# LINUX USER GUIDE (III)

## Working on the UNIX command line, part II:

## Basic commands and file systems (cont.)

Cristian Vidrașcu
cristian.vidrascu@info.uaic.ro

February, 2024

Introduction . . . 3

**UNIX and users** **4**
- User accounts and user groups . . . 5
- Commands about the users . . . 7

**Files in UNIX systems (cont.)** **8**
- The physical structure of file systems . . . 9
- Mounting the file systems . . . 10
- File protection through access permissions . . . 11
- Other commands for working with files (cont.) . . . 14

**Other categories of simple commands** **15**
- Processes & the process system . . . 16
- Commands that provide various information . . . 17
- Other categories of commands . . . 18

**Troubleshooting** **19**
- "Commandline Survival Guide" . . . 20

**Bibliographical references** **21**

## Overview

Introduction

**UNIX and users**
- User accounts and user groups
- Commands about the users

**Files in UNIX systems (cont.)**
- The physical structure of file systems
- Mounting the file systems
- File protection through access permissions
- Other commands for working with files (cont.)

**Other categories of simple commands**
- Processes & the process system
- Commands that provide various information
- Other categories of commands

**Troubleshooting**
- "Commandline Survival Guide"

**Bibliographical references**



## Introduction

In the first part of this presentation we discussed about:

- The main categories of commands
  - Help commands
  - Text editors
  - Compilers, debuggers, etc.
  - Commands for remote connection to a UNIX server
- Files in UNIX systems
  - The logical structure of file systems
  - Basic commands for working with files (and directories)
  - Other commands for working with files

* * *

Now we will continue the presentation with the topics summarized on the previous slide.

## Outline

Introduction

UNIX and users
- User accounts and user groups
- Commands about the users

**Files in UNIX systems (cont.)**
- The physical structure of file systems
- Mounting the file systems
- File protection through access permissions
- Other commands for working with files (cont.)

**Other categories of simple commands**
- Processes & the process system
- Commands that provide various information
- Other categories of commands

**Troubleshooting**
- "Commandline Survival Guide"

**Bibliographical references**



## User accounts and user groups

- Each **user**, to be able to work on a UNIX system, must have an account on the respective UNIX system. The account is identified by a *username* and an associated *password* , which must be provided when connecting to the system (*i.e.*, the system authentication operation, called "*login*" for short).
  Also, each user account is assigned a UID (*User IDentification*).
  There is a special user, called *root* (or *superuser*), with UID = 0. He has full rights over the entire system, being responsible for system administration.

  The "database" with information about users who have accounts in the system is stored in the file /etc/passwd ([3]), containing lines of text conforming to the following template (consisting of 7 fields with information, separated by the character ':') :

  *username* : **x** : *UID* : *GID* : *userdata* : *home directory* : *login shell*

  where *UID* is the user ID, *GID* is the ID of the primary group it belongs to, and *userdata* are the user's personal data (only if they were entered by the superuser when the account was created).

  *Note*: that **x** in the second field indicates that the password associated with the account is stored, in encrypted form, in the file /etc/shadow (which is accessible only to the superuser).

## User accounts and user groups (cont.)

- There are **user groups**, which make it easier to manage user access rights and restrictions to system resources.
  Any user group is assigned a name (called the *groupname*) and a number called GID (*i.e.*, *Group IDentification*), as well as an associated password, usable for "temporary membership".

  Each user is part of a primary group (indicated by the *GID* field in `/etc/passwd`) and can optionally be affiliated with other additional groups (based on associations in `/etc/group`).

  The "database" with information about user groups in the system is stored in the file `/etc/group` ([4]), containing lines of text conforming to the following template (consisting of 4 fields with information, separated by the character ':') :

  *groupname* : **x** : *GID* : *list of users*

  where *GID* is the group ID, and the last field contains a list, possibly empty, of usernames separated by the character ',' , which "declares" the affiliation of the respective users to that group.

  *Note*: that **x** in the second field indicates that the password associated with the group is stored, in encrypted form, in the file `/etc/gshadow` (which is accessible only to the superuser).

