Linux User Group Bern

Introduction to Linux System Administration

Cedric Bösiger / Patrik Schilt

Linux System Administration

Today's topics:

- Mastering processes management
- Mastering memory management
- Managing users, groups and passwords
- Starting and Stopping daemons and configuring run-levels
- Scheduling automatic tasks
- Configuring system logging and reading popular log files

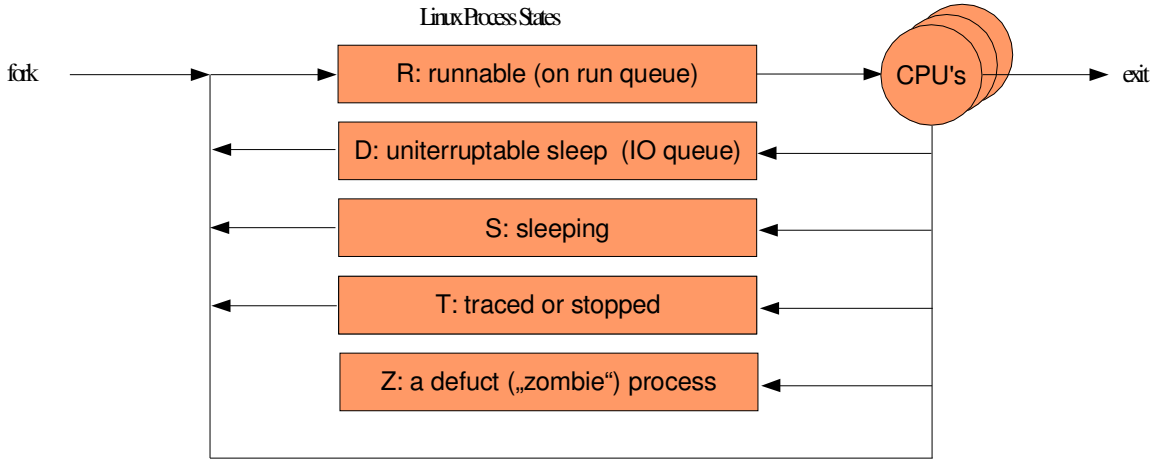
Note: File and Filesystem manipulation is not part of this presentation.

Process and memory management

process management

DESCRIPTION

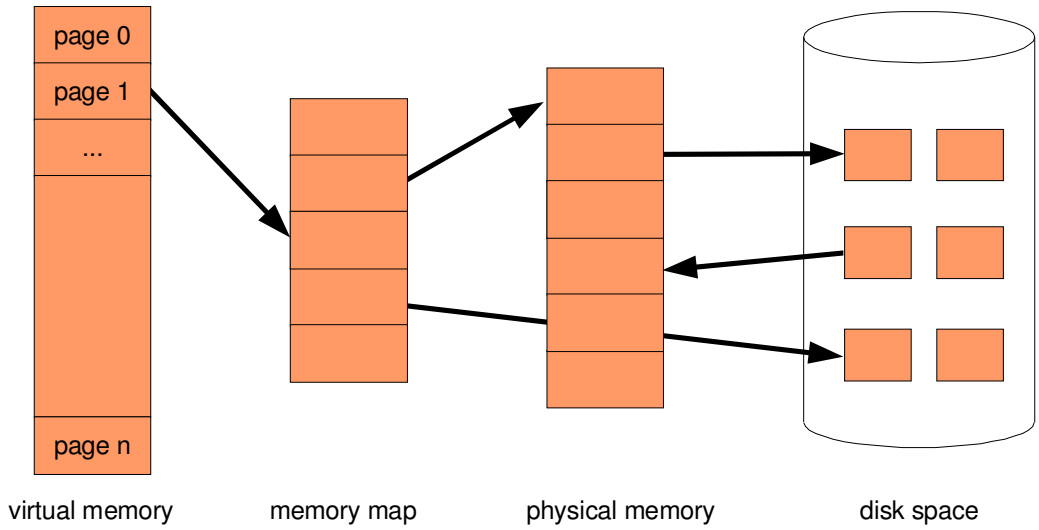
In Unix a running program is a process. Every process holds its own unique process id (PID). Unix is a time sharing system, which means that the processes take turns on running on the CPU's. Each turn is called timeslice. The loading and unloading of processes on the CPU is called context switching. All the processes loaded on the system are organized by process states (queues).



virtual memory

DESCRIPTION

Virtual memory is a technique that allows the execution of processes that may not be completely in memory. It separates the logical memory from the physical one. This separation allows an extremely large virtual memory. In Linux the virtual memory is implemented by demand paging.



ps(1) - report process status

DESCRIPTION

ps gives a snapshot of the current processes.

