UNIX AND LINIX
IN INFOCOMMUNICATION
Week 8

O. Sadov

Users and groups

As we remember, users are one of the three pillars on which the UNIX world stands.

You can use some graphical interfaces to manage users and groups, but simple CLI utilities are often more convenient. There are:

- `adduser`, `useradd` — create a new user or update default new user information
- `groupadd` — create a new group
- `passwd` — update user's authentication tokens

To create a new user, we (as 'root') simply have to run the program '`adduser`' and set a password with '`passwd`'. But it's not over yet!

Actually, adduser is also black magic — in fact, all data related to users and groups is placed in common text files that can only be modified with ordinary text editors:

```
less /etc/passwd
```

The file format is quite simple — one line per user with colon-separated fields:

```
$ man 5 passwd
```

The fields, in order from left to right, are:

1. User name: the string a user would type in when logging into the operating system: the logname. Must be unique across users listed in the file.
2. Information used to validate a user's password; And at the very beginning, the password data was actually placed in this field. But we can read this file as a regular user, this is a design requirement. Did users have the ability to read passwords of other users at this time? Not. In Robert Morris and Ken Thompson's classic article "Password Security: A Case History" about the UNIX password system, Morris described a real-life incident he himself saw:

> Perhaps the most memorable such occasion occurred in the early 1960s when a system administrator on the CTSS system at MIT was editing the password file and another system administrator was editing the daily message that is printed on everyone's terminal on login. Due to a software design error, the temporary editor files of the two users were interchanged and thus, for a time, the password file was printed on every terminal when it was logged in.

And the main idea of UNIX passwords is not to believe that you can simply hide them. Better not to save passwords in the system at all. Actually, when creating a password, a random code was simply generated (the so-called SALT code), and then from this code and password by means of a one-way 'crypt' procedure with the DES algorithm:

```
man crypt
```

And the result of this operation cannot be decrypted (actually, we received some kind of hash) — when entering the system, the system receives SALT from the password field, encrypts it with the entered password, and simply compares it with the contents of the password field.

In most modern uses, this field is usually set to "`x`" (or "`*`", or some other indicator) with the actual password information being stored in a separate shadow password file. On Linux systems, setting this field to an asterisk ("`*`") is a common way to disable direct logins to an account while still preserving its name, while another possible value is "`*NP*`" which indicates to use an NIS server to obtain the password.[2] Without password shadowing in effect, this field would typically contain a cryptographic hash of the user's password (in combination with a salt).

3. User identifier number, used by the operating system for internal purposes. It need not be unique. Moreover, a superuser is simply a user with a zero user ID, and you can have multiple superusers in addition to the traditional "root" superuser. For example, you can create some superuser with UID 0 and a name like "`halt`" and with the command "`shutdown`" as a shell for the user, and provide a password for that user to anyone who needs to shutdown the system at night.

4. Group identifier number, which identifies the primary group of the user; all files that are created by this user may initially be accessible to this group. You can change this default during the current session with the command ‘`newgrp`’.

5. Gecos field, commentary that describes the person or account. Some early Unix systems at Bell Labs used GE/Honeywell mainframe computers with General Comprehensive Operating System (GCOS) for print spooling and various other services, so this field was added to carry information on a user’s GECOS identity.

   Typically, now this is a set of comma-separated values including the user’s full name and contact details which may be used by some commands for example by mail user agent.

6. Path to the user’s home directory.

7. Program that is started every time the user logs into the system. For an interactive user, this is usually one of the system’s command line interpreters (shells). For example, for pseudo-users who do not need interactive sessions, this could be ‘`nologin`’ or just ‘`false`’ executables, which will exit immediately upon startup.

The description of the groups is also placed in a text file:

```
less /etc/group
```

In this file, we see a similar format:

```
man 5 group
```

1. group_name — the name of the group.

2. password — Password field that has never been used

3. GID — the numeric group ID.

4. user_list — a list of the usernames that are members of this group, separated by commas.

