Introduction to Unix

Éric Renault

Organization

- Tutorial 1 — 12/10, 14h-17h30 — INT B03
  - Introduction
  - Filesystem
- Tutorial 2 — 19/10, 14h-17h30 — INT B08
  - Process management
- Tutorial 3 — 26/10, 14h-17h30 — INT B02
  - Communication
  - Save / Restore
  - Visual editor
- Tutorial 4 — 02/11, 14h-17h30 — INT B07
  - Script shells
- Tutorial 5 — 09/11, 14h-17h30 — INT B08
  - Script shells
- Tutorial 6 — 16/11, 14h-17h30 — INT B08
  - LaTeX
  - Makefile

Introduction to Unix

Éric Renault

Institut National des Télécommunications
9, rue Charles Fourier
91011 Évry Cedex, France

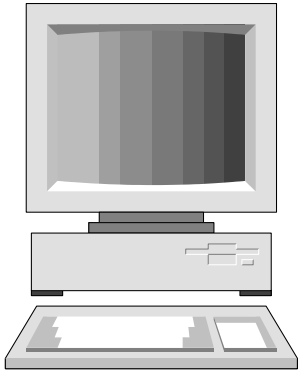
Contents

INTRODUCTION

Introduction

What is a machine ?

- Central unit
  - Processors
  - Memory
  - Cache memory ...
- Devices
  - Keyboard
  - Monitor
  - Mouse ...



What is an operating system ?

History

Dates :

- 1969 : Single user version written in assembly
- 1970 : First multi-user version
- 1972 : Pipes
- 1973 : First version written in C
- 1980 : First version from Berkeley

Available versions (some of them) :

- Freeware : FreeBSD, Linux
- Others : AIX, HP_UX, NeXT, SCO, Solaris, SunOS

Characteristics

- Multi : user, task, processes, processors
- Simple
- Written in C (understandable by everyone)
- Open source (most of the versions)
- Efficient and secure filesystem
- Everything is seen as a file :
  - “Classic” files, directories
  - Devices, processes, memory, ...
- Powerful user interface
- Multi-level system

Evolutions

- Graphical ...
  - ... interfaces :
    - XFree, Xwindows
  - ... environments :
    - cde, ctwm, fvwm, gnome, kde, Motif, twm
- Normalization :
  - POSIX from IEEE
- Real time :
  - RT-Linux, Unix RT

Bibliography

- The Design of the UNIX Operating System.

  M.J. Bach.

  Prentice Hall, 1986.
- Operating Systems : Design and Implementation.

  A.S. Tanenbaum.

  Prentice Hall, 1987.
- The Design and Implementation of the 4.4BSD Operating System.

  M.K. Mc Kusick, K. Bostic, M.J. Karels and J. Quarterman.

  Addison Wesley, 1996.

Login / Logout

Session

- Login
  - Enter the login and the password

```
Login :
Password :
```

- Change the password
  - Enter the current password and the new password twice in order to avoid errors
  - `passwd` or `yppasswd`
- Logout
  - At the end of the session
  - `logout` or `exit`

Syntax

```
Command [ Options ...  ]  [ Parameters ...  ]
```

`Options`

- Change the behavior of the command
- Usually agglutinative

`Parameters`

- Objects the command is working on

Convention

- <*a*> means that *a* must be replaced by its value

  Example : `<user>` means that a user name must be used
- *a*|*b* means *a* or *b*

  Example : `cols|rows` means that both `cols` or `rows` words can be used
- [*a*] means that *a* is optional

  Example : `[-l]` means that `-l` may be used
- {*a*} means that *a* is mandatory

  Example : `{u|g|o|a}` means that at least one character from the list must be used

