UNIX for Administrators

Edoardo Pugliese

USERS AND GROUPS

USER: one identified by number UID and they belongs to a primary group and ≥0 supplemental groups. UID = 0 = ROOT

GROUP: identified by group ID (GID) and contain users (≥0)

User's informations are stored in plain text file (/etc/passwd) in the form:

username : password : UID : GID : full name : home : shell

- full name = user's real name + supplementary infos
- home = user's home directory
- shell = path to shell started at login (/etc/shells)

This file must be readable for all users to map UIDs to usernames

New systems use "x" in place of password and store passwords in /etc/shadow (readable only for root) e.g. to avoid bruteforce attacks. There info are stored as follow:

username : password : last changed : min age : max age : warn before : lock after : acc. exp. : reserved

- last changed = last time password has changed
- min age = min day before user can change password
- max age = max days before password can be changed
- warn before : number of days before password expiry when system warns user
- lock after : number of days after ponword expiry when account is locked
- acc. exp : when account will be locked

Group's info are stored in /etc/group in this form: groupname : password : GID : user list

- user list = comma-separated list of users for whom this is a supplementary group.

Again we have /etc/gshadow → groupname : password:

eg. /etc/password

```
root:x:0:0::/root:/bin/bash
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
adm:x:3:4:adm:/var/log:
lp:x:4:7:lp:/var/spool/lpd:
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/:
news:x:9:13:news:/usr/lib/news:
uucp:x:10:14:uucp:/var/spool/uucppublic:
operator:x:11:0:operator:/root:/bin/bash
games:x:12:100:games:/usr/games:
ftp:x:14:50::/home/ftp:
mysql:x:27:27:MySQL:/var/lib/mysql:/bin/bash
gdm:x:42:42:GDM:/var/state/gdm:/bin/bash
nobody:x:99:99:nobody:/:
janko:x:1000:100:Janko Hrasko,,,:/home/janko:/bin/bash
```

eg /etc/shadow

```
root:nhC.YP4s8lF1Y:11783:0:::::
bin:*:9797:0:::::
daemon:*:9797:0:::::
adm:*:9797:0:::::
lp:*:9797:0:::::
sync:*:9797:0:::::
shutdown:*:9797:0:::::
halt:*:9797:0:::::
mail:*:9797:0:::::
news:*:9797:0:::::
uucp:*:9797:0:::::
operator:*:9797:0:::::
games:*:9797:0:::::
ftp:*:9797:0:::::
mysql:*:9797:0:::::
gdm:*:9797:0:::::
nobody:*:9797:0:::::
janko:in9.jjl2XgsXQ:11783:0:99999:7:::
```

How to create users

3 ways:

① manual edit of /etc/passwd, /etc/shadow and /etc/group be carefull!

② useradd [-c FullName] [-m] username
- -m = creates home directory
- -g grp = primary group
- -G grp[,...] = supplementary groups

and more...

③ Script adduser

Delete: userdel [-r] username
↳ delete also user's home directory

Edit: usermod options username
↳ -c FullName, -d home [-m], -g grp ...
-L (lock), -U (unlock)

passwd username = change password
- -l = lock password
- -u = unlock password

} root can change/lock/unlock all user's password

Some for groups:

① by hand editing /etc/group

② groupadd groupname

groupdel groupname = delete

groupmod -n newname oldname = rename

PROCESSES

Every process has:

UID
- ↳ real = UID of user who started process
- ↳ effective = UID of user who the process is using currently
- ↳ saved