Finally, a file with real data regarding passwords:

```
ls -l /etc/shadow
```

As we can see, only the superuser has access to this file. The transfer of password hashes from '/etc/passwd' to this file was carried out to prevent brute-force attacks using modern computing equipment, which is now becoming cheaper and cheaper. And we can see the hashes of the passwords in the second field of the records for each user:

```
man 5 shadow
```

A password field which starts with an exclamation mark means that the password is locked. The rest of the characters in the string represent the password field before the password was locked, and you can simply remove the exclamation mark to unlock it.

On a multiuser system with many administrators, it is advisable to use the 'vipw' and 'vigr' commands to avoid conflicts when multiple administrators are editing the same file at the same time:

```
man vipw
```

This file-based machinery for handling of user accounts is not hardcoded. You can switch to network authentication services such as LDAP or Winbind using the setup utility:

```
setup
```

or simply by editing the text configuration file '/etc/nsswitch.conf'

```
less /etc/nsswitch.conf
```

Other security related settings on Linux systems can be done in the '/etc/security' and PAM configuration directories:

```
ls /etc/security/
ls /etc/pam.d/
```

As we can see, the UNIX system administration paradigm does not hide the details from the user, everything can be configured by hands or scripts. For beginners, such systems simply provide more or less user-firendly tools and wizards to lower the barrier to entry.

Partitions

As storage systems grow, they need to be separated to store different data. And for this the partitioning was invented. There are different partition schemes developed by different vendors like IBM, Apple, Microsoft, etc.

In common PCs the Master Boot Record (MBR) partitioning scheme, widely used since the early 1980s, imposed limitations for use of modern hardware. A major deficiency is the limited size of 32 bits for block addresses and related information. For hard disks with 512-byte sectors, the MBR partition table entries allow a maximum size of 2 TiB. Also, the standard partitioning scheme only supports 4 primary partitions, and as the disk space increases, it will become necessary to implement such complex solutions as extended and logical partitions.

In the late 1990s, Intel developed a new partition table format as part of what eventually became the Unified Extensible Firmware Interface (UEFI). As of 2010, the GUID Partition Table (GPT) forms a subset of the UEFI specification. GPT uses 64 bits for logical block addresses, allowing a ZB disk size. Number of partitions — Depends on the space allocated for the partition table. By default, the GPT contains space to define 128 partitions.

Different systems use different naming schemes for devices and partitions. Modern Linux, for example, has special /dev/sd files for SCSI or SATA devices with a naming schema like this:

```
ls /dev/sd*
/dev/sda /dev/sda2 /dev/sda4 /dev/sda6 /dev/sdb /dev/sdb2
/dev/sda1 /dev/sda3 /dev/sda5 /dev/sda7 /dev/sdb1 /dev/sdb5
```

The letter “a” stands for the first device on the bus, and the numbers are the partitions. We can also access disk devices by disk labels:

```
ls -l /dev/disk/by-label
```

BSD disklabels, which also used on many commercial UNIXes, traditionally contain 8 entries for describing partitions. These are, by convention, labeled alphabetically, ’a’ through to ’h’. Some BSD variants have since increased this to 16 partitions, labeled ’a’ through to ’p’.

Also by convention, partitions ’a’, ’b’, and ’c’ have fixed meanings:

- ‘a’ is the “root” partition, the volume from which the operating system is bootstrapped. The boot code in the Volume Boot Record containing the disklabel is thus simplified, as it need only look in one fixed location to find the location of the boot volume;
- ‘b’ is the “swap” partition;
- ‘c’ overlaps all of the other partitions and describes the entire disk. Its start and length are fixed. On systems where the disklabel co-exists with another partitioning scheme (such as on PC hardware), partition ’c’ may actually only extend to an area of disk allocated to the BSD operating system, and partition ’d’ is used to cover the whole physical disk.

On Linux, MBR related tools are:

- ‘`fdisk`’ — is a simple text-based tool:

  ```
  # fdisk /dev/sda
  Command (m for help): m
  Command action
     d delete a partition
     g create a new empty GPT partition table
     G create an IRIX (SGI) partition table
     l list known partition types
     m print this menu
     n add a new partition
     o create a new empty DOS partition table
     p print the partition table
     q quit without saving changes
     s create a new empty Sun disklabel
     t change a partition's system id
     v verify the partition table
     w write table to disk and exit
     x extra functionality (experts only)

  Command (m for help): p
  ...
  ```

  The most useful operations: `m`, `p`, `n`, `t`, `d`, `q`, `w`.