6 / 21

## Commands about the users

- Commands for managing user accounts:
  - changing the data associated with a regular user account (by him): `passwd`, `chsh`, etc.
  - changing the data associated with a user account (only by the superuser): `chage`, etc.
- Commands that provide various information about users:
  - finding out information about users of the system (*i.e.*, about user accounts): `whoami`, `id`, `groups`
  - finding out information about users connected to the system (and about their work sessions on the system): `users`, `w`, `who`, `finger`, `last`

*Demo*: see the examples [Info user(s) #1] and [Info user(s) #2] in the lab support, available here.

7 / 21

## Outline

Introduction

**UNIX and users**
- User accounts and user groups
- Commands about the users

**Files in UNIX systems (cont.)**
- The physical structure of file systems
- Mounting the file systems
- File protection through access permissions
- Other commands for working with files (cont.)

**Other categories of simple commands**
- Processes & the process system
- Commands that provide various information
- Other categories of commands

**Troubleshooting**
- "Commandline Survival Guide"

**Bibliographical references**



## The physical structure of file systems

In the UNIX family of operating systems, there are several "types" of file systems, all of which have a tree-like logical structure, but differ from each other in the physical structure of each one, *i.e.* in the data structures used to represent data (*i.e.*, of information about files stored in a file system of that "type") and by the format of their storage on disk.

Examples of file system "types" in Linux: ext2fs / ext3fs / ext4fs, xfs, reiserfs, btrfs, zfs, vfat, ntfs, etc. For info on these file system "types" see the documentation `man 5 fs` .
*Remark*: these are "types" of file systems used to store files on disk. However, there are also "types" with various specialized roles, which store information not on disk, but in volatile RAM memory or in another form of storage (*e.g.*, tmpfs, procfs, devfs, sysfs, unionfs, squashfs, etc.), or distributed file systems (*e.g.*, NFS, AFS, SMB/CIFS, etc.).

**The "main" file system**: in a UNIX system we have only one "logical disk", called *the root filesystem*, and referred to by its root, named "/" (unlike Windows, where we have several units of "logical disks", referred to by C:, D:, and so on...).

This filesystem is configured when the operating system is installed on that computer, and all other file volumes, which need to be accessed, are "included" as subtrees in it, through the *mounting* operation.

## Mounting the file systems

A UNIX system can contain multiple file volumes, each volume being a file system of a particular "type" and stored on disk in a "storage unit" for file volumes.

Typically, a computer can have multiple disks (*i.e.*, physical storage devices), and each disk can be "split" into one or more partitions. And each disk partition can store only one volume of files. However, there are also other "storage unit" definition schemes (*e.g.*, RAID systems).

Each file volume logically has a root directory (referred to by "/", relative to the "storage unit" that contains it), and can be navigated by loading the operating system from that volume.
If, however, any file system on a storage device must be used without loading the operating system from that device, there is the solution of **mounting** the tree structure of that file system into the tree structure of the "main" file system, from which the operating system was loaded.

*Mounting* a file system is done with the `mount` command (only by superuser), and then accessing its contents is done via the mount point.
And the reverse operation, *unmounting* a file system, is done with the `umount` command.

Configuration files: `/etc/fstab` (for automatic mounting), `/etc/mtab`, `/etc/filesystems`.
For details on these files see the documentation `man 8 mount` and `man 5 fstab` .

10 / 21

## File protection through access permissions

**The classic security model from** UNIX:
Each file has a specific user associated with it as the **owning user**, and a specific user group as the file's **owning group**.

Users can thus be classified into three categories in relation to a file:

- the owner of the file – **u** (*user*)
- group mates of the owner – **g** (*group*)
- all other users – **o** (*others*)

Each file has three **access permissions** associated with it, for each of the three categories of users of that file:

- **r** (*read*) – permission to read the file
- **w** (*write*) – permission to write the file
- **x** (*execute*) – permission to execute the file

11 / 21

## File protection through access permissions (cont.)

For directories, access permissions have a special meaning:

- **r** (*read*) – permission to read the contents of the directory (*i.e.*, permission to find out the names of the files in the directory)
- **w** (*write*) – permission to write the contents of the directory (*i.e.*, permission to add new files to the directory and to delete or rename files in the directory)
- **x** (*execute*) – permission to inspect the contents of the directory (*i.e.*, permission to access the files in the directory)

Changing the access permissions of a file can be done with the `chmod` command, only by its owner (and the superuser).

