LINUX USER GUIDE (II)

Working on the UNIX command line, part I:

Basic commands and file systems

Cristian Vidrașcu
cristian.vidrascu@info.uaic.ro

February, 2024

Overview

Introduction

A *command interpreter* (also called a *shell*) is an executable program that creates the interface between the user and the operating system, *i.e.* it takes the commands entered by the user, executes them and displays the results of their execution.
In `UNIX` systems there are several command interpreters, which offer the possibility of working on the command line, *i.e.* the user interface is text-based, not graphical.

In `UNIX` systems there are two categories of *simple commands*:

- *Built-in commands*: those that are implemented in command interpreters.
  Examples: `cd`, `help`, etc.
- *External commands*: they are implemented independently (*i.e.*, they are each found in a file, having the same name as the respective command), in:
  - executable files (*i.e.*, executable programs obtained by compiling from source programs written in C or other programming languages).
    Examples: `passwd`, `ls`, etc.
  - text files with sequences of commands, called *scripts*.
    Examples: `.profile`, `.bashrc`, etc.

Introduction (cont.)

The general form of launching a simple command into execution:

`UNIX> command_name [options] [arguments]`

Remarks:

- The “UNIX> ” text is the prompt displayed by the shell, at which it waits for you to type the desired command, followed by pressing the ENTER key.
  Note: the text displayed as a prompt is configurable – we’ll discuss how later.
- The separator character between the command name and its parameters, as well as between each of the parameters, is the character SPACE or TAB.
- Options and/or arguments specified after the command name may be missing (*i.e.*, they are optional, indicated by a pair of brackets '[...]'; these do not need to be typed).
- By convention, options are preceded by the '-' character (or '--', for long options).
- The meaning of the arguments depends on the command (*e.g.*, most often they are filenames).
- External commands can also be specified by the full name (*i.e.*, the *absolute or relative path*) of the respective file.
- A longer command (*i.e.*, with many parameters) can be entered on several lines, in which case each line must be terminated with the character '\' followed by ENTER, except for the last line (the termination it is done by pressing only the ENTER key).

The main categories of commands

Outline

Help commands

- Documentation about built-in commands is available using the `help` built-in command. It can be called like this:
  `UNIX> help`
  It will list all built-in commands available in that shell.
  `UNIX> help built-in_command_name`
  It will display the help page for the specified built-in command.
- Documentation about external commands, as well as C functions in the POSIX API and those in C STANDARD LIBRARY, is available using the commands `man`, `whatis`, `which`, `whereis`, `apropos`, `info` ([3]).

  Remark: the man pages are organized into 8 sections. **Section 1** contains documentation about commands accessible to regular users, and **section 8** contains documentation about commands accessible only to the system administrator. **Section 2** contains documentation about `Linux` system calls (*i.e.*, the POSIX API), and **section 3** contains documentation about C standard library functions. You can read about the rest of the sections and other details here and here .

6 / 19

Help commands (cont.)

How to call the commands that provide access to documentation about external commands and C functions from the POSIX API and the C STANDARD LIBRARY :

- `UNIX> whatis name`
  It will list the existing commands and C functions with the specified name, as well as the manual sections that contain them.
- `UNIX> man [section] name`
  It will display the man page (from the specified section) for the specified command or C function.
- `UNIX> which name`
  It will show the location of the command.
- `UNIX> whereis name`
  It will show the location of the command and associated manual pages.
- `UNIX> apropos word`
  It will search for the specified word in the command descriptions, displaying the list of results found.
- `UNIX> info [options] [word ...]`
  It will display the documentation, in the TEXINFO alternative format, for the specified words.

Demo: see the example [Accessing the man pages] in the lab support, available here.

7 / 19

Text editors

As in `Windows`, in `UNIX` systems there are several categories of editors:

- Text editors for “plain” text (*i.e.*, no specialized formatting):
  a) with text-mode UI: `mcedit`, `pico`, `nano`, `vi`, `vim`, `joe`, etc. ([4])
  b) with graphical UI: `gedit`, `kedit`, etc.
- Office suite (with graphical UI): OpenOffice, LibreOffice, etc.
- Specialized editors for HTML: they allow the writing of web pages in the HTML language, which contains instructions for formatting the actual text.
- Specialized editors for TeX / LaTeX: such an editor is a *working environment* that allows technical editing of scientific documents in the TeX / LaTeX language.
- `emacs`: is a text-mode editor, part of the GNU project, with powerful facilities, usable as an IDE (*i.e.*, integrated programming environment).
- Other types of editors (*e.g.*, those with specialized facilities for writing programs in various programming languages, often as part of an IDE).