- ‘cfdisk’ — is a fullscreen program text-based variant of ‘fdisk’.
- ‘sfdisk’ — non-interactive variant of ‘fdisk‘, it’s useful for scripting.

Partition management programs that support GPT:

- ‘parted’ — GNU Parted.

```
# parted /dev/sda
GNU Parted 3.1
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) help
  align-check TYPE N check partition N for TYPE(min|opt) alignment
  help [COMMAND] print general help, or help on COMMAND
  mklabel,mktable LABEL-TYPE create a new disklabel (partition table)
  mkpart PART-TYPE [FS-TYPE] START END make a partition
  name NUMBER NAME name partition NUMBER as NAME
  print [devices|free|list,all|NUMBER] display the partition table, available
        partitions, or a particular partition
  quit exit program
  rescue START END rescue a lost partition near START and END

  resizepart NUMBER END resize partition NUMBER
  rm NUMBER delete partition NUMBER
  select DEVICE choose the device to edit
  disk_set FLAG STATE change the FLAG on selected device
  disk_toggle [FLAG] toggle the state of FLAG on selected device
  set NUMBER FLAG STATE change the FLAG on partition NUMBER
  toggle [NUMBER [FLAG]] toggle the state of FLAG on partition NUMBER
  unit UNIT set the default unit to UNIT
  version display the version number and copyright information of GNU Parted
(parted)
```

- ‘gparted’ — GUI variant of ‘parted’.

File systems

And now that our partitioning is complete, it's time to see how we can use our disk space. As mentioned earlier, UNIX-like systems can treat disks or disk partitions as files. And some databases, for example, can use raw disk partitions to store data with higher performance.

But most of the time, disk space is used in file systems. UNIX-like systems and especially Linux support many different file systems. All the details of their implementation are reduced to a common denominator — the VFS abstraction. Then we can mount them in a single directory tree, navigate through the hierarchy, read, write, work with owners and permissions, according to the restrictions imposed by the original file systems.

The standard tool for creating a new filesystem is '`mkfs`':

```
man mkfs
```

We can choose the type of filesystem and some parameters to create. But it's actually just a wrapper around the real mkfs tools for different types of filesystems:

```
# ls /sbin/mkfs*
/sbin/mkfs /sbin/mkfs.cramfs /sbin/mkfs.ext4 /sbin/mkfs.xfs
/sbin/mkfs.btrfs /sbin/mkfs.ext3 /sbin/mkfs.vfat
```

and they all support their own set of options:

```
# man mkfs.ext4
```

The most commonly used file systems in Linux right now are:

- EXT4 is the Linux journaling file system, or the Fourth Extended File System, which is the successor to the extended file system line originally created in 1992 by Rémy Card to overcome certain limitations of the MINIX file system. The ext4 filesystem can support files up to 16TB and volumes with sizes up to 1 exbibyte (EiB), but this may be limited for certain system versions. For example for RHEL 7/8 — 50TB.

- XFS is a high performance 64-bit journaling file system created by Silicon Graphics, Inc (SGI) in 1993 for their UNIX called IRIX. Although

> XFS scales to exabytes, the host operating system limits can reduce this limit. For example — 500 TB for the maximum file size and file system size for RHEL7 and 1PB/8EB for RHEL8.

Typically ext4 provides better performance for small filesystems on machines with limited I/O capabilities, while XFS provides better performance for large filesystems on machines with high-performance parallel I/O. Also in XFS it is more difficult to reduce the size of the filesystem.

With the '`mkfs`' options, you can set various parameters for creating the filesystem, for example, optimize it for storing large files or for more smaller files.

