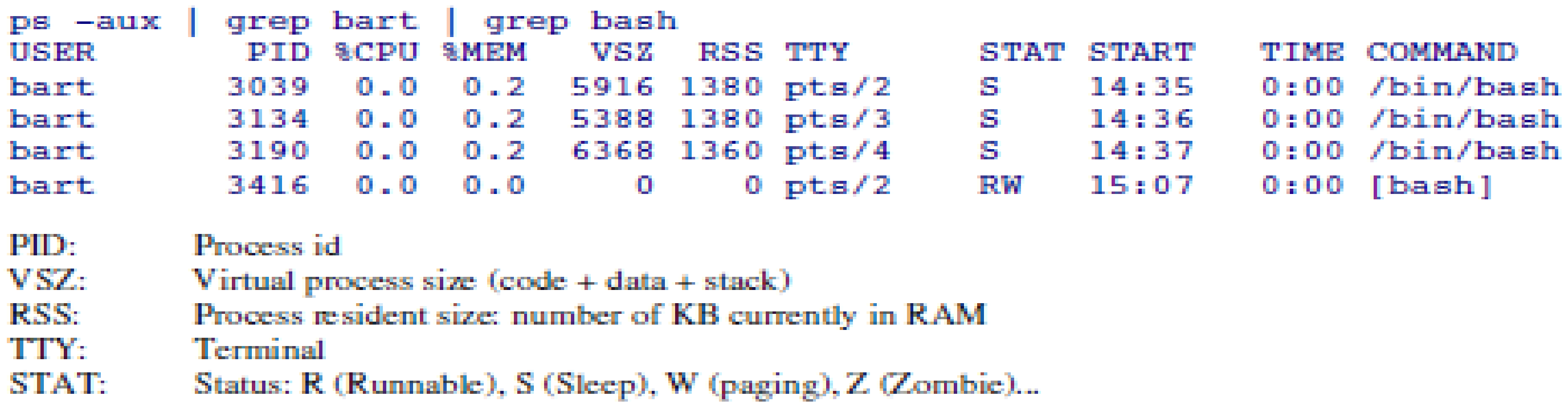
$ ps –ux

- ➢ Lists all the processes belonging to the current user

$ ps –aux (Note: ps –edf on System V systems)

- ➢ Lists all the processes running on the system

$ ps –aux | grep bart | grep bash



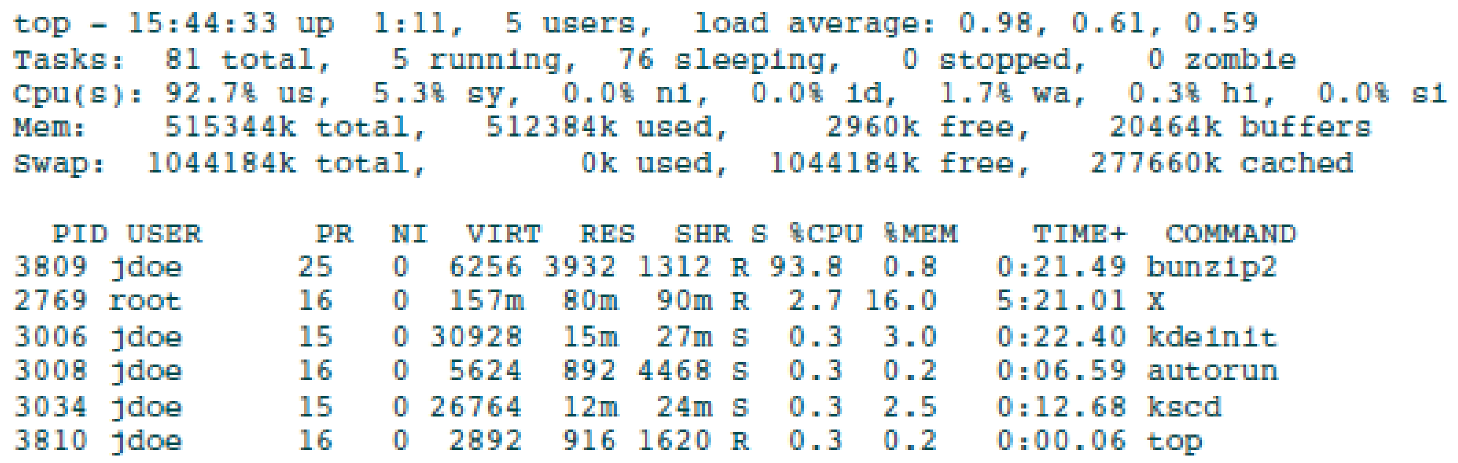
```
ps -aux | grep bart | grep bash
USER        PID %CPU %MEM   VSZ  RSS TTY      STAT START   TIME COMMAND
bart       3039  0.0  0.2  5916 1380 pts/2    S    14:35   0:00 /bin/bash
bart       3134  0.0  0.2  5388 1380 pts/3    S    14:36   0:00 /bin/bash
bart       3190  0.0  0.2  6368 1360 pts/4    S    14:37   0:00 /bin/bash
bart       3416  0.0  0.0     0    0 pts/2    RW   15:07   0:00 [bash]
```

| | |
|---|---|
| PID: | Process id |
| VSZ: | Virtual process size (code + data + stack) |
| RSS: | Process resident size: number of KB currently in RAM |
| TTY: | Terminal |
| STAT: | Status: R (Runnable), S (Sleep), W (paging), Z (Zombie)... |