GID: real, effective, saved

List of supplementary group IDs

PERMISSIONS

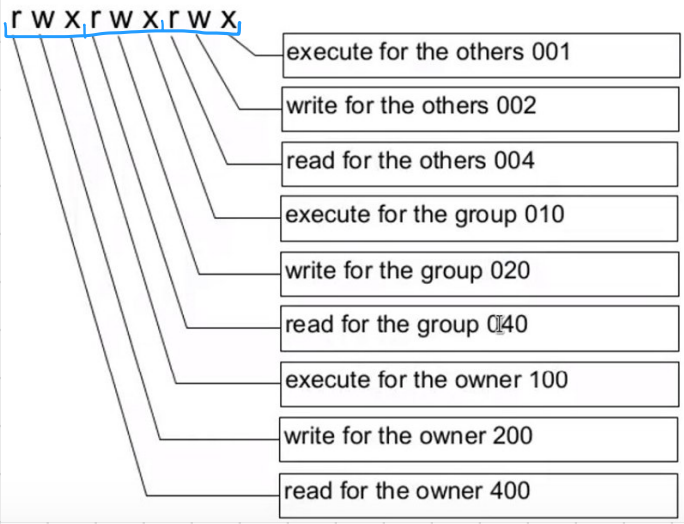
Every object in filesystem has an owner, a group and access rights (permissions): read, write and execute/search directory (for owner, group and others)

```
jerry@jerryntb:/$ ls -l
total 80
drwxr-xr-x    2 root     bin          4096 May  4  2002 bin
drwxr-xr-x    2 root     root         4096 Dec 29 08:28 boot
drwxr-xr-x    2 root     root         4096 May  4  2002 cdrom
drwxr-xr-x    1 root     root            0 Jan  1  1970 dev
```

rights
owner
group
type

d = directory

If UID of object's owner = effective UID → use owner's permissions
Else if GID = effective GID of process or GID of supplementary group of process → use group's perm.
Else → use other's permissions
If UID of process = 0 → no restrictions



eg. file: 511 (4+1) = all can execute, owner can also read
eg directory: 750 = group can read and use, owner also can modify, others can do nothing.

Change Permissions

Only root and owner can! → chmod rights objects
(rights → numbers)

or chmod whoOpRight [,...] objects
Who: u = owner, g = group, o = others, a = ugo (all)
Op: "+" = add, "-" = remove, "=" = set
Right: r, w, x
-R = recursively apply to subtree

```
chmod 640 file          chmod u=rw,g=r,o= file
chmod 711 file          chmod u=rwx,go=x file
chmod 711 file          chmod a=rwx,go-rw file
chmod 711 file          chmod a=x,u+rw file
```

Change owner: name or GID of group
- chown [-R] owner[:group] object ... ONLY ROOT CAN DO THIS
  ↓ username or UID

Change group
- chgrp [-R] group object Root can change all groups, owner only groups he is in.

When file/directory is created:
- owner = UID of creating process
- group = effective GID of creating process or group of parent directory (BSD)

Rights can be modified by umask: eg umask 022 → group and others are denied to write (2=write)

Some programs needs other type of rights eg passwd needs to write to /etc/passwd
- set-UID permission bit: user will be able to execute file with some rights of owner.
  - chmod: 4000 / chmod u+s → rwsr-xr-x → lowercase = x included capital S = no x
  - the process will have its saved UID and its effective UID equal to executed file owner's UID
  - the process can switch effective UID between its real and saved UID
- set-GID permission bit:
  - chmod 2000/g+s → rwxrwsr-x
  - process will have saved GID and effective GID equal to GID of executed file's group
  - process can switch effective GID between real and saved GID

↳ set-GID permission on directories (SysV): created files and subdirectories will have their group equal to directory's group and subdirectory will have their set-GID bit set.

- sticky bit on directories
  - chmod: 1000/+t → rwxrwxrwt
  - an object in this directory can be removed only by owner, owner of directory or root. Standard rights are applied too → need w right. (/tmp has rights 1777)

UNIX DIRECTORY TREE

- /dev - character and block devices
- /etc - configuration files
- /bin - basic system programs for normal users
- /sbin - // for admins
- /lib - basic system shared libraries
- /tmp - temp files
- /boot - kernel and files for boot loader
- /proc - system information and parameters
- /var - varying files, queues, logs etc.
- /root - root's home directory
- /home - home directory for normal users. → divide in sub-dir if users are a lot
- /mnt - mount-point for temp filesystems
- /opt - dir for optional subsystems
- /usr/bin - most app. executables
- /usr/sbin - supplemental programs for admins
- /usr/include - headers for c/c++ programs
- /usr/lib - libraries
- /usr/shared - shareable files across architectures
- /usr/local - hierarchy for locally installed software (eg compiled source code)
- /usr/X11 - X windows (in some systems)
- /media - directory for mounting removable media
- /sys - Linux sysfs = special filesystem to access information and controll some devices and drivers