Compilers, debuggers, etc.

On `UNIX` systems there are compilers and interpreters for most existing programming languages, newer or older : C, C++, Pascal, Fortran, Java, Ada, etc.

For C and C++ languages, the most popular is the compiler from the GCC (*the GNU Compiler Collection*) suite ([5]), which is used like this:

- Compiling a C program is done with the command:
  ```
  UNIX> gcc source.c [-o executable] [-Wall]
  ```
  which performs the compilation (including link-editing of the object code) of the specified source file in the executable file with the name specified by the `-o` option.
  Remarks: i) without the `-o` option, the default name of the output executable is `a.out` ;
  ii) the `-Wall` option will include all warnings, not just compile errors ;
  iii) it is mandatory to use the .c extension in the source file name !
- Similarly, compiling a C++ program is done with the command:
  ```
  UNIX> g++ source.cpp [-o executable] [-Wall]
  ```

To debug the programs, you can use the GNU DeBugger, which is accessible through the `gdb` command.

Commands for remote connection to a UNIX server

- Remote interactive work session on a UNIX (or Linux) server:
  - with unencrypted communications:
    `UNIX> telnet [options] [computer [port]]`
  - with encrypted communications ([6]):
    `UNIX> ssh [options] [[username@]computer]`
    Note: under Windows you can also use the Putty application for connection to a UNIX server.
- Interactive remote file transfer session:
  - with unencrypted communications:
    `UNIX> ftp [options] [computer [port]]`
  - with encrypted communications ([6]):
    `UNIX> sftp [options] computer`
    A non-interactive variant is the command scp:
    `UNIX> scp [opts] [[user@]host1:]file1 [[user@]host2:]file2`
    Note: under Windows you can also use the WinSCP application for remote file transfer.

Remark: within FII there is a Linux server for student use that you can work on until you install Linux on your personal computer. The name of the server is students.info.uaic.ro and it only allows encrypted connections and, for now, only from within the faculty's local network perimeter.

10 / 19

Other categories of commands...

Remark:
We will first introduce the concepts of *files* and *file systems*, and then we will continue with the presentation of other categories of commands available on UNIX systems.

(to be continued)

11 / 19

Files in UNIX systems

Outline

Files and file systems

As in `Windows`, in `UNIX` family of operating systems user data and programs are stored on disk in *files*. Each file is identified by a *name* assigned by its *owner* (*i.e.*, the user who created it). Files are organized (*i.e.*, grouped) into "volumes" on disk, called *file systems* (or *filesystems*).

File names in `UNIX` filesystems are *case-sensitive*, unlike the filesystems in `Windows` (`FAT` and `NTFS`). File names can be up to 255 characters long and can contain any number of '.' (*i.e.*, they are not split in the form 8.3, *name.extension*, as in the `FAT` filesystem in `MS-DOS` or `Windows`), the only restriction being the non-use of the characters `NULL` and '/' as well as some other special characters, *e.g.* `LF` (*Line Feed*), `CR` (*Carriage Return*), etc.

Note: although `UNIX` systems do not enforce any file naming convention, there are still standardly used suffixes such as: `.c` and `.h` for C source files, `.cpp` and `.h` for C++ source files, `.txt` for plain, unformatted text files, `.sh` for files with command sequences (called *scripts*), `.tar`/`.gz`/`.zip` for tar/gz/zip, etc. format archives.
These suffixes are called *extensions* and are useful from the point of view of users, to "hint" at them what kind of content that file has, just like in `Windows`.

Files and file systems (cont.)

In UNIX filesystems, files are classified into the following 6 *filetypes*:

- normal files – are files with ordinary content (data or programs) ;
- directories – are "containers" of files ;
- hard or symbolic links – they are a kind of aliases, *i.e.* other synonymous names for already existing files ;
- special device files, block mode or character mode – are peripheral (physical or logical) drivers ;
- *fifo* files – are local communication mechanisms between processes ;
- *socket* files – are used for remote communication between processes over a network (*i.e.*, between programs running on different computers).

