



Linux Administration Notes

T.J. ROBINSON

November 25, 2025

1 File Management

1.1 File Permissions

File Permissions are used to prevent unauthorized access by users to files and directories

1.2 Permission Classes

Permission Classes unique categorizes utilized by the kernel to maintain file security via access rights.

Users are assigned to 3 categories:

1. **User Owner (u)**
2. **Group (g)**
3. **Other (o)**
4. **All (a)** - represents all 3 classes

1.3 Permission Types

There are 3 types of permission bits:

1. **Read (r)** - view and copy
2. **Write (w)** - modify
3. **Execute (x)** - run
4. **null (-)** - permission not granted

1.4 Permission Modes

1. Append permission bit (+)
2. Revoke permission bit (-)
3. Assign permission bit (=)

1.5 Modifying Permissions

chmod is used to change permissions of files & directories

1.5.1 Symbolic vs. Octal Notation

- **Symbolic Notation** uses letters (ex. **u,g,o**) & symbols (ex. **+,,-,=**) to modify permissions.
- **Octal Notation** uses 3-digit numbering (ex. **766**) to modify permissions.

1.6 Default Permission

umask is used to set default permissions on a file without modify permissions on existing files and directories.

- The default *umask* value for all users including the root user is **0022**.
- The default initial permission value for files is **666** & **777** for directories.

1.7 Calculating Default Permission

Calculating default permissions for files:

Initial Permission	666
umask	- 022
<hr/>	
Default Permission	044

Calculating default permissions for directories:

Initial Permission	777
umask	- 022
<hr/>	
Default Permission	055

1.8 Special File Permission

There are 3 Special Permission Bits that can be configured for binary files and directories:

1. **SETUID** (SET User IDentifier) - applied to binary executable files at the *user owner (u)* level.
It gives non-owners the same file permissions as the user owner.
2. **SETGID** (SET Group IDentifier) - applied to binary executable files at the *user owner (u)* level.
It gives non-owners and group members the same file permissions as the user & group owner.
3. **Sticky Bit** - is set on public directories to prevent other users from deleting or moving files.

1.9 File Searching

find is the command used to search for files on a Linux System and perform actions on found files.

After invoking the find command, the first option is the location path to search (ex. current **(.)**, **/tmp**, **/home/**).

1.10 find Command Options

- use **-iname** to search for files that begins with a string.

example input: `find /dev -iname usb*`

example output:

```
/dev/usb1  
/dev/monusb0  
/dev/monusb1
```

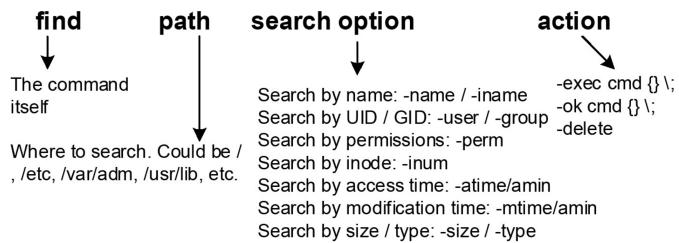


Figure 1: **find** Command Syntax

- use **-size** to search for files by size
 - use (-) to find items smaller than designated size
example input: `find /dev -size -2M`
 - use (+) to find items larger than designated size
example input: `find /dev -size +2M`
- find files owned by a specific user (*daemon*) and exclude specific group (*user1*).
example input: `find /dev -user daemon -not -group user1`
- use **-type** to search by filetype (d=directory, f=file)
example input: `find /usr -type d -name src`
- use **-maxdepth** to search set maximum subdirectory depth to search
example input: `find /home -maxdepth 3 -type f -name src`

1.10.1 Using the **-exec** and **-ok** options

- **-exec** is used to perform actions on the files found by **find**.
- **-ok** is the same as **-exec**, but requires user confirmation to execute.

example input: `find /Documents -type f -name BLS* -exec ls -ld {} \;`

- `({})` represents each file found
- `(;)` terminates the command.
- `(\)` is used to escape `(;)`

2 Linux Processes and Job Scheduling

2.1 Processes & Priorities

process a unit for provisioning system resources. It is any program, command, or application running on the system.

daemon critical system processes that startup automatically and run in the background.

- One parent process can spawn one or many child processes and passes attributes to them during creation (i.e., nice score).
- A *Process Identification Number (PID)* is assigned to each process.
- The PID is utilized by the kernel to manage the process during its lifespan.

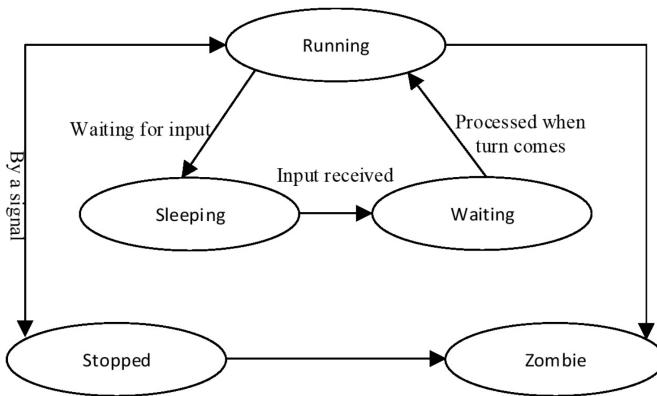


Figure 2: Process States

2.2 Process States

A process can jump from one operating state to another throughout its lifespan. Every process is in one of the **5 Basic Operating States**:

1. **running** - the process is currently being executed on the system.
2. **sleeping** - the process is waiting for input from the user or other source.
3. **waiting** - input has been received by the process and it is waiting to run.
4. **stopped** - the process has been halted and will not run until a signal is received to change its state.
5. **zombie** - The process is *dead*, aka *defunct*. There is an entry in the process table, but the process takes up no resources (i.e., CPU).

2.3 Viewing and Monitoring System Processes using ps & top

- **ps** (process status)
 - Useful Commands & Options
 - * `ps -as` output all processes and include file size.
 - * pipe `ps` output to `grep` to filter output.
- **top** (table of processes)

2.4 Process Niceness & Priority

- the `nice` command can launch a program at a non-default priority.
 - nice can also be used to confirm default nice score on RHEL systems.
- the `renice` command is utilized to alter the priority of running processes.
- A process's execution priority is determined by the nice score assigned to it when it spawned. There are 40 niceness scores from -20 (best) to 19 (worst).
- Higher Nice Score = More CPU Attention
- The default `nice` score for a process is 0.
- Child process inherit the nice score of its parent (calling) process.

```

devconnected@debian-10:/etc/cron.d$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# ----- minute (0 - 59)
# | ----- hour (0 - 23)
# | | ----- day of month (1 - 31)
# | | | ----- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | ----- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# | | | |
# * * * * * user-name command to be executed
17 * * * * root    cd / && run-parts --report /etc/cron.hourly
25 6 * * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6 * * 7 root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6 1 * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#

```

Figure 3: `/etc/crontab` File

3 Job Scheduling

job scheduling allows users to run a command in the future.

atd is the service daemon used to schedule a one time task in the future.

crond is the service daemon used to schedule repetitive tasks.

- during system boot the cron daemon reads jobs from `/var/spool/cron/crontabs` (user cron jobs) & `/etc/cron.d` (system cron jobs).
- `crond` loads scheduled jobs into memory at boot and scans the files regularly thereafter.

crontab linux app used to edit the `/etc/crontab` file.

3.1 Crontab Usage

crontab -e edit `/etc/crontab`

crontab -i view scheduled cronjobs