UNIX FILESYSTEM

i-NODE

i-node is a data structure with information about a filesystem object such as type, size, owner, group, permissions etc.
Every object in the filesystem has a physical link to its inode (with an identification number). When accesing a file the o.s. search for its inode to find required info such as permissions.
There can be unlimited links to an object → object can be removed when links from directory or open files are zero.

One link is created when object is created. Others can be created with commond:
- ln existingObject dest → dest link created

  or
- ln ex1 ex2 ... directory → creates multiple links with some names of existing objects in that directory

SYMBOLIC LINK

Also called soft link or symlink. Special file containing a path to other object. Common operations except remove are performed on on object that symlink points to
REMOVE: just remove the link.
They have no permissions.
BROKEN LINK = link to non-existing object

Symlink can be created with commond:
- ln -s path dest = creates symlink dest containing specified path
- ln -s path1 path2 ... directory = creates symlinks in specified directory with names path1 path2...

Path can be:
relative = relative to symlink
absolute = starts with /

Symlink vs. hard link

- symlink can point to a directory
- symlink can point to a different filesystem
- symlink does not prevent you from removing the object – it has no influence on the number of links in the i-node
- symlink can point to a non-existent object
- symlink can be differentiated from the primary object

PIPE (FIFO) = communication channel between processes (one way). One opens it to write, the other to read.
mkfifo [-m mode] name ...
↳ permissions, default 666

BLOCK & CHARACTER DEVICES

Special files representing most devices (not network interfaces). located in /dev and identified by:

- major number = similar to group ID
- minor number = similar to UID

mknod [-m mode] name b|c major minor
↳ block or character

BLOCK: basic unit is a block (eg. CD, floppy, ramdisk ...), can contain filesystem
CHARACTER: basic unit is a character / byte (eg. serial port, printer, console ...)

Examples (Char. Devices)

- /dev/null: read = empty file, write = throw away all data (like sending data to the thrash bin)
- /dev/zero: read = infinite file of zero bytes (to create files of different size)
- /dev/full: read = /dev/zero, write = full disk (to test program under full disk condition)
- /dev/random: random generator output, when runs out of entropy it blocks until new entropy is gathered.
- /dev/urandom: PSEUDO random gen., doesnt block
- /dev/tty: current controlling terminal of a process, the shell/console where the process has been started
- /dev/tty1, /dev/tty2: virtual consoles
- /dev/console, /dev/tty0: current virtual console
- /dev/ttyS0, /dev/ttyUSB0: serial ports
- /dev/lp0: printer of first parallel port → not used anymore
- /dev/ptmx, /dev/pts/0: pseudoterminal devices
- /dev/tty??, /dev/pty??: pseudoterminal devices (old)

Examples (Block Devices)