First commands

Help and calendar

Online help

- `man [<section>] <cmd>`
- `whatis <cmd>`
- `apropos <string>`
- `info <cmd>`

Calendar

- `date`
- `cal [[<month>] <year>]`

User and host information

User information

- `who`
- `whoami`
- `finger [-l] [<user>]`
- `logname`

Host information

- `hostname`
- `uname [-a]`

Environment

Environment

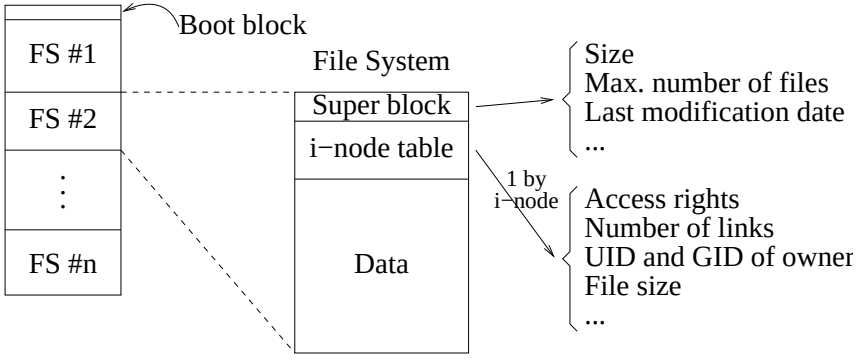
- `stty [-a|{cols|rows} <value>]`
- `env`

Display

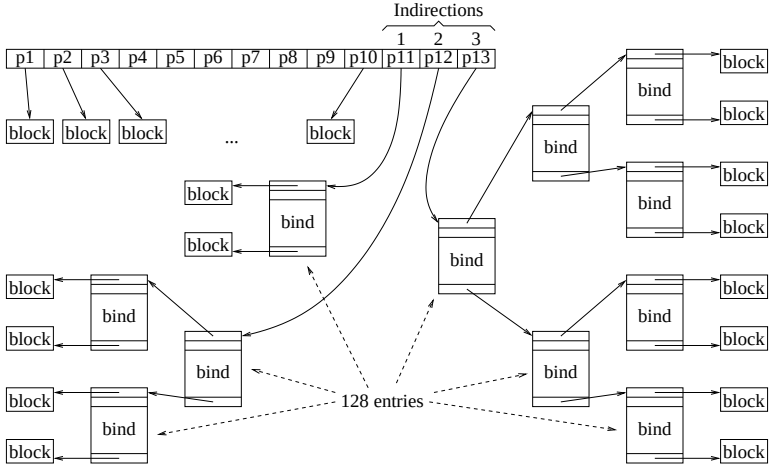
- `echo [-n] <text>`
- `banner [<text>]`
- `clear`

THE FILE SYSTEM

Disk structure

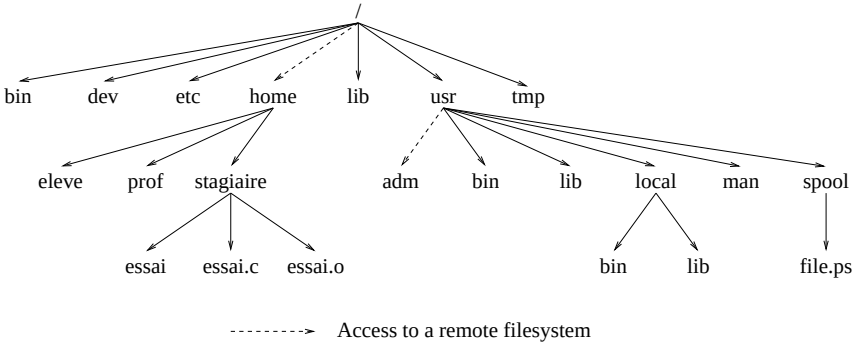


Block access



Maximum size for a file : $512 \times (10 + 128 + 128^2 + 128^3) \simeq 1$ GB

Tree structure



Name and types

Name :

- `[a-zA-Z0-9_.]+`
- Avoid the other characters
- Hiden files (start with a `.`)

Type :

- `-` : “regular” file
- `b` : block device
- `c` : character device
- `d` : directory
- `l` : symbolic link
- `p` : named pipe

Security

Three levels :

- User
- Group
- Other (the