Once the filesystem is created, we can "`mount`" it. In most cases, this happens automatically when you insert a flash drive or SD card into your computer. But in some cases it needs to be done manually, and you can do it by running the '`mount`' command.

```
man mount
```

You just need to specify the device and directory — mount point, and after mounting you will see the contents of the file system from this device or pseudo device in this directory. And also you can choose the "`mount`" options. For example, we can mount the ISO image with the '`loop`' option:

```
ls /mnt
mount -o loop ...iso /mnt
ls /mnt
```

And then we can '`unmount`' it:

```
umount /mnt
ls /mnt
```

by setting the device or the mount directory as an argument:

```
man umount
```

But Linux/UNIX will not allow you to unmount a device that is busy. There are many reasons for this (such as program accessing partition or open file), but the most important one is to prevent the data loss. You can use the 'fuser' and 'lsof' commands to find the processes that are loading your filesystems:

```
man fuser
...
-k, --kill
...
```

Finally, we can check our filesystem. For journaled file systems, recovering from a power outage is not as relevant, but in some cases it may be useful. In a difficult situation, such as a damaged hard disk, during system boot, you may receive an error message recommending that you run the 'fsck' command to check the file system:

```
man fsck - check and repair a Linux file system
```

As we can see, we have many options for the 'fsck' command, but the main one is 'y' — 'yes'. This means — always try to fix any file system corruption you find automatically, otherwise you could get a zillion troubleshooting questions during the fixup.

After completing the repair, you can find some lost data in a special directory "/lost+found" in the root of the damaged filesystem, which consists of many directories and files whose names contain only numbers — so called 'i-nodes'.

And then you can rename them manually — for example, you found some directory with the files:

- 'passwd', 'group' and 'shadow' — this means that this is '/etc'
- or 'sh', 'ls' and 'cp' — this means '/bin'

and so on...

Swapping

And finally, a few words about swapping. Swapping or paging is a memory management scheme by which a computer stores and retrieves data from secondary storage for use in main memory. It is an MMU-driven virtual memory mechanism that is used in modern operating systems to use secondary storage in order for programs to exceed the amount of available physical memory.

Under the Hood — Virtual Memory

This means you can run applications with a total memory usage that exceeds the physical RAM on your system. The scheduler sends inactive processes to disk swap and loads active tasks from disk into memory. This can reduce overall system performance, but it will increase the ability to run applications.

The main program for creating of swapping area is ‘mkswap‘:

```
man mkswap
```

You just need to specify the device as an argument. Or a file if you need temporary swap space like Microsoft does on Windows.

Then you can enable swap space with the command ‘`swapon`’:

```
man swapon
```

After that, you will see additional swap space using the ‘`free`’ command or in the pseudo file ‘`/proc/meminfo`’. You can also turn off the swap area with the ‘`swapoff`’ command.

OK. But all these mounts and swaps will be connected to our system only until the reboot. To automatically mount them at boot time, we must write them to the filesystems table in the file ‘`/etc/fstab`’:

```
cat /etc/fstab
```

In this text file, we can place static information about connecting file systems and enabling swapping areas:

```
man fstab
```

Each filesystem is described on a separate line; fields on each line are separated by tabs or spaces.

- The first field (fs_spec) describes the block special device or remote filesystem to be mounted.
- The second field (fs_file) describes the mount point for the filesystem. For swap partitions, this field should be specified as ‘none’.

- The third field (fs_vfstype) describes the type of the filesystem. An entry 'swap' denotes a file or partition to be used for swapping.
- The fourth field (fs_mntops) describes the mount options associated with the filesystem.
- The fifth field (fs_freq) is used for these filesystems by the dump command to determine which filesystems need to be backuped.
- The sixth field (fs_passno) is used by the fsck(8) program to determine the order in which filesystem checks are done at reboot time.

After putting some entry in the fstab file, you can run the '`mount`' command with only one of them: device or mount point.

Disk space

Another important task with data in your file system is archiving and backing up. It's wise to look into your filesystems first to analyze disk usage. In case your file system is full on many systems, some graphical disk analysis program will run and you can detect problems visually. But we can also do this job using command line tools that can help you automate some of the admin tasks.

The main tool for reporting file system disk space usage is the '`df`' utility:

```
man df
```

The most useful option is "`-h, -human-readable`" for human readable print sizes in kilobytes, megabytes, gigabytes.