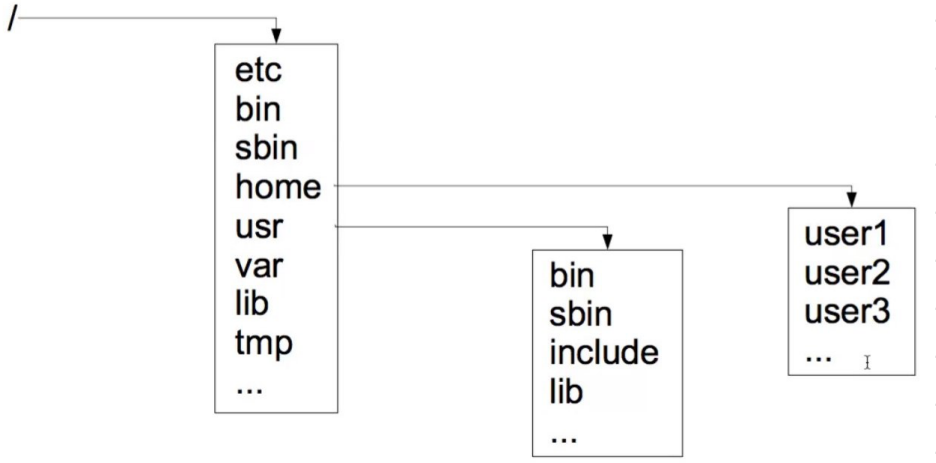
- /dev/sda, /dev/sdb ...: disks (SATA, USB etc)
- /dev/sda#: #1-4 = primary partitions of /dev/sda #5-... = logical partitions
- /dev/fd0: floppy
- /dev/hda, /dev/hdb ...: IDE disks and CD-ROMs with older drivers
- /dev/sr0, /dev/sr1 ...: CD/DVD
- /dev/sg0, /dev/sg1 ...: SCSI generic device
- /dev/c0t1d0s2: (Solaris system) SCSI device, controller 0, target 1, LUN 0, slice 2
- /dev/ram0, /dev/ram1 ...: ramdisks
- /dev/loop0, /dev/loop1 ...: allow to access every file as block device, losetup /dev/loop0 file

Difference between /dev/null and /dev/zero: the first one produce no output (null), the second one produce a continuous stream of NULL (zero) bytes

Difference between /dev/random and /dev/urandom: second one is non-blocking and "less random" than the first one.

MOUNTING FILESYSTEMS

Direct acyclic graph for file access in UNIX. Filesystems are mounted on directories (called mount points). Original content of dir. become temporary inaccessible and replaced with root of mounted filesystem



Separate directories in different mounting points (eg users for "backup").

Mount: mount [params] block_dev dir - mount filesystem on block device to directory
Unmount: unmount {block_dev | dir} = unmount filesystem on given block device mounted on given directory
Params:
-a [-t type] = mounts all filesystems of given type in /etc/fstab without noauto attribute
-r, -w - mount read only or read-write
-n = dont update /etc/mtab (list of mounted filesystem)
-o opt1 [, opt2 ...] = various options

File /etc/fstab contains info about filesystems used by mount:
block_dev mount_point type options fs_freq pass_no
fs_freq = used by some backup systems
pass_no = order of checking FS on boot (0 = without)

Options:
- sync/async : sync/async writes , the first one assure the write after confirmation, second increases perform.
- [no] atime : update last access time
- [no] auto : automatically mount (by mount -a)
- [no] dev : FS can contain block/char devices
- [no] exec : FS can contain executable files
- [no] suid : FS can contain setUID and setGID programs
- [no] user : FS can be mounted by a user (implies noexec, nosuid, nodev)
- ro : FS is read-only
- rw : FS is read-write
⇒ defaults : rw, suid, dev, exec, auto, nouser, async.

- remount : edit params of mounted FS

Most common FS:
- proc: store no data, present an interface to system parameters and about running processes. (/proc)
- tmpfs: temporary storage (in memory, no block device)

eg.

- `mount -t vfat /dev/sda1 /mnt`
  - mounts the vfat FS from /dev/sda1 to /mnt
- `mount /cdrom`
  - mounts the FS according to /etc/fstab on /cdrom
- `mount -o ro,remount /`
  - makes the root FS read-only

Create: mkfs -t type [fs-option] block_dev

SYSTEM STARTUP

BIOS/Firmware loads and starts a bootloader who loads kernel of OS, passes params to it and starts it.
The kernel initialize necessary devices, mounts filesystem on / and executes /sbin/init as first process (init reads /etc/inittab)

Boot is characterized by runlevel (0-6) that defines processes that can run on it (eg. 2,3,4 = multi user mode)

/etc/inittab = defines command/processes to execute at boot, id:runlevels:action:process
- id = unique line identifier
- runlevels = list of runlevels to which this line applies
- action = what to be done
- process = command to execute