Attention: do not confuse the extension in a file name with its type !
All the extension examples mentioned on the previous slide are used with normal type files.

Note: you are already familiar with the first 3 file types from Windows; *e.g.*, the equivalent of links are the so-called *shortcuts* (on the desktop). In fact, the other file types also have equivalents in Windows, but you probably haven't encountered them in practice yet (you will meet them in 3^{rd} year, in the CSSO course).

The logical structure of file systems

In UNIX systems, each **filesystem** is organized *logically* (*i.e.*, by the way it is "seen" by users and programs accessing the files) into a *tree hierarchy*, based on the idea of *directory* (*i.e.*, a "container" of files):

as in Windows, the file system is organized as a recursive tree of directories, which can contain (sub)directories and actual files (*i.e.*, normal and other file types).

But, unlike Windows, in a UNIX system we have a single "logical tree", called *the root filesystem*, and the root of this tree is referred to by the name "/".

The *full name* of a file, regardless of its type, is given by the linear representation of the path from the root "/" to that file, in this "logical tree".

Examples: `/usr/bin/bash` , `/etc/passwd` , `/dev/sda` , `/home/`*`username`*`/.profile` , etc.

Note: this linear representation, also called the *absolute path* of that file, uses the '/' character as a separator between (sub)directory names in that path (unlike Windows, which uses the '\' character as a separator).

Actually, just like in Windows, also in UNIX systems the files can be accessed (specified) in two ways:
– relative to the root "/" of the file system (called *specification by absolute path*)
– or relative to the *current working directory* (called *specification by path relative to current working dir.*)

Basic commands for working with files (and directories)

- Basic commands for manipulating directories:
  - `mkdir` – to create a directory (or more)
  - `rmdir` – to delete a directory (or more)
  - `ls` – to display the contents of a directory (or more)
  - `pwd` – to find out the current working directory
  - `cd` – to change the current working directory

Demo: see the examples [Basic file operations #1], i) and [Basic file operations #2], iii) in the lab support, available here.

16 / 19

Basic commands for working with files (and directories) (cont.)

- Basic commands for manipulating files of any type:
  - `ln` – for creating a file (or more) of type *hard or symbolic link*
  - `mkfifo` – for creating a file (or more) of type *fifo*
  - `mknod` – for creating a file (or more) of type *device*
  - to create a file (or more) of the normal type, we can use either any plain text editor or the commands `touch`, `truncate`, etc.
  - `rm` – for deleting files of any type
  - `cp` – for copying files of any type
  - `mv` – for moving and/or renaming files of any type
  - `rename` – for multiple file renaming
  - `realpath` – to display the absolute path of one (or more) files
  - `mc` – the program “Midnight Commander”, having two panels with text-mode UI, for various file manipulations

Demo: see the examples [Basic file operations #1], ii)-iv) and [Basic file operations #2], i)-ii) in the lab support, available here.

17 / 19

Other commands for working with files

- Commands for various processing of file contents:
  - `cat`, `tac`, `more`, `less`, `head`, `tail`, `mcview` – for displaying, in various formats, the contents of a file (or more) of the normal type
  - `file` – provides clues about the “type of information” in a file, based on inspection of its contents
  - `stat` – for displaying attributes (*i.e.*, associated metadata) about a file
    (*Note*: `stat` is a more “powerful” command than the `ls -l` command.)
  - `grep` – for selecting lines of text, containing a certain pattern, from a file (or more) of the normal type (*i.e.*, this command does “horizontal selection”)
  - `cut` – for selecting specific columns of text from a file (or more) of normal type (*i.e.*, this command does “vertical selection”)

Demo: see the examples [Basic file operations #3], [Basic file operations #4], [cut #1] and [grep #1] in the lab support, available here.

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_lab1.html`

Additional bibliography:

[3] The documentation of *help* commands: `man 1 man` , `man 1 whatis` , etc.

[4] The documentation of plain text editors: `man mcedit` , `man nano` , `man vim` , etc.

[5] Information about GCC (*the GNU Compiler Collection*)

[6] The documentation of remote login commands: `man 1 ssh` , `man 1 scp` , etc.

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