Changing the owning user, or the owning group, of a file can be done using the commands `chown`, respectively `chgrp` ([5]). Likewise, only by the current owner of the file (and the superuser).

*Demo*: see the example [Basic file operations #5] in the lab support, available here.

12 / 21

## File protection through access permissions (cont.)

*Checking access permissions* is performed by the operating system at the time of starting a work session with a file (*i.e.*, at the time of calling `open()`) or at the time of starting the execution of a file (*i.e.*, at the time of calling `exec()`), like this:

We assume that the file to be accessed is located at the absolute path: `/d1/d2/d3/.../file1` , and the owner of the command that wants to access the file is user Y.
The OS "scans", one by one, all directories in that path (*i.e.*, `/`, `d1`, `d2`, `d3`, `...`) and checks the presence of the permission **x** for the category to which Y belongs in relation to the owner of each directory. And for the file itself, `file1`, the presence of the corresponding permission(s) of the access type declared in the `open()` call is checked, respectively the presence of the permission **x** in the case of the `exec()` call is checked, for the category to which Y belongs in relation to the owner of that file.

If at least one of these individual checks fails (*i.e.*, if that access permission is denied), then that call (`open()` or `exec()`) fails, returning the value `-1` and, as the cause of the error, assigns in the `errno` variable a numerical value with the meaning "Permission denied".

13 / 21

### Other commands for working with files (cont.)

- Commands for various processing of file contents ([6]):
  - `sort` – for sorting a text file (or several), according to various criteria
  - `wc` – provides some statistics about the contents of text files (*e.g.*, the number of characters, words or lines of text)
  - `tr` – filter to "translate" certain specified characters
  - `uniq` – filter that removes consecutive duplicate lines
  - `rev` – filter to "reverse" each line of text
- Other useful file commands:
  - `find` – for searching files (of any type) according to various criteria ([6])
  - `cmp`, `comm`, `diff` – for comparing two files, according to various criteria
  - `sum`, `md5sum`, `sha1sum`, `sha256sum` – calculates various checksums
    (*Note*: these are useful for checking the integrity of files stored over longer periods.)

*Demo*: see the examples [sort #1], [sort #2], [wc #1], [find #1] and [find #2] in the lab support, available here.



## Other categories of simple commands 15 / 21

### Outline

Introduction

**`UNIX` and users**
- User accounts and user groups
- Commands about the users

**Files in `UNIX` systems (cont.)**
- The physical structure of file systems
- Mounting the file systems
- File protection through access permissions
- Other commands for working with files (cont.)

**Other categories of simple commands**
- Processes & the process system
- Commands that provide various information
- Other categories of commands

**Troubleshooting**
- "Commandline Survival Guide"

**Bibliographical references**

## Processes & the process system

**Definition**: a *process* is an execution instance of a program.
All the processes of a system are logically organized in a tree hierarchy, based on a mechanism of "*genetic inheritance*": each process in the system has a process that created it, called the *parent process*, and from which it "inherits" a certain set of characteristics (such as owner, access permissions, etc.), and may in turn create one or more *child* processes.
Each process is assigned a PID (acronym from *Process IDentification*), which is a positive integer and is unique during the lifetime of that process (*i.e.*, at any given time, no two processes with the same PID exist in the system).
There is a special process, the one with PID = `0`, which is created when the `UNIX` system is initialized (booted) on that computer. It has no parent process, being the root of the tree of processes that will be created over time (until the computer is turned off).

- commands that provide information about active processes in the system: `ps`, `pstree`, `top`, `jobs`, `fg`/`bg`
- commands for scheduling / managing process execution: `kill`/`killall`, `at`, `nice`, `time`/`times`, `timeout`

*Demo*: see the example [Info running programs] in the lab support, available here.