EXAMPLE

Show all processes of all users running on the system (BSD stile)

```
# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   1332   484 ?        S    20:56   0:04 init
root         2  0.0  0.0      0     0 ?        SW   20:56   0:00 [keventd]
root         3  0.0  0.0      0     0 ?        SW   20:56   0:00 [kapmd]
course   22084  0.0  0.2   4344  1372 pts/4    S    00:13   0:00 bash
course   22085  0.0  0.1   2728   800 pts/4    R    00:13   0:00 ps aux
```

- USER - Owner
- PID - Process ID
- %CPU - CPU time / real time percentage
- %MEM - Virtual memory percentage
- VSZ - Total virtual memory used
- RSS - Resident set size (Physical memory used)
- TTY - The minor ttyp number (Terminal owning the process)
- STAT - Process states
- START - Start time of process
- Time - Consumed CPU time

ps(1) - report process status

EXAMPLE

Show all processes owned by root

```
# ps uU root
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   1332   484 ?        S    20:56   0:04 init
root         2  0.0  0.0      0     0 ?        SW   20:56   0:00 [keventd]
root         3  0.0  0.0      0     0 ?        SW   20:56   0:00 [kapmd]
```

Options

- a - Select all processes, including those of other users
- u - Display user oriented format
- x - Select processes without controlling ttys
- U - Select processes by specified user
- -C - Select by command name
- r - Restrict output to running processes
- v - Display virtual memory format

ps(1) - report process status (cont.)

EXAMPLE

Show the virutal memory usage of every process

```
# ps avx
  PID TTY      STAT   TIME  MAJFL   TRS   DRS  RSS %MEM COMMAND
    1 ?        S      0:04    120    27  1304  484  0.0 init
    2 ?        SW     0:00      0     0     0    0  0.0 [keventd]
    3 ?        SW     0:00      0     0     0    0  0.0 [kapmd]
 1513 ?        S      0:00    182   186 11961 3512  0.6 /usr/bin/gdm
 1514 ?        R     25:46   1822  1506 291305 27348  5.3 /usr/X11R6/bin/X :0
 2008 ?        S      0:00   2651   104 18015 8724  1.6 /usr/bin/gnome-session
 2017 ?        S      0:00      6    44  2307  800  0.1 /usr/bin/ssh-agent --
```

MAJFL - Major Faults. The number of major faults the process has made, those which have required loading a memory page from disk

TRS - Text Resident Size. Size of the text segment (does not hold shared libraries)

DRS - Date Resident Size. Size of the data segment (includes shared libraries)

RSS - Resident Set Size. Size of the process in physical memory.

ps(1) - report process status (cont.)

EXAMPLE

Search for processes

```
# ps aux | grep gdm
 1513 ?        S      0:00    182   186 11961 3512  0.6 /usr/bin/gdm

# pgrep gdm
 1513 gdm
```

Show the process with the highest Memory consumption at the bottom of the list. Repeat this every second. :-)

```
# while (true) do ps havx | awk ' { print $8 " " $10}' | sort -n ; echo "---" ;
sleep 5 ; done
...
11312 gnome-panel
11620 gnome-terminal
13576 /usr/libexec/gweather-applet-2
26380 /usr/X11R6/bin/X
33960 /usr/lib/mozilla/mozilla-bin
56596 /opt/OpenOffice.org1.0.1/program/soffice.bin
---
```

Linux User Group Bern

vmstat(8) - report virtual memory

DESCRIPTION

vmstat provides information about processes, memory, paging, block IO, traps and cpu activity. The first report produced gives average values since the last reboot of the system. All additional reports are averages of the sampling periods.

