Unix. File system and Command-Interpreter (shell)

Concepts

Absolute and relative filenames, path, file permissions, owner, links, device independence, redirection, pipes, filters.

Description

The Unix file system is organized as a tree, with the possibility of establishing additional links (so that it is actually a graph). The objective of this laboratory is to become familiar with the Unix file system and start working interactively using a shell: names, directories, files, paths, and links. By using filters (through pipes) and the redirection mechanism of the shell we will observe the device independence of Unix. Finally, we will analyze several features offered by the shell.

Steps

Solve the proposed exercises, practicing with the ***bash*** command interpreter.

Documentation

- Unix class notes.
- Unix online help (***man***).

Linux basics

Unix features

+ Multiprogrammed OS, multiuser and multi-terminal (interactive)

+ Standard for minicomputers and workstations

+ Powerful shell

  . redirections (< > >>)
  . *pipes* for inter-process communication (|)
  . *background* tasks (&)

+ Many utilities/tools

  . linguistic applications
  . communication
  . compilers/linkers/debuggers
  . text-processing

- Commands are rather cryptic

Command syntax

Special symbols:

<	standard input redirection (example: `prog < input_file`)
>	standard output redirection (example: `prog > output_file`)
>>	output redirection in append mode (example: `prog >> output_file`)
<<	input data comes after (used in shell scripts)
&	background task, concurrent with the next commands (example: `prog &`)
|	concurrent tasks: left's output is right's input (example: `prog1 | prog2`)

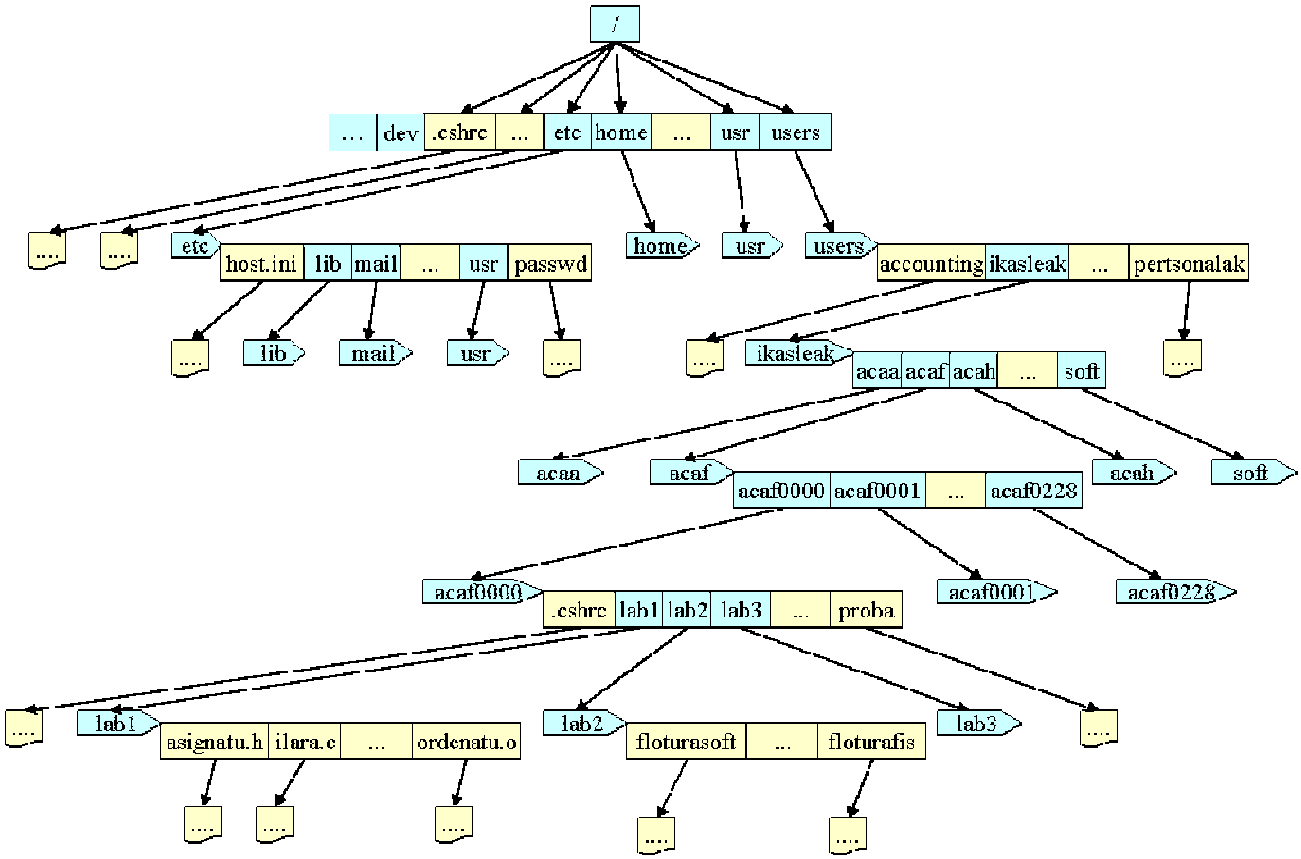
(Only the input/output of programs using the standard input/output can be redirected. Similarly, the program placed at the left/right of a pipe in a command must use its standard input/output)