Actions:
- initdefault = specify default runlevel, processes are ignored
- sysinit = the process will start on start-up, init will wait its termination, runlevels ignored
- bootwait = same as sysinit but after it, runlevels not always ignored (depends on system)
- boot = like bootwait but init doesnt wait its termination.
- wait = process started on entry to runlevel and init waits its termination
- once = init doesnt wait
- respawn = started again after its termination (to keep process running)
- off = no action
- ctrlaltdel = process started when press ctrl+alt+del
- powerwait = process start on power failure and init waits it
- powerfail = same but no waiting
- powerok = when power restored
- powerfailnow = on low battery signal

"recovery"

Runlevel can be changed:
- using telinit new_level
- using shutdown -h(=0) / -r(=6) (1 otherwise)

When level is changed, init will kill all processes that are not in the right level and starts the ones they are.

Note:
- Folder /etc/init.d contains scripts that can be started at startup
- Folder /etc/rcX.d contains symlink to these scripts
  X = runlevel number

These symlinks are named SnnName and KnnName according to the fact that the script is enabled or not. (S = start, K = kill)

JOB SCHEDULING

at: irregular Job scheduling
at time ... = reads commands to be executed from standard input at specified time
eg. at now [+ increment]
at [time, date] → nameOfMonth day [Year] / dayOfWeek ecc
↳ hh:mm / noon / midnight
eg at now + 2hours = after 2 hours from now
at 16:00 next monday = next Monday at 16:00

cron: regular Job scheduling, Job started every min hour month year dayOfWeek
eg. -10 6 * * * = every day at 6:10
dayOfWeek → 0 = sunday 1 = Monday etc.
eg - * 7-9 * * * = every minutes between 7 and 9:59
eg - */x 5 * * * = every x minutes between 5 and 5:59

Submit a table: crontab file [-u username]
-e [username] = edit
-l ["] = list current table
-d ["] = delete current table

Process crond manage execution of jobs of cron subsystem (/var/spool/cron/crontabs)
For "at" instead: atd process or atrun (executed by cron)

NETWORKING

Most used is TCP/IP protocol. Unix connects to networks using network interfaces.
They can be permanent (eg. network cards), dial-up connections (eg. logical network)

IP address = 4 bytes / 32 bit = network address + host id of machine
Network Mask = specify network address part eg 255.255.254.0 → first 23 bit are network address

ROUTING TABLE: determines where to send packet in next step based on destination IP address. First matching rule is used (if more match the packet)
eg IP, MASK, ROUTER, INTERFACE

NETWORK CONFIGURATION

ifconfig
- -a : show all interfaces
- interface IP_add/mask : configure new IP address

up / down = activate / deactivate interface
pointopoint address = configure the other side's address of a point-to-point link
hw ether address = configure link-layer address

For routing table management:
route : lists routing table
route add -net dest netmask mask : add record for a network to the routing table
- -host dest : add record for single host

del -host/net ... : remove record

Editing in this way, changes are not permanent → use script at boot!
Writing 1 to /proc/sys/net/ipv4/ip_forward enables routing in Linux, 0 disables it.

> **Route**: destination-gateway pair that indicates that if we want to reach the given destination, packets must communicate via given gateway

/etc/network/interfaces on Debian is a space separated list for interfaces: auto = configure on startup, allow-hotplug = configure at creation
iface interface protocol-family method param
↳ eg inet (i.e. ipv4) ↳ /etc/network/if-param.d/ = script execute after the specified command

REMINDER: given IP address and subnet mask, how to retrieve hosts range, network address and broadcast address?
eg IP 158.135.22.128
MASK 255.255.255.128 (= /25) → 25 bit of IP are network address
IP in binary = [10011110.11000011.00010110.]0000000
network address = 158.192.22.128 i.e. take the 25TH bit (1) and add zeros to complete the octet in this case the number is the same
broadcast address = 158.192.22.255 i.e as network address but complete the octet with ones
Host min = Network address + 1 = 158.192.22.129
Host max = Broadcast address - 1 = 158.192.22.254

Note: given a range of IPs to calculate mask must calculate $32 - \log_2 X$ with X = IP range eg [.1, .15] → x = 16