EXAMPLE

Show 5 reports with a delay of 1 second

```
# vmstat 1 5
   procs                  memory      swap          io     system      cpu
 r  b  w   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id
 3  0  0      0  54968  51264 251140    0    0    10     3  181  1627 13  4 83
 0  0  0      0  54956  51264 251140    0    0     0     0  206   814  3  1 96
 2  0  0      0  54960  51264 251140    0    0     0     0  243   909  3  0 97
 0  0  0      0  54928  51264 251140    0    0     0     0  273  1011  3  0 97
 2  0  0      0  54772  51264 251268    0    0   128     0  205   824  2  1 97
```

Procs

r: The number of processes waiting for run time.
b: The number of processes in uninterruptable sleep.
w: The number of processes swapped out but otherwise runnable.

vmstat(8) - report virtual memory

Memory

swpd:	the amount of virtual memory used (kB).
free:	the amount of idle memory (kB).
buff:	the amount of memory used as buffers (kB).
cache:	the amount of memory used as cache (kB).

Swap

si:	Amount of memory swapped in from disk (kB/s).
so:	Amount of memory swapped to disk (kB/s).

IO

bi:	Blocks sent to a block device (blocks/s).
bo:	Blocks received from a block device (blocks/s).

CPU

us:	user time (%)
sy:	system time (%)
id:	idle time (%)

top(1) - display top cpu processes

DESCRIPTION

top provides an ongoing look at processor activity in real time. It displays a listing of the most CPU-intensive tasks on the system, and can provide an interactive interface for manipulating processes. It can sort the tasks by CPU usage, memory usage and runtime. Most features can either be selected by an interactive command.

EXAMPLE

Show the process activities

```
# top
 16:23:31  up  4:24,  4 users,  load average: 0.03, 0.05, 0.02
67 processes: 64 sleeping, 2 running, 0 zombie, 1 stopped
CPU states:  0.5% user,  0.7% system,  0.0% nice,  0.0% iowait, 98.7% idle
Mem:   514964k av,  318132k used,  196832k free,       0k shrd,   14492k buff
        45960k active,             244252k inactive
Swap:  489972k av,       0k used,  489972k free                  187752k cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
 4448 course    17   0  1056 1056   832 R     0.7  0.2   0:00 top
 2432 course    18   0 11564  11M  7248 R     0.5  2.2   0:11 gnome-terminal
    1 root       9   0   484  484   420 S     0.0  0.0   0:04 init
  ...
```

top(1) - display top cpu processes (cont.)

EXAMPLE

Press **?** or **h** for help

```
Interactive commands are:


h or ?  Help            Space   Update display
q       Quit            ^L      Redraw the screen
oO      Change order of displayed fields
fF      Add and remove fields
W       Write configuration file ~/.toprc
n or #  Set the number of processes to show
u       Show only a specific user
k       Kill a task (with any signal)
r       Renice a task
s       Set the delay in seconds between updates
Toggle:
    C:collapsed SMP CPU info    H:threads               l:load average
    S:cumulative mode           i:idle processes        m:memory info
    I:Irix/Solaris view (SMP)   c:command line          t:summary info
Sort by:
    A:age               M:resident memory usage
    N:pid               T:time (or cumulative time)
    P:CPU usage
```

Job control

DESCRIPTION

Job control lets you place foreground jobs in the background, bring background jobs to the foreground, or suspend (temporarily) stop running jobs. Job control is a function provided by the shell as Built-in command.

COMMANDS

bg	Put a job in the background
fg	Put a job in the foreground
jobs	List active jobs
kill	Terminate a job
CTRL-Z	Suspend a foreground job
&	Start job as background job

Job control (cont.)

EXAMPLE

Suspend a foreground job by pressing CTRL-Z

```
# xterm
CTRL-Z
[1]+  Stopped                 xterm
```

List jobs

```
# jobs
[1]-  Stopped                 vi /tmp/test
[2]+  Stopped                 xterm
[3]   Running                 tail -f /var/log/messages &
```

Run a stopped job in background

```
# jobs
[1]-  Stopped                 vi /tmp/test
[2]+  Stopped                 xterm
[3]   Running                 tail -f /var/log/messages &
# bg 2
[2]+ xterm &
```

Job control (cont.)