For a more accurate analysis, you can use the '`du`' utility to estimate the file space usage of directories and files:

```
man du
```

So, we can get the size of some directory:

```
# du -hs /tmp/
136K /tmp/
```

And the most commonly used options are "`-k`", which displays sizes in kilobytes, and "`-x`", which means that it will scan only this file system and skip directories on other file systems. Let's take a look at an example of using the command line tools to find the largest directories and files:

```
$ du -kx /tmp | sort -rn | less
```

We examine the directory '`/tmp`', perform a numeric sorting of the sizes of directories and files, and redirect the result to the viewer '`less`' for analysis.

And after finding the largest files and directories, we can clean up our file system and before this archive and back up some data. The easiest way is to simply copy using the '`cp -a`' command to some external drive, or using '`scp -rC`' or '`rsync -avz`' to a remote host.

Also, using the '`cp`' or '`scp`' commands, you can copy any partition or the entire disk, because for us they are just files. But a more efficient way to do this is with the '`dd`' command:

```
man dd
```

By default, it just copies stdin to stdout, perhaps with some re-coding. But the most interesting options for us are: '`if`', '`of`', '`bs`', '`count`', '`seek`' and '`skip`'. With a combination of these options, we can select the input and output files, choose the block size to increase speed of I&O, the number of blocks we want to copy, and seek/skip on output/input. Thus, we can cut and paste any fragment from one device or file to another.

We can also use the '`od`' command to low-level view of a file or device in different formats:

```
man od
```

For example — to our hard drive:

```
# od -bc /dev/sda1 | less
```

Archiving and backups

Historically the archiver, also known simply as '`ar`', is a first Unix utility developed at 1971 in AT&T that maintains groups of files as a single archive file. Today, ar is generally used only to create and update static library files that the link editor or linker uses. An implementation of '`ar`' is included as one of the GNU Binutils.

But the most widely used archiving tools are '`tar`' and '`cpio`'.

The '`tar`' is a "tape archiver" originally developed in AT&T at end of 70s for storing data on magnetic tape. It saves many files together into a single tape or disk archive, and can restore individual files from the archive:

```
man tar
```

Basic operations with a tar archive:

- '`c`' — create archive
- '`t`' — list files in archive and
- '`x`' — extract files from archive

Useful options:

- '`v`' — verbose
- '`f`' — file or device file with archive.
- '`-`' — "Dash" means standard input or output.

  The GNU version also supports compressing/decompressing archives:

- '`-a, -auto-compress`' — use archive suffix to determine the compression program
- '`-z, -gzip, -gunzip, -ungzip`'
- '`-Z, -compress, -uncompress`'
- '`-j, -bzip2`'
- '`-J, -xz`'

```
tar cvzf arch.tar.gz some_files...
tar tvzf arch.tar.gz
tar xvzf arch.tar.gz some_file
```

For standard UNIX ‘tar’, external compression/decompression programs should be used:

```
tar cvf - some_files... | gzip -c > arch.tar.gz
gunzip -c arch.tar.gz | tar tvf -
gunzip -c arch.tar.gz | tar xvf - some_file
```

The main problem with compressed archives is if you have corruption in the middle of the archive file, you will lose all content from the tail. Just because this format is focused on storing on tape and all the metadata about files stored sequentially, not in some central directory. And if you want to improve the reliability of your data, it makes sense to compress the files separately and put them in an uncompressed archive.

Another widely used archiving tool is ‘cpio’:

```
man cpio
```

Basic operations with it:

- `-o|-create`
- `-t|-list`: Print a table of contents of the input.
- `-i|-extract`
- also `-p|-pass-through` is so-called copy-pass mode, cpio copies files from one directory tree to another, combining the copy-out and copy-in steps without actually using an archive.

Unlike ‘tar’, which works with files, ‘cpio’ works with stdin/stdout.

This is good, but such an archive may contain some special files that are not properly processed. For example, you can get hard links split across multiple files. And for real backup in UNIX-like systems, special programs have been developed that know about the internal structure of the file system, for example:

- dump/restore — ext2/3/4 filesystem backup/restore.
- xfsdump/xfsrestore — XFS filesystems backup/retore. The main arguments are: the list of files and directories for dump and ‘`-f dump_file`’.