Replacing characters (meta-characters):

*****	replaces any sequence of characters (string), including the empty one
?	replaces any single character

When any of these characters is found in a command, the shell (***bash*** in our case) takes it as a pattern, replacing it by the list of entries in the specified directory corresponding to that pattern (alphabetically ordered).

Example of a Unix file system



- Tree structure, with a single root (/).
- Devices need to be mounted/un-mounted (usually automatically)
- Filenames = [path/]name
  - absolute path (from the root) `/dir1/dir2/dir3/file`
  - relative path (from current directory) `file`
    `dir3/file`
    `../../dir2/dir3/fitx`
    `./` current directory
    `../` parent directory

Basic commands

- Exiting the shell (finishing the working session): ***exit, logout***

```
exit
logout
```

- Changing our password: ***passwd***

```
passwd
```

- Getting the online help about a command: ***man***

  ```
  man command
  ```

  Shows in the standard output the information related to that command.

  Example:

  ```
  man ls
  ```

- Listing the content of a directory: ***ls***

  ```
  ls [options] [dir]
  ```

  Shows in the standard output the entries of the specified directory.

  Some options:

  -a all entries, including those whose name begins by point (.)

  -l full information: permissions, links, owner, group, size, dates, name

  -R subdirectories are also shown (in a recursive way)

  Examples:

  ```
  ls
  ls -a
  ls -al
  ls –l dir1
  ```

- Changing the current directory: ***cd***

  ```
  cd [dir]
  ```

  Changes the current directory to the specified one. The name of the new directory can be indicated in an absolute (form the root) or relative way (from the current directory). If no directory is indicated, it changes to the user's home directory.

- Knowing which the current directory is: ***pwd***

  ```
  pwd
  ```

  Shows in the standard output the absolute path of the current directory.

- Creating a directory: ***mkdir***

  ```
  mkdir [path/]directory
  ```

  Of no path is indicated, the new directory is created in the current directory.

- Removing a directory: ***rmdir***

  ```
  rmdir [path/]directory
  ```

  If the directory is not empty, the command fails and gives an error message.

- Showing messages in the standard output related to arguments: ***echo***

  ```
  echo arg1 arg2 … argN
  ```

- Copying files: ***cp***

  ```
  cp file_source file_destination
  cp file_source1 … file_sourceN dir_destination
  cp –R dir_source dir_destination
  ```

  Allows copying a file. Allows also copying a directory. Allows also copying files to another directory.

- Changing the name of a file: ***mv***

  ```
  mv source destination
  mv source1 … sourceN dir_destination
  ```

  Allows changing the name of a file/directory. If the last argument is the name of an existing directory, then it moves all the previous files to that directory, keeping their names unchanged.

- Removing files: ***rm***

  ```
  rm file1 … fileN
  rm –R dir
  rm –i file
  ```

  Removes all the files passed as arguments. Directories that are not empty are not removed (when the –R option is used, the directory and all its content are recursively removed). If the –i option is used, it asks a confirmation.