EXAMPLE

Run a job in foreground

```
# jobs
[1]-  Stopped                 vim /tmp/test
[2]+  Stopped                 xterm
[3]   Running                 tail -f /var/log/messages &

# fg 1
```

Start a process direct into background

```
# xclock &
```

kill(1) - terminate a process

DESCRIPTION

kill sends the specified signal to the specified process. If no signal is specified, the TERM signal is sent as default. If the TERM signal does not end the process, it might be necessary to use the KILL (9) signal, since this signal cannot be caught by the process.

EXAMPLE

Find and kill a process

```
# ps aux | grep ssh
course      1407  0.0  0.2  2720 1260 ?        S    11:59   0:00 /usr/sbin/sshd

# kill 1407
```

Kill all ssh processes owned by course

```
# pkill ssh -U course
```

User management

/etc/passwd - The user database

DESCRIPTION

The file /etc/passwd contains user attributes. It is an ASCII File containing for each user one entry. An entry has the following form:

```
name:password:uid:gid:comment:home_dir:shell
```

- name - Login name
- password - The encrypted password.
  `x` indicates that the password is in the /etc/shadow file.
- uid - User ID
- gid - Initial group ID
- comment - A comment. Usually the real name
- home_dir - The home directory
- shell - The default shell

EXAMPLE

The entry for the user *course* will look like this:

```
# grep course /etc/passwd
course:x:5001:100::/home/course:/bin/bash
```

/etc/group - The group database

DESCRIPTION

The file /etc/group contains group attributes. It is an ASCII File holding for each group one entry. An entry has the following form:

```
name:password:gid:user1,user2,...,userN
```

- name - Group name
- password - The encrypted password.
  If empty, no password is needed
- gid - Group ID
- users_list - All group member's user names, separated by commas

EXAMPLE

The entry for the group *users* will look like this:

```
# grep users /etc/group
users::100:user1,user2
```

/etc/shadow - The password database

DESCRIPTION

The file /etc/shadow contains passwords and password aging information. It is an ASCII File containing for each user one entry and is only readable by root. An entry has the following form:

```
name:password:lastchg:min:max:warn:inactive:expire:flag
```

- name - User name
- password - The encrypted password.
  Empty, no password required
  * or !, account is disabled
- lastchg - Number of day's since the password was changed
- min - Number of day's before the password may be changed
- max - Number of day's after the password must be changed
- warn - Number of day's to warn a users before expiration
- inactive - Number of day's after expiration that the account gets disabled
- expire - Number of day's the account has been disabled
- flag - reserved (not used)

add, modify and delete users and groups

DESCRIPTION

useradd(8) creates a new user or modifies an existing user.
It will add the according entries into the system files `/etc/passwd`, `/etc/group` and `/etc/shadow` and creates a home directory for the user. The initial configuration files will be copied into the new home directory.

The most important parameters of a Unix user are:

- login - Unique name in the system
- uid - User ID
- gid - Initial Group ID
- home_dir - Home directory
- shell - Default shell

The user `root` is the unix super user account. Root has always the UID = 0 and has access to the complete system. There are no restrictions for this user. It is not a good idea to use the root account for daily work and only a few selected people should have access to this account.

add, modify and delete users and groups (cont.)

EXAMPLE

Creates a user *course* with uid 5001 and gid 100

```
# useradd -m -u 5001 -g 100 course
```

Changes the default shell of the user *course* to tcsh

```
# usermod -s /bin/tcsh course
```

Delete the user *course* and remove its related files

```
# userdel -r course
```

Creates a new user group *class*

```
# groupadd class
```

Deletes the group *class*

```
# groupdel class
```

passwd - change user password

DESCRIPTION

passwd changes passwords for user and group accounts. A normal user may only change the password of its own account, while the root user can change any account.

The password will tested for complexity. It should consists of 6 to 8 characters including one of the following:

- Lower case alphabetics
- Upper case alphabetics
- Digits 0 through 9
- Punctuation marks

EXAMPLE

Creates a password for the user *course*

```
# passwd course
New UNIX password: ######
Retype new UNIX password: ######
```

id - Display user id

DESCRIPTION

id displays the user id (uid) and its group id's and names. This command is useful to query the groups a user belongs to.

EXAMPLE

show my own id

```
# id
uid=5001(course) gid=100(users) groups=100(users)
```

show the id of root

```
# id root
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),10(wheel)
```

w - get a system overview

DESCRIPTION

w shows who is logged on and what they are doing. The header shows the current time, how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5 and 15 minutes.