> But we can also choose the "`dump level`", which is just an integer. A level 0, full backup, specified by -0 guarantees the entire file system is copied. A level number above 0, is so-called "incremental backup", tells '`dump`' to copy all files new or modified since the last dump of a lower level. The default level is 0.

And this makes it possible to implement various "backup strategies". For example, you can create a full backup at the end of the week and then make incremental backups every day of the week. Then at the end of the week for new full backup, you can use the oldest backup storage from the full backup pool. This way, you will have weekly full backups for a specific period and daily saved states in incremental backups for a week or two.

And then you can extract the full dump or individual files or directories from the saved dump using the restore utility:

```
man restore
```

You can also do this interactively (`-i`).

Software installation

Well. Now let's talk about installing software. An initial set of software is installed during system installation, and installed software packages are updated by system services when newer versions are published to the distribution sites. But let's take a deeper look.

Diomidis Spinellis, "Evolution of the Unix System Architecture: An Exploratory Case Study"

From the First Research Edition (November 3, 1971) in which the PDP-7 Unix was rewritten on the PDP-11 processor, UNIX documented the "User Maintained Programs" guidelines by organizing third-party code as so-called "packages" or "ports".

The two first Berkeley distributions introduced to the user community third-party software packages targeting Unix. Over the years packages proliferated and got distributed, initially through USENET newsgroups and later over the internet in the form of ports to a specific operating system distribution. The established filesystem directory hierarchy, provided a template for laying out the source code, the documentation, and the manual pages. In addition, the use of '`make`' utility provided a common way for expressing compilation and deployment rules.

And now if we are talking about free and open source, the most general way is compiling from source. Many projects simply require downloading the source, running the configure script, and running '`make install`'. This command reads an instruction from a file named "`makefile`" and installs the software into the target directory, by default '`/usr/local`'. You can change this and other settings during configuration.

Sounds good, but in most cases we may have problems uninstalling and updating installed software, because in most cases such actions may not be so simple, and the purpose of "uninstall" is not implemented in the Makefiles. And to perform a complete set of actions with the software, a special type of files called software packages and software were developed to manage them. They differ on different systems and distributions.

Repositories and packages

BSD UNIX packaging has been extended to the FreeBSD 'ports' machinery, which provides a mechanism for compiling and installing third-party packages and their dependencies. The main package Linux utilities are '`rpm`' (RPM Package Manager) for RH-like systems and '`dpkg`' for Debian-based distributions:

```
man rpm
man dpkg
```

A package is a file that you can install, remove, and update. But when we have many packages with a complex system of dependencies between them, it can be too difficult to handle the full set of dependent packages during package manipulation.

To solve this problem, so-called repositories have been developed, which collect many packages, resolve dependencies between them and put information about this in metadata files.Such repositories are hosted on the servers of the respective projects, and we can access them via the Internet.

The main tools for working with repositories are:

- '`yum`' — Yellowdog Updater Modified for RPM repositories, now replaced with '`dnf`' package manager.
- '`apt`' — package manager for Debian-based repositories. As we can see, using these tools, we can perform the same actions as with '`rpm`' and '`dpkg`': install, remove and update packages with dependencies. We can also get information about packages and package groups in the repositories, get a list of packages, search for packages by name or by files included in the package.

Also may be useful '`yumdownloader`' — program for downloading RPMs from Yum repositories.

You can find descriptions of the repositories in:

- `/etc/yum.repos.d/` —
- `/etc/apt/sources.list`

You can see the Table of equivalent commands for package management on both Ubuntu/Debian and Red Hat/Fedora systems

Under the hood — Devices and drivers

CPAN, PyPi, NPM, static binaries, docker containers

Booting and services starting/stopping

All right. But how does our system get started? In fact, when you turn on the computer, a special piece of code is launched, built into the hardware – firmware. There are many such firmware: legacy PC BIOS, UEFI, Uboot, OpenBoot, Coreboot, etc. The firmware reads the main bootloader from disk: BIOS from MBR, UEFI from EFI system partition, and so on.