16 / 21

## Commands that provide various information

- current time (date + hour) and calendar information: `date`, `cal`/`ncal`
- information about the time since the system has been started: `uptime`
- information about terminals: `tty` , `stty`
- information about the system (computer) name – short name, FQDN name (*i.e.*, full name in the DNS system), IP address, etc.: `hostname`, `hostnamectl`, plus the configuration files `/etc/hostname` ([7]), `/etc/machine-id`, `/etc/machine-info` and `/etc/hosts`
- information about the kernel (`Linux` or other `UNIX`) of the installed OS: `uname`
- miscellaneous information about computer hardware components: `lshw`, `lscpu`, `lsmem`, `lspci`, etc.[a]
- information about the `Linux` distribution installed on the computer: the configuration file `/etc/os-release` ([7]) **or** `lsb_release`[b]

*Demo*: see the example [System info] in the lab support, available here.

[a] These commands are specific only for the `Linux` operating system.
[b] This command may not be present in some `Linux` distributions.

17 / 21

## Other categories of commands

- writing messages on the screen (*i.e.*, `stdout`) : `echo` (unformatted messages), `printf` (formatted messages)
- non-interactive editors : `ed` (*line-oriented text editor*), `sed` (*stream editor*)
- *scripting* languages: `awk`, `perl`, `python`, etc.
- archiving, compression, file encoding : `gzip` / `gunzip`, `zip` / `unzip`, `tar`, `fsarchiver`, `compress` / `uncompress`, `uuencode` / `uudecode`, etc.
- writing messages on another user's screen : `write`, `talk`
- connecting to / disconnecting from a `UNIX` system : `ssh`, `telnet`, `login` / `logout`, `exit`[a]
- programs for various INTERNET protocols : `ftp`, `sftp`, `scp`, `mail` / `pine`, `lynx` / `links`, `host`, etc. (*Deprecated protocols* : `finger`, `talk`)
- diagnostic, debugging and instructional tools : `strace` and related tools (`ltrace`, `perf-trace`, `trace-cmd`, etc.) [b]

[a] Called only from the login instance of the shell.
[b] Currently, these tools are only available for the `Linux` operating system.



# Troubleshooting



## Outline

Introduction

**`UNIX` and users**
- User accounts and user groups
- Commands about the users

**Files in `UNIX` systems (cont.)**
- The physical structure of file systems
- Mounting the file systems
- File protection through access permissions
- Other commands for working with files (cont.)

**Other categories of simple commands**
- Processes & the process system
- Commands that provide various information
- Other categories of commands

**Troubleshooting**
- "Commandline Survival Guide"

**Bibliographical references**

### "Commandline Survival Guide"

What to do if a command is not responsive ? Follow the steps below:

- First, wait a reasonable amount of time, maybe the program is not "stuck" but just busy with intensive calculations. If the prompt still does not appear, then:
- Press (simultaneously) the keys CTRL + C . This causes the interrupt signal SIGINT to be sent to that program. If the prompt still does not appear, then:
- Press (simultaneously) the keys CTRL + \ . This causes the termination signal SIGQUIT to be sent to that program. If the prompt still does not appear, then:
- Press (simultaneously) the keys CTRL + Z . This causes that program to be suspended (*i.e.*, placed in the SUSPENDED state) and the prompt is displayed.
  Further, to stop that program (it is only suspended, not finished), proceed as follows:
  Using the command ps find the PID of that non-responsive program, and then type the command
  UNIX> kill -9 *pid*
  where *pid* is the previously found PID.
  As a result of this command, that process is killed (*i.e.*, forcibly terminated).



## Bibliographical references



### Mandatory bibliography

[1] Chapter 2, §2.1 and §2.2 from the book "*Sisteme de operare – manual pentru ID*", by C. Vidrașcu, UAIC Publishing House, 2006. This is available as ebook at the address:

- https://profs.info.uaic.ro/~vidrascu/SO/books/ManualID-SO.pdf

[2] The online support lesson associated with this presentation:

- https://profs.info.uaic.ro/~vidrascu/SO/support-lessons/bash/en/support_lab2.html

Additional bibliography:

[3] Documentation about /etc/passwd : man 5 passwd and man 5 shadow

[4] Documentation about /etc/group : man 5 group and man 5 gshadow

[5] man 1 chmod , man 1 chown and man 1 chgrp

[6] man 1 sort , man 1 wc , man 1 tr , man 1 uniq , man 1 rev and man 1 find

[7] The documentation about /etc/hostname, /etc/os-release, etc. : man 5 hostname , man 5 machine-id , man 5 machine-info and man 5 os-release

Plus the documentation of all the other commands shown, accessible with the man command.