It shows for each logged on user the following information:

- Login name
- TTY used
- Remote host (if any)
- Login time
- Idle time
- JCPU (CPU time consumed by all processes attached to the tty)
- PCPU (CPU time consumed by the process)
- Command line

w - get a system overview (cont.)

EXAMPLE

Show who is logged in

```
# w
 13:31:34  up  3:26,  4 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
course   vc/1     -                 1:30pm  1:06   0.01s  0.01s  -bash
root     vc/2     -                 1:30pm 53.00s  0.01s  0.01s  -bash
cedric   vc/3     -                 1:30pm 19.00s  0.07s  0.06s  ssh base
cedric   pts/4    base              1:31pm 19.00s  0.00s  0.00s  -bash
```

last - Last logged in users

DESCRIPTION

last shows a listing of last logged in users. It searches back tghough the file `/var/log/wtmp` and displays a list of all logged in users.
The pseudo user reboot logs in at each time the system is rebootet. Thus `last reboot` will show a log of all reboots.

EXAMPLE

show the last logins

```
# last | more
cedric   pts/4        base             Sun Apr 20 13:31   still logged in
cedric   vc/3                          Sun Apr 20 13:30   still logged in
root     vc/2                          Sun Apr 20 13:30   still logged in
course   vc/1                          Sun Apr 20 13:30   still logged in
cedric   pts/3        :0.0             Sun Apr 20 13:06   still logged in
cedric   pts/3        :0.0             Sun Apr 20 12:56 - 13:06  (00:09)
course   vc/1                          Sun Apr 20 12:32 - 12:32  (00:00)
course   vc/1                          Sun Apr 20 12:29 - 12:29  (00:00)
cedric   :0                            Sun Apr 20 10:08   still logged in
reboot   system boot  2.4.19          Sun Apr 20 10:05          (04:09)
```

su - Change user id's

DESCRIPTION

su (switch user) is used to become another user during a login session. Invoked without a username, su defaults to become super user. The argument - may be used to provide an environment similar to the real logged in user.

EXAMPLE

Become the user *course* without the shell environment

```
# su course
Password: #####
```

Become the user *root* with the shell environment

```
# su -
Password: #####
```