Then the secondary bootloader started. This loader can be more or less complex and customizable. The most commonly used Linux bootloaders currently are lightweight boot system `SYSLINUX` for FAT file system and `ISOLINUX` for ISO images, which is mainly used to boot installation or live images, and the general purpose Grub bootloader.

Usually Grub is installed during system installation, including configuring the boot of other operating systems installed on your computer. But in some cases MS Windows can reinstall the bootloader without asking you, in which case you may lose your boot settings. To restore it, you need to boot from the installation media in repair mode and run the '`grub2-install`' program for your system storage. For example:

```
grub2-install /dev/sda
```

After the kernel is loaded, a special process called '`init`' is started. In original UNIX, as well as BSD '`init`', just run the script '`/etc/rc`', which completely determines the further behavior of the system.

A different '`init`' machinery is implemented for SYSV systems. On such systems, you can invoke the '`init`' program at any time by setting the run-level as a parameter. Runlevels defined in the '`init`' configuration is located in the '`/etc/inittab`' file. Each line in the `inittab` file consists of four colon-delimited fields and describes:

- what processes to start, monitor, and restart if they terminate
- what actions to take when the system enters a new runlevel
- the default runlevel

Inittab's fields:

```
id:runlevels:action:process
```

id (identification code) — consists of a sequence of one to four characters that identifies its function.

- runlevels — lists the run levels to which this entry applies.
- action — specific codes in this field tell init how to treat the process. Possible values include: initdefault, sysinit, boot, bootwait, wait, and respawn.
- process — defines the command or script to execute.

The line '`initdefault`' defines the default runlevel. Different systems define different init level hierarchies, but some of them have the same meaning:

- Runlevel 0 is halt.
- Runlevel 6 is reboot.
- Runlevel 1 is single-user.
- 2–5 are most often some multiuser runlevels.

Most often, the executable scripts in '`inittab`' are just some '`rc`' scripts that go through the appropriate `/etc/rc<runlevel>.d/` directories and run the stop scripts first, starting with a big `K` (kill) with a '`stop`' parameter, and then starting the scripts that start with large `S` with a '`start`' parameter:

```
$ ls -l /etc/rc.d/rc5.d/
```

And, as we can see, this script is simply symbolic links to scripts from '`/etc/init.d/`', moreover, the Kill scripts and the Start scripts can be linked to the same file. If we look at them, we will see — there are just scripts that do something according to the start or stop parameter:

```
$ less /etc/init.d/network
```

And if you want to implement our own service script, you just have to support the '`start`' and '`stop`' parameters. To configure your own policy for stopping and starting services at any level, you simply have to link the scripts which you need from '`/etc/init.d/`' to the appropriate runlevel directories. The order in which scripts are run is determined by the numbers at the beginning of the filenames.

Some commands that can help you with this work:

- '`service`' – run a System V init script
- '`setup`' and '`chkconfig`' — updates and queries runlevel information for system services

The most commonly used Linux distributions now use '`systemd`' instead of such systems. The main advantage of this system is faster parallel launch of services at system startup, as well as unification of services and work with devices. These utilities and scripts are still present for compatibility, but now the main tool is '`systemctl`':

```
man systemctl
```

We can list system services, start, stop and get their status, enable and disable them to automatically start at boot time.

To find out the log messages about boot startup and system operation, we can look at the system log files:

- `/var/log/messages` — RH-like Linuxes
- `/var/log/syslog` — Debian, Ubuntu

In modern Linux based on '`systemd`' we have '`journalctl`' to work with the '`systemd`' journal system.

Network configuration

Finally, let's discuss the administrative tasks associated with the network. In most cases, after installing the system, you have a more or less configured network in accordance with the DHCP settings of your local network. The most you need is to set a password for your WiFi.

But in some cases it would be helpful to have some utilities to view and manage network settings. Unlike other devices, network interfaces do not have their own representations in the device files in the `/dev` directory. You can work with them only with the help of special utilities. Traditional set of utilities for network configuration:

- `ifconfig` — configure a network interface
- `route` – show/manipulate the IP routing table

With no arguments, `ifconfig` just shows us the current state of enabled network interfaces. By pointing to a network interface, we can "up" and "down" them, we can also manually set the IP address and netmask:

```
man ifconfig
```

You can use the '`route`' utility to work with the routing table. To view the route table we may run command:

```
route -n
```

The option '`-n`' show numerical addresses instead of trying to determine symbolic host names. This can be useful if you are having problems accessing the DNS server.

```
man route
```

With this command we can add and remove routes to hosts and networks. We can set gateways to them.