# Live process activity

$ top

- Displays most important processes, sorted by cpu percentage



```
top - 15:44:33 up  1:11,  5 users,  load average: 0.98, 0.61, 0.59
Tasks:  81 total,   5 running,  76 sleeping,   0 stopped,   0 zombie
Cpu(s): 92.7% us,  5.3% sy,  0.0% ni,  0.0% id,  1.7% wa,  0.3% hi,  0.0% si
Mem:    515344k total,   512384k used,     2960k free,    20464k buffers
Swap:  1044184k total,        0k used,  1044184k free,   277660k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
3809 jdoe      25   0  6256 3932 1312 R 93.8  0.8   0:21.49 bunzip2
2769 root      16   0  157m  80m  90m R  2.7 16.0   5:21.01 X
3006 jdoe      15   0 30928  15m  27m S  0.3  3.0   0:22.40 kdeinit
3008 jdoe      16   0  5624  892 4468 S  0.3  0.2   0:06.59 autorun
3034 jdoe      15   0 26764  12m  24m S  0.3  2.5   0:12.68 kscd
3810 jdoe      16   0  2892  916 1620 R  0.3  0.2   0:00.06 top
```

- You can change the sorting order by typing
  - M: Memory usage, P: %CPU, T: Time
- You can kill a task by typing k and the process id.

## Killing processes

$ kill <pids>

- Sends an abort signal to the given processes. Lets processes save data and exit by themselves. Should be used first. Example:
- $ kill 3039 3134 3190 3416

$ kill -9 <pids>

- Sends an immediate termination signal. The system itself terminates the processes. Useful when a process is really stuck (doesn’t answer to kill -1).

$ kill -9 -1

- Kills all the processes of the current user. -1: means all processes.

$ killall [-<signal>] <command>

- Kills all the jobs running <command>. Example: $ killall bash

$ xkill

- Lets you kill a graphical application by clicking on it! Very quick! Convenient when you don’t know the application command name.

## File Searching Utilities

- Find the simplest and yet more effective means of looking up files on their system.
- Linux, like almost any other operating system, utilizes several mechanisms to answer search queries for users.
- Two of the most popular file searching utilities accessible to users are called
  - find
  - Locate
- The locate utility works better and faster than it's find counterpart because instead of searching the file system when a file search is initiated – Something find does – locate would look through a database.
- This database contains bits and parts of files and their corresponding paths on your system.

Uplatz

# The find command

- Searches files in a directory hierarchy.
  - Displays all the file names having extension .txt

    **$ find . -name "*.txt" -print**
  - Finds and removes all *.txt files. -exec tells to executed the command rm. {} \; this need to appended as it is including the spaces.

    **$ find . -name "*.txt" -exec rm {} \;**
  - Displays all the file names having permissions 777

    **$ find . -perm 777 -print**
  - Displays all files which were modified before 5 days back

    **$ find . -mtime +5 -print**
  - Displays all files which were modified within last 5 days

    **$ find . -mtime -5 -print**

Uplatz

# The locate command

```
$ locate f1.txt
$ locate f1.txt –n 5
$ locate -c f1.txt
$ locate -i f1.txt
$ sudo updatedb # Refresh mlocate Database

# Separate output entries without new line
$ locate -i -0 f1.txt
$ locate -S # Review your locate database

# Choose a different mlocate Location
$ locate –d <new db path> <filename>
```

## The grep command

- **Grep** is an acronym that stands for **G**lobal **R**egular **E**xpression **P**rint.
- Grep is a Linux / Unix command-line tool used to search for a string of characters in a specified file.
- The text search pattern is called a regular expression.
- When it finds a match, it prints the line with the result.
- The grep command is handy when searching through large log files.

Uplatz

```
$ grep this f1.txt
$ grep is f1.txt f2.txt f3.txt # Search multiple files
$ grep is * # Search all files in directory
$ grep –w is * # Find whole words only
$ grep -i This * # Ignore case
$ grep -r this * # Search sub-directories
$ grep -v this f1.txt # Inverse grep search
# Show lines that exactly match a search string
$ grep -x "this is line 2" f1.txt
$ grep -l this * # List Names of matching files
$ grep -c this * # Count number of matches
```

Uplatz

```
# Display number of lines before or after a search string
$ grep -A 2 vi f1.txt
$ grep -B 2 vi f1.txt
$ grep -C 2 vi f1.txt # Before and After

# Display Line Numbers with grep matches
$ grep -n -C 2 this f1.txt

# Limit grep output to a fixed number of lines
$ grep -m2 this f1.txt

$ grep -o this f1.txt # Display only matched pattern
$ grep ^this f1.txt # Match the lines that start with
$ grep "line 1$" f1.txt # Match the lines that end with
```