NETWORK SERVICES

/etc/services – contains mapping between service name and port numbers/protocols
- name portnumber/protocol alias

Protocols are mapped to names in /etc/protocols ← Edit this first when creating new service! Then "Restart" the file with KILL -HUP inetd

Programs can be started by "superserver" inetd configured in /etc/inetd.conf
name type protocol wait/nowait user prog args
- name = service name according to /etc/services
- type = dgram or stream
- protocol = udp or tcp
- wait = wait process' termination before processing new requests on port
- nowait = new instance of program for every request (most used)
- user = username of the user to run the program as
- prog = path of executable program/file
- args = arguments (including the 0TH one – the name of program)

File /etc/hosts is the original solution for small networks for mapping IP addresses with name (and alias).
Problem: what to do in large networks and how to assign unique names to IP addresses.
⇒ Domain Name System: largest distributed database for conversion between domain names and IP addresses.
Top level (1) domains: com, net, org etc.
DNS is essentially a tree in which each node is called domains. Domains can be subdivided into zones
– Zone file format:
  - $ORIGIN domain: specifies suffix for relative domain names (ends with dot if absolute domain name)
  - $TTL time: default time-to-live for the records
– domain [TTL] class type data:
  - class = IN (internet)
  - domain name
  - type (of the record)
  - data = value of record
  - TTL
– DNS record types:
  - SOA ... = basic zone information
  - A ... = ip address for domain name
  - CNAME = canonical domain name for the domain name of the record (alias)
  - NS = delegation to another DNS server
  - MX = specifies mail-exchanger for the domain (sorted by priority)
  - PTR = inverse mapping (from IP to domain names)

File /etc/resolv.conf contains the DNS resolver (client) configuration.

Back in our system. Files /etc/hosts.deny and /etc/hosts.allow define rule that selectively allow or deny clients access to server daemons
Format: daemon_list : client_list [: command]
① comma-separated list of daemon or ALL
② same but for IP
③ optional command executed when client try to access

eg. in hosts.deny : [in.telnetd] : ALL
all client are denied to use service telnet
[...] = args field from /etc/inetd.conf = the daemon

NETWORK FILESYSTEM (NFS)

Standard means of file sharing in UNIX system based on RPC protocol. Can also transfer info about owners, groups and permissions. Use mount command like a common FS (with fixes)
mount -t nfs server:path dest
Can be specified other parameters to control behaviour of NFS in case of errors.

Shared directories are specified in /etc/exports (each line is a folder).
Each directory has a list of allowed clients associated with it (with options about permissions).
eg path allowed_clients (IP/mask) (options eg ro ...)

LINUX ACCESS CONTROL with ACLs

We already know standard permissions: read, write, execute/use
Problems:
- too coarse structure of subjects
- can't set default permissions (umask is not perfect)

Access Control List (ACL)

- More subjects: specific user and specific group
- Every filesystem object is assigned a list of entries: entry's type, identifier of user/group, permissions.
- Specify object permissions in directory

- entry types:
  - ACL_USER_OBJ – the permissions for the owner
  - ACL_USER – the permissions for a specified user
  - ACL_GROUP_OBJ – the permissions for the object's group
  - ACL_GROUP – the permissions for a specified group
  - ACL_OTHER – the permissions for the others
  - ACL_MASK – the maximal permissions for ACL_USER, ACL_GROUP a ACL_GROUP_OBJ
- 1, 0 or mode, 0 – 1 (1 if a green exists)

Order:
1) Process UID = owner UID
2) Process UID = specific user UID
   (ANDed with ACL_MASK)
3) All groups permissions (ORed together)
   ANDed with ACL_MASK
4) ACL_OTHER

Group permissions (classic one) corresponds to ACL_GROUP_OBJ if ACL_MASK is not present, ACL_MASK otherwise

Long version: one entry per line, type:id:permissions
Permissions = 3 from {r, w, x, -}
Types:
- user = ACL_USER or ACL_USER_OBJ if ID=""
- group = ACL_GROUP or ACL_GROUP_OBJ if ID=""
- mask
- other

Short version: commaseparated, types abbreviated with u, g, m, o, - can be omitted

To show ACL permissions: getfacl object
To set ACL permissions: setfacl -m acl object [-R]
↳ short version ↳ recursively
-x = remove specified entry → acl = specification without permissions
setfacl [-R] -b object → remove all entries
-n = by default it sets mask as OR of all ACL_USER, ACL_GROUP and ACL_GROUP_OBJ if not specified. Switch -n avoid this.