System logging, crontab and run-levels

init(8), inittab(5) - sysv-compatible init process

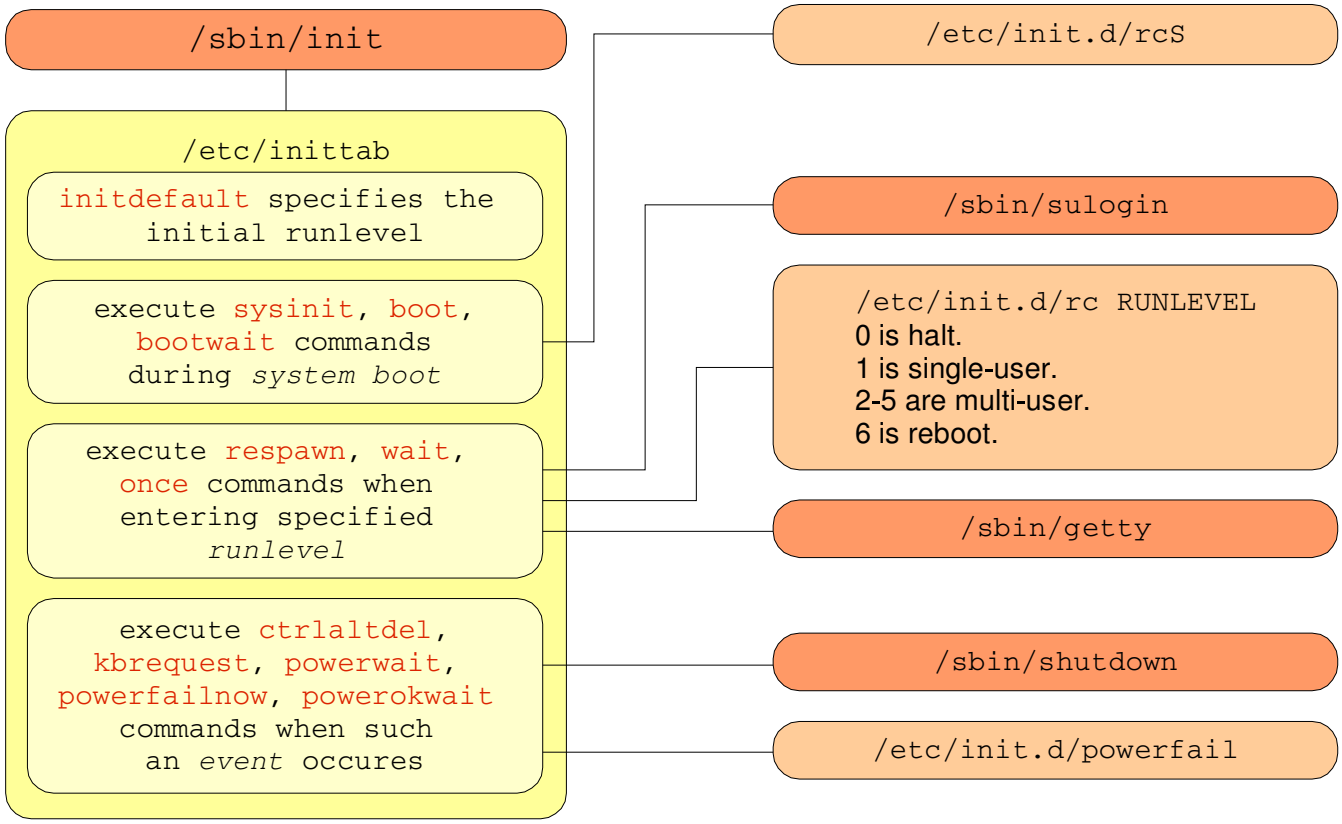
DESCRIPTION

When the Linux kernel has been loaded and the hardware initialized the kernel starts the init(8) process as the last step of the kernel boot sequence. Init is the parent of all subsequent processes. Init its primary role is to create processes from a script stored in the file `/etc/inittab` (see inittab(5)).

Inittab usually has entries which cause init to spawn gettys(8) on each line that users can log in. Further it defines a default runlevel and what to do when changing runlevels. Further init starts processes and is watching them. If one is terminating, it will restart it.

A runlevel is initalized by executing the run control (rc) script, named `/etc/init.d/rc`, which again executes many other scripts to complete. The run script executes the scripts in directory `/etc/rc?.d/, which` begin with `K` and `S`, where ? is the runlevel. K means kill and S means start. First it executes the kill scripts then the start scripts, both in alphabetical order. The scripts in `/etc/rc?.d/` are actually symbolical links, the real scripts are located in `/etc/init.d/`.

init(8), inittab(5) - sysv-compatible init process



init(8), inittab(5) - sysv-compatible init process

EXAMPLE

Find out the current runlevel. runlevel prints the previous and the current runlevel, while N means there is no previous runlevel.

```
$ runlevel
N 2
```

Change runlevel. Who is another common command to get the current runlevel.

```
# who -r
         run-level 3  Apr 26 21:13                   last=2
# init 2
# who -r
         run-level 2  Apr 26 21:14                   last=3
```

Re-examine /etc/inittab.

```
# init q
```

It is possible to pass a number of flags to init from the boot monitor. Boot into runlevel 1, regardless of the initdefault settings.

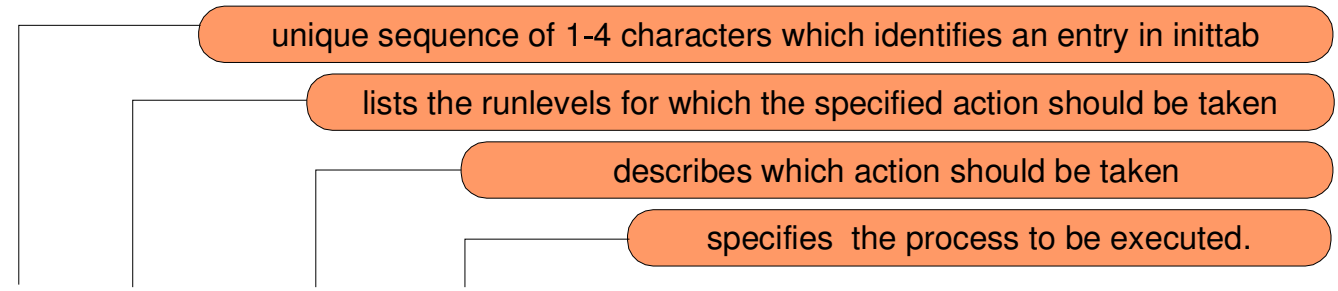
```
lilo: 1
```

Emergency, boot directly into a single user shell without running any other startup scripts.

```
lilo: emergency
```