Uplatz

```
# Use multiple times with -e (Expression)
$ grep -e "this" -e "is" -e "line" f1.txt

# Takes patterns from file, one per line
$ cat patterns.txt
this
is
$ grep -f patterns.txt f1.txt
```

# The grep command

- Scans the given files and displays the lines which match the given pattern.
  - grep <pattern> <files>
- Displays all the lines containing error in the *.log files
  - grep error *.log
- Same, but case insensitive
  - grep –i error *.log
- Same, but recursively in all the files in . and its subdirectories.
  - grep –ri error .
- Outputs all the lines in the files except those containing info.
  - grep –v info *.log

Uplatz

# Commands – egrep and fgrep

- Match extended pattens – egrep
- Fixed grep (fgrep)
- Take multiple patterns, each separated by new line (fgrep)
- Extended grep (egrep) is also executed as grep -E
- Fixed grep (fgrep) is also executed as grep –F

## Differences between grep, egrep and fgrep

```
$ cat check_file.txt
grep is a command that can be used on unix-like systems.
it searches for any string in list of strings or file.
It is very fast.
(f|g)ile
$ grep -C 0 '(f|g)ile' check_file.txt
(f|g)ile
$ grep -C 0 '\(f\|g\)ile' check_file.txt
it searches for any string in list of strings or file.
```

https://www.tecmint.com/difference-between-grep-egrep-and-fgrep-in-linux/

Uplatz

```
$ cat check_file.txt
grep is a command that can be used on unix-like systems.
it searches for any string in list of strings or file.
It is very fast.
(f|g)ile
$ grep '(f|g)ile' check_file.txt
(f|g)ile
$ grep '\(f\|g\)ile' check_file.txt
it searches for any string in list of strings or file.
$ egrep '(f|g)ile' check_file.txt
it searches for any string in list of strings or file.
$ egrep '\(f\|g\)ile' check_file.txt
(f|g)ile
$ fgrep '(f|g)ile' check_file.txt
(f|g)ile
$ fgrep '\(f\|g\)ile' check_file.txt
$
```

# Command History

- $ history
  - Displays the latest commands that you ran and their number. You can copy and paste command strings.
- You can recall the latest command:
  - !!
- You can recall a command by its number
  - !1003
- You can recall the latest command matching a starting string
  - !cat
- You can make substitutions on the latest command:
  - ^more^less
- You can run another command with the same arguments:
  - $ more !*

Uplatz

## The tee command

- The tee command can be used to send standard output to the screen and to a file simultaneously.

```
$ tee [-a] file
```

- Runs the make command and stores its output to build.log.

```
$ make | tee build.log
```

- Runs the make install command and appends its output to build.log.

```
$ make install | tee –a build.log
```

- Another Example

```
$ grep -i this file1.txt | sort -u | tee teeTest.txt
```

Uplatz

## The free command

- Display amount of free and used memory in the system

```
$ free
$ free -h
```

## Compressing and decompressing

Very useful for shrinking huge files and saving space

```
$ g[un]zip <file>
```

GNU zip compression utility. Creates .gz files. Ordinary performance (similar to Zip).

```
$ b[un]zip2 <file>
```

More recent and effective compression utility. Creates .bz2 files. Usually 20-25% better than gzip.

```
$ [un]lzma <file>
```

Much better compression ration than bzip2 (upto 10 to 20%). Compatible command line options.

## Archiving - tar

Useful to backup or release a set of files within 1 file

- tar: originally “tape archive”
- Syntax

```
$ tar cvf <archive> <files or directories>
$ tar tvf <archive>
$ tar xvf <archive>
$ tar xvf <archive> <files or directories>
```

  - c: create. Creating an archive.
  - t: test. Viewing the contents of an archive or integrity check.
  - x: extract. Extracting all the files from an archive or extracting just a few files or directories (paths relative to archive root directory) from an archive.
  - v: verbose. Useful to follow archiving progress.
  - f: file. Archive created in file (tape used otherwise)
- Example

```
$ tar cvf /backup/home.tar /home
$ bzip2 /backup/home.tar
```

Uplatz

# Archiving - cpio

- Copy in and out of archives
- 3 major variants
- Take the list of files from standard input, prepare an archive and send it to the standard output
  - $ cpio –o ….
- Take the archive from the standard input, write to files in the current directory
  - $ cpio –i ….
- Pass through: Combination of –o and –i, prepare an archive from first directory and dump that structure into second directory.

# Extra options in GNU tar

- GNU tar on GNU / Linux → tar = gtar
- Can compress and uncompress archives on the fly. Useful to avoid creating huge intermediate files. Much simpler to do than with tar and bzip2!
  - Option j: [un]compresses on the fly with bzip2
  - Option z: [un]compresses on the fly with gzip
  - --lzma option: [un]compresses on the fly with lzma
  - Examples (which one will you remember?)

```
$ gtar jcvf bills_bugs.tar.bz2 bills_bugs
$ tar cvf – bills_bugs | bzip2 > bills_bugs.tar.bz2
```

Uplatz