New objects' ACL are initialized from ACL of directory. Permissions not requested on creation are removed from entries. If directory has no default ACL, their ACL will contain entries corresponding to standard permissions initialized according to the requested permissions and umask.
Setting default ACL: setfacl -d ...

-k = remove default ACL

For mounting devices with ACL, use -o acl.

FIREWALLING

- On host: access control to network services and restriction of outgoing communication
- On router: restriction of communication between networks and Network Address Translation (convert private IP for in/out connections)

Firewall can be:
- stateless - handle each packet separately, low CPU usage, only simple security policies
- statefull - keep info about "connection", high CPU usage, complex policy

Part of linux kernel is dedicated to netfilter subsystem that provides tools for statefull and stateless firewall.

TABLE

Tables contain chain of rules:
- filter
  - INPUT
  - OUTPUT
  - FORWARD

  } incoming, outgoing and forwarded packets
- nat
  - PREROUTING: before routing
  - OUTPUT: procen packet generated locally
  - POSTROUTING: after routing (just before transmission)

According to rule the packet can be ACCEPTed or DROPped.
User can also create custom chains.

For base syntax check CNS notes.
In addition:
--state RELATED: first packet of a connection related to an existing one (i.e. expected)
--state INVALID: packet can't be identified
REJECT (target): drop packet and responds with an ICMP error message.
REDIRECT (target): redirect packet to specified port (for transparent proxy servers)
SNAT / DNAT (target): rewrite source / destination address to the specified one in POSTROUTING / PREROUTING, OUTPUT chain in NAT table

eg REJECT: iptables -A INPUT -m --state NEW -j REJECT --reject-with icmp-port-unreachable

FUSE - FILESYSTEM IN USERSPACE

Idea: allow implementation of filesystem using ordinary process. It is easier to implement filesystem "drivers" (ordinary program) than kernel modules. Allow normal users to mount filesystem to the tree.
A single kernel module, Fuse module, passes requests from mounted filesystem to the process which implements the filesystem operations.
No need to work on /etc/fstab!
eg. sshfs: allow user to mount remote PC's filesystem, accesible via SFTP
mount → username@host:path mountpoint
unmount → fusermount -u mountpoint
↳ helper program to support (un)mounting

A process that wants to mount a filesystem to a mountpoint must have full access permision on mountpoint.
Fuse ignores permision checks in the mounted filesystem by default → access control up to the process implementing the filesystem (-o default_permissions enables standard access control in kernel).
Mounted filesystem is inaccesible for other users by default (not even root) (-o allow_other /allow_root, /etc/fuse.conf to enable these two options for all normal users)

LINUX BOOTING

STANDARD PROCEDURE: BIOS loads and starts a boot loader, bootloader loads the system kernel and CPIO archive with the contents of initial root filesystem (initramfs). Then starts the kernel which extracts initramfs to / and starts /init. /init mounts the real root filesystem and starts /sbin/init

Nowadays Kernel is usually universal, the drivers are loaded from initramfs before mounting real root fs.
/etc/initramfs-tools/... for configuration.
Is possible to gives some parameters to the bootloader to configure it:
init = ... → file to use instead of /sbin/init
rdinit = ... → file to use instead of /init
root = ... → device containing root fs.
- /dev/sda2
- UUID = 2028zab2- .... universal UID of fs
- LABEL = root label of fs
- /dev/nfs

HOW BIOS LOADS BOOTLOADER

BIOS reads 1st disk sector (MBR) and runs it (max 446B of bootloader, rest is partition table)
Standard PC bootloader in MBR loads 1st sector of active partition and runs it (kernel of os or rest of longer bootloader)

syslinux.cfg: allow user to specify parameters for the kernel on a command line. Can be used to create simple boot menu.