- Showing the calendar in the standard output: ***cal***

  cal [month [year]]

  Shows the current month. If a month or/and year is indicated, then it is shown.

- Showing the full content of a file (also for concatenating files): ***cat***

  cat file_list

  Shows in the standard output the content of the files passed as arguments, concatenating them. If no argument is passed, "echoes" the standard input.

  Examples:

  ```
  cat file1.txt
  cat file1.txt file2.txt
  ```

- Showing the content of a file screen by screen: ***more***

  more file

- Showing the first lines of a text file: ***head***

  head [-n] file

  If no number is indicated, the first 10 lines are shown.

  Examples:

  ```
  head file1.txt
  head -18 file1.txt
  ```

- Showing the last lines of a text file: ***tail***

  tail [-n] file

  If no number is indicated, the last 10 lines are shown.

  Examples:

  ```
  tail file1.txt
  tail -8 file1.txt
  ```

- Filters: ***grep***

  ```
  grep [-v] pattern file_list
  ```

  Analyzes the content of the files passed as arguments, showing in the standard output the lines that contain the pattern passed as argument. If no file is passed, it takes the standard input as input. If the –v option is used, it shows the lines not containing the pattern.

- Links: ***ln***

  ```
  ln [-s] existing_name new_name
  ```

  Hardware link: without the -s option. It creates a new entry in the specified directory, which points to the *inode* (attributes + data) of the file whose name is passed as argument (link by *inode*).

  ```
  ln existing_name new_name
  ```

  Software link: with -s option. It creates a new entry in the specified directory, which points to the name (path) of the file whose name is passed as argument (link by *name*).

  ```
  ln -s existing_name new_name
  ```

- Editing a text file: ***nano, emacs, vi…***

  ```
  nano text_file
  emacs text_file
  vi text_file
  ```

- Changing file/directory permissions: ***chmod***

  ```
  chmod mode file_list
  ```

  It changes the read (r), write (w) or execution (x) permission of the files passed as arguments. The change can be applied to the owner of the file (u), to the group (g), to the rest of users (o) or to all of them (a). Permissions are coded as a sequence of 9 bits.

The mode argument indicates the new permissions, in two possible ways: as three octal numbers (4: read, 2: write, 1: execution), or as a sequence of characters (u|g|o|a for indicating the user, +|- for adding/removing permissions, and r|w|x for indicating which permissions are going to be modified).

Examples:

```
chmod 740 *.txt       (rwx r-- ---)
chmod u+x file        (put 'x' to owner)
```

Some additional commands

- Showing the list of connected users: ***who, w***
- Showing the active processes: ***ps***
- Terminating a process: ***kill***

```
kill [-9] process_id
```

- Preparing a file for printing: ***pr***
- Printing a file (*spooling*): ***lpr***

```
lpr –m file1          (message at the end)
lpr -Plaser file1     (printer name: laser)
```

- Showing the printer queue: ***lpq***

```
lpq –Plaser           (printer name: laser)
```

- Showing/changing permission mask: ***umask***
- Changing file owner and group: ***chown*** and ***chgrp***
- Finding files in the file system: ***find***

```
find / -name file -print
```

- Duplicating the standard output (showing it and storing it in a file): ***tee***
- Counting lines, words and characters: ***wc***
- Sorting lines alphabetically: ***sort***
- Showing the Unix information: ***uname***

- Showing for how long is the system running: ***uptime***
- Showing the use status of the file system: ***df***

Examples

`rm *.txt`	remove all files whose name finishes by .txt
`cat maiz?.txt`	concatenate all files whose name begins by "maiz", ends with .txt and have one character **in the middle**
`lpr maiz[1-3].txt`	print files maiz1, maiz2 eta maiz3.txt
`cat file1 >file2`	equivalent to *`cp fitx1 fitx2`*
`head file1 >>buru`	Append the first 10 lines of file file1 to the end of file buru
`man cp >lis1 &`	copies the man page of the cp command to file lis1, allowing executing new commands (concurrently)
`head -50 file1 | lpr`	prints the first 50 lines of the file file1