init(8), inittab(5) - sysv-compatible init process

Format of the Inittab



EXAMPLE

```
id:3:initdefault:
si::sysinit:/etc/init.d/rcS
...
l3:3:wait:/etc/init.d/rc 3
...
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
...
1:2345:respawn:/sbin/getty 38400 tty1
```

cron(8) - scheduling commands (Vixie Cron)

DESCRIPTION

Vixie Cron is a daemon to execute scheduled commands. Cron searches its spool area (`/var/spool/cron/crontabs`) for crontab files (which are named after accounts in `/etc/passwd`); crontabs found are loaded into memory. Note that crontabs in this directory should not be accessed directly - the crontab command should be used to access and update them.

Cron also reads `/etc/crontab`, which is in a slightly different format (see crontab(5)). Additionally, cron reads the files in `/etc/cron.d`. Edit `/etc/crontab` with your favourite editor, don't use crontab(1).

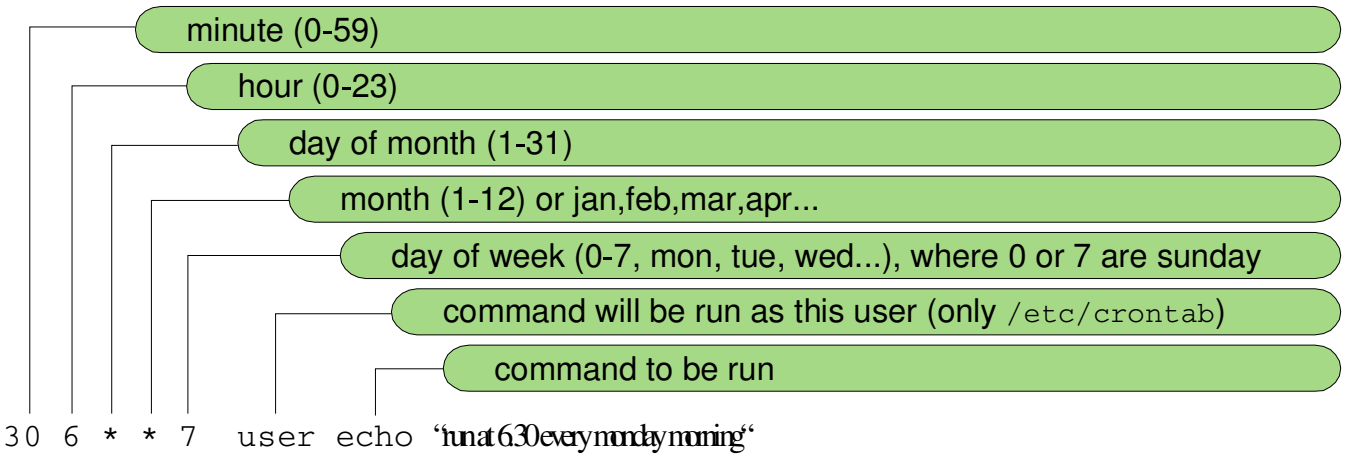
Cron then wakes up every minute, examining all stored crontabs, checking each command to see if it should be run in the current minute. When executing commands, any output is mailed to the owner of the crontab (or to the user named in the MAILTO environment variable in the crontab, if such exists).

Additionally, cron checks each minute to see if its spool directory's modtime (or the modtime on `/etc/crontab`) has changed, and if it has, cron will then examine the modtime on all crontabs and reload those which have changed. Thus cron need not be restarted whenever a crontab file is modified.

crontab(5) - cron configuration files

DESCRIPTION

A crontab file contains instructions to the cron daemon of the general form: ``run this command at this time on this date''.