SYSTEMD

Global service manager (replace /sbin/init) and also per user service manager.
In /etc/systemd/ system.conf and user.conf can be configured with various default values.

UNIT

Work with systemd. They are text files describing: a service (replacement of init), a socket (replacement of inetd), a device etc...

eg.
[Unit] ~ sections
Key = value → eg. Description = text description of unit
...
[Install]
...
[Othersection]
...

Documentation = URI list of documentation
Wants = list of units started with this one, also unit.wants may contain symlinks to wanted unit files
Requires = stronger than wants, if required units fail/stop, the current one will not start/will be stopped
And many more ...

SERVICE UNIT

Uses .service suffix and contain [Service section]
Type = simple, exec, forking, oneshot etc.

COMMAND RECAP

REMEMBER TO SET ROOT PASSWORD!

- Create user (with home directory): useradd -m name
- Create group: groupadd name → -G groupname (create user with given group)
- Set/Change password: passwd username
- Add user to supplementary group: usermod -G groupname username (-g to change main group)
- Edit permissions: chmod xyz target xyz = { 2 = write, 1 = execute, 4 = read, 0 = none }

Note: execute permission on directory should be preserved → can't see content or write inside

- Only owner of file/directory or root can delete → Sticky bit: chmod xyz +t (-t to remove it)
- File and subdirectory inherit directory group (not permission, set umask for them or do it manually): chmod xyz g+s
- Create hard/soft link: ln -s source destination
  - Hardlink: delete, rename, move original file does not effect hardlink, not directories
  - Softlink: delete, rename, move original source makes link useless
- Show device content in a specific directory: mount device_path (/dev/ubdc) dest_path [-o ro etc]
- Edit mount settings/permissions → edit file /etc/fstab (remember FS type)
  - ↳ mount -a to apply changes / see errors
  - eg. /dev/ubdc /mnt iso9660 user,ro... 0 0 → can mount with "mount /dev/ubdc" or "mount /mnt"
- Create device: mknod dest b|c major minor
  - ↳ ls -l in /dev to show other devices info.
- Make device "usable": mkfs -t ext2 device
  - ↳ example of FS type
- Boot script → /etc/inittab : unique_id : runlevels_list : trigger_action : command_to_execute
- Runlevels Script: /etc/rcX.d/ → symlinks to /etc/init.d scripts (Snn = active, Knn = killed)
  - ↳ can be changed with command telinit level_number
- Schedule a command:
  - Once → at date / at now [+increment] } command from standard input done a single time; atq = list of planned jobs. atrm = remove job from list
  - Repeated → crontab → /etc/crontab
- Configure network interface: ifconfig interface IP/mask_bits
  ifconfig interface IP netmask mask broadcast brc_address

Check back for how to calculate broadcast address etc.

Note: ifconfig is settings are not permanent → edit /etc/network/interfaces

eg.
```
auto interface
iface interface inet static
      address 10.0.0.175
      netmask 255.255.255.0
     [gateway ... ]
```

- Routing = given a destination where should packets go?: route add -net IP netmask MASK gw GATEWAY
  -host if single IP

Set default gateway: route add default gw gw_address

- Network Services:
  ① Create service in /etc/services in form name port/protocol
  ② Configure program in /etc/inetd.conf → see page 14 to learn how!
  ③ Restart inetd with command kill -HUP inetd

  /etc/hosts.deny or .allow to define who is allowed or not to execute services in form
  daemon_list : client_list
  ↳ from inetd.conf ↳ IP or ALL

- NFS
  Mount specifying nfs as system type (may need -o vers=3)
  Your shared directories are defined in /etc/exports in form path allowed_clients (IP/mask) + options eg ro

- ACL
  • show permission: getfacl obj
  • set permission: setfacl -m u:user:perms ... obj [-R]
  • set default permissions: setfacl -d ...