Proposed exercises

1. Exit the system and log in again.
2. Change your password.
3. Execute the following commands:

```
man man
man echo
man 1 echo
```

4. Execute the following commands and analyze the results:

```
echo Kaixo   zemuz   zaude?
echo “Kaixo   “ zemuz   zaude?
echo “Kaixo    zemuz   zaude?“
echo *
echo ‘*’
```

5. Get the absolute name of the current directory.
6. Show the content of the current directory.
7. Show the ***complete*** content of the current directory.
8. Show the content of the `/users/alumnos` directory.
9. Place in your *`home`* directory and test If these two command are functionally equivalent or not:

```
cd ../../../../usr
cd /usr
```

10. Create a directory **Lab1** in your home directory.
11. Go to the created directory (*Lab1*) and create the following directory structure:

```
---|-- Programs ----|---- bin
   |                |---- data ------|---- January
   |                |                |---- February
   |                |                |---- March
   |                |---- lib
   |-- Leisure
   |-- Rubbish
```

12. Go to the *data* directory. Analyze which is the resulting directory when executing each one of the following commands (commands are independent of each):

```
cd  ../..
cd  ../../Leisure
cd  ../Leisure
cd  January/../../../Leisure
cd  January/../../Leisure
cd  /users/alumnos/acaf/acafxxxx/Lab1/Leisure
cd  $HOME/Lab1/Leisure
cd
```

13. Go to your *home* directory and create a software link SOFT to the following directory: `/users/alumnos/soft/acaf/`
14. Go to the created SOFT directory.
15. Go to your `Lab1` directory and create a file `example.txt` (using the `nano` editor) of at least 40 lines of text.
16. Copy the file `example.txt` into a file `example1.txt`.
17. Copy the file `example1.txt` into the `Leisure` directory.
18. Copy the file `example.txt` into the `Leisure` directory with the name `example2.txt`.
19. Rename the file `example2.txt` of the `Leisure` directory to the name `example4.txt`.
20. Got to the `Lab1` directory and create a subdirectory `Leisure2`.
21. Copy all the content of the `Leisure` directory to the `Leisure2` directory.
22. Go to the `Lab1` directory and execute the following command, analyzing the result: `cp –R Leisure Leisure3`
23. Execute the following commands and compare their results:

```
cat example1.txt
more example1.txt
```

24. Get the list of users that are currently connected to the machine.
25. Go to the `Leisure` directory and get the first 10 lines of the `example1.txt` file.
26. Get the last 10 lines of the `example1.txt` file.
27. Get the last 4 lines of the `example1.txt` file.
28. Get the lines from 21 to 30 (both included) of the `example1.txt` file.

29. Get the number of lines of the `example1.txt` file.
30. Get the number of words of the `example1.txt` file.
31. Get the number of characters of the `example1.txt` file.
32. Get the lines of the `example1.txt` file containing the text "`hello`".
33. Get the number of lines of the `example1.txt` file containing the text "`hello`".
34. Get the lines of the `example1.txt` file NOT containing the text "`hello`".
35. Copy the file `example1.txt` to a file `example3.txt`.
36. Copy the file `example1.txt` to a file `example33.txt`.
37. Copy the file `example1.txt` to a file `example333.txt`.
38. Execute the following commands and analyze the results:

```
ls example*
echo ls example*
ls example?.txt
echo ls example?.txt
echo example?.txt
echo example*
```

39. Remove the file `example333.txt`.
40. Execute the following command and analyze the result:

```
echo rm *
```

41. Get the calendar of the current month. Also that of September of 1752. What's wrong?
42. Execute the following commands and analyze the results:

```
cat example1.txt
cat example1.txt > result.txt
ls
cat result.txt
ls > result.txt
cat result.txt
```

43. Create the file `a1.txt` with three lines of text inside.
44. Execute the following commands and analyze the results:

```
cat a1.txt
cat a1.txt > result.txt
cat result.txt
cat a1.txt >> result.txt
cat result.txt
ls >> result.txt
cat result.txt
```