A field may be an asterisk (*), which always stands for ``first-last''. Ranges of numbers are allowed. Ranges are two numbers separated with a hyphen. The specified range is inclusive. For example, 8-11 for an ``hours'' entry specifies execution at hours 8, 9, 10 and 11. Lists are allowed. A list is a set of numbers (or ranges) separated by commas. Examples: ``1,2,5,9'', ``0-4,8-12''. See crontab(5) for more options.

crontab(1), crontab(5) - cron configuration

EXAMPLE

Edit system wide crontab.

```
# vi /etc/crontab
```

Print user crontab of user joe.

```
# crontab -u joe -l
5 6 * * *       /usr/bin/fetchmail > /dev/null
```

Edit your user crontab.

```
$ crontab -e
```

Print your user crontab.

```
$ crontab -l
```

Remove your user crontab.

```
$ crontab -r
```

Logfile management

DESCRIPTION

The Linux system logs system messages into a set of files. These files are stored at

- `/var/log`.

There are two daemons responsible to log system and application messages:

- `syslogd`
- `klogd`

Note: Not all programs use the syslog daemon to log their messages, they sometimes write their own log files into /var/log instead.

EXAMPLE

Follow up the syslog on screen in realtime

```
# tail -f /var/log/syslog
Apr 26 12:00:01 base syslogd 1.4.1: restart.
```

Find errors within a logfile

```
# grep -i error /etc/log/messages | more
```

syslogd(8) - System loggin utility

DESCRIPTION

syslogd provides support for system logging. It allows local and remote logging. It logs system messages into a set of files described by the configuration file

`/etc/syslog.conf`.

Each message is one line in the log file and the messages are separated in 8 severity levels (priorities):

0	`emerg`	emergencies, panic messages
1	`alert`	alerts, that require emediate action
2	`crit`	critical errors
3	`err`	errors, non critical errors
4	`warnings`	warnings
5	`notice`	notifications, non error related
6	`info`	informational
7	`debug`	debugging

syslog.conf(5) - syslogd configuration file

DESCRIPTION

syslogd.conf is the main configuration file for the syslog daemon. This file specifies rules for logging. Every rule consists of two fields, a **selector** field and an **action** field. The **selector** field itself holds two components, a **facility** and a **priority**.
The following entry would write all messages with priority **error** and above into /var/log/messages:

```
*.err     /var/log/messages
```

where is:

`*.err`	The selector field. The symbol * is the facility and err is the priority.
`/var/log/messages`	The action field.

syslog.conf(5) - syslogd configuration file

DESCRIPTION

The facilities defines the subsystem (process) that produced the message, for example, all mail programs log with the mail facility.

facilities:

auth	mail
authpriv	news
cron	security
daemon	syslog
kern	user
lpr	uucp
mark	local0-7

Special characters within the syslog.conf:

*	stands for all facilities or all priorities
,	separates facilities with same priority
;	separates separators
=	to specify only a single priority and not any above
!	to ignore all that priorities

syslog.conf(5) - syslogd configuration file

DESCRIPTION

Actions of a rule define were to write the log message. A message does not need to be a real file. Syslog provides the following actions:

Regular File	A real log file. The file has to be specified by the absolute pathname.
Named Pipes	This will write to a fifo. The fifo must be created using mkfifo
Console	/dev/console
Remote machine	A remote host running syslogd. Put a @ in front of the hostname
List of Users	You may list the users separated by a ,

syslog.conf(5) - syslogd configuration file

EXAMPLE

All kernel messages go to /var/log/kernel

```
kern.*                  /var/log/kernel
```

All critical and above messages are send to /dev/console

```
kern.crit               /dev/console
```

All mail messages except for the info priority are send to host foobar

```
mail.*;mail.!=info      @foobar
```

All mail and news of priority info go to /var/log/info

```
mail,news.=info         /var/log/info
```

Send all messages to a remote host foobar

```
*.*                     @foobar
```

/var/log - the system log directory

DESCRIPTION

The `/var/log` is the default directory to write log files to. The following is a list of the some important or not self explaining log files of a regular Debian system (this list of log files is not complete):

`/var/log/auth.log`
Processes like *login, su* will write their authoritiy messages in this file.

`/var/log/syslog`
Everything (*.*) gets written into this file. This is a good file to search with grep for messages.

`/var/log/daemon.log`
Daemons like *init, inetd, sshd* write here.

`/var/log/kern.log`
All the kernel messages (like boot messages) will be written here.

`/var/log/messages`
Mail and news group messages