In modern Linux distributions, these '`net-tools`' are outdated and are not installed by default. They are migrating to the more versatile '`ip`' utility, which also supports more advanced networking options than '`net-tools`':

```
man ip
```

You can use '`ip link`' to perform the same tasks as '`ifconfig`' and '`ip route`' to replace '`route`':

```
$ ip link help 2>&1 | less
$ ip route help 2>&1 | less
```

The next important task when setting up your network is setting up DNS resolving. This configuration is placed in '`/etc/resolv.conf`':

```
$ cat /etc/resolv.conf
```

Here we can configure up to 3 nameservers.

```
man resolv.conf
```

Also, using the '`search`' configuration option, we can configure the domains in which short names will be searched.

Network access

And finally, a few words about regulating network access to your computer. As we understand, network attacks are currently the most dangerous. And the most famous tool for restricting access is a firewall.

At its deep level, the firewall in Linux is controlled by the '`iptables`' utility and moving to '`nftables`'. But at a higher level, different systems manage it in different distributions:

- Canonical's Uncomplicated Firewall ('`ufw`') to configure the iptables on every Ubuntu and Debian system I've used in recent years. The firewall isn't enabled by default in Ubuntu nor installed by default in Debian. As its name suggests, it's fairly uncomplicated to set up and maintain. It has an easy to remember syntax that's more friendly to human users than the underlying iptables rules.
- Fedora and Red Hat Enterprise Linux enables FirewallD by default. Its '`firewall-cmd`' front-end has almost the same feature set for basic firewall management as '`ufw`', and adds network zone management to the mix. Zone management allows you to set up presets with rules for different network conditions/locations. For example "Home" and "Office" where all communications with local machines are allowed, and "Public Wi-Fi" where no communication with the same subnet would be allowed. Rules can be applied automatically per network interface, or used through NetworkManager and the GNOME network GUI '`system-config-firewall`'.

Both front-ends come with pre-defined rules for common server services and protocols. These rules include a keyword/name and associated industry standards and default ports for running services publicly. They each come in their own formats that aren't interoperable with each other, of course. `ufw` uses service-named files containing one line with port and protocol, and FirewallD uses six lines of XML to create the same profile.

The best policy is to simply close all services from the Internet that you do not need on your computer, and only open those that you need for remote access. For example — SSH.

And then just use the lighter tweak tool — host access control files:

```
/etc/hosts.deny:
         ALL: ALL
/etc/hosts.allow:
         ALL: LOCAL @some_netgroup
         ALL: .foo.bar EXCEPT hacker.foo.bar
```

This is the configuration for the so-called tcp wrapper, which was originally developed as a '`tcpd`' service for the '`inetd`' superserver, and now its functionality is included in the '`libwrap`' library that is used by several network services such as NFS.

Again, the easiest way is to disable everything in the configuration file '`/etc/hosts.deny`' and allow only a fixed list of hosts to access your computer, for example, via SSH in '`/etc/hosts.allow`'. Using this mechanism is easier than using a firewall because all changes are made immediately after saving the configuration file — you don't need to reconfigure and reload any services.

Network services

And if we talk about older classical systems, then tcp_wrapper '`tcpd`' is configured through the '`inetd`' service. As we'll see, this is a so-called super-server designed to make the life of network developers easier. To create the server side of a network application, you just need to develop an application that reads stdin and writes to stdout, and write the service configuration to the '`inetd`' configuration file. All the work of listening on network sockets and maintaining the connection makes a super-server for you.

Classic services such as FTP, POP3, and telnet were designed to work with '`inetd`'. It is also possible to configure an HTTP server for access via '`inetd`'. Newer systems have replaced '`inetd`' with '`xinetd`', which provides better protection against DOS attacks, and replaced with '`systemd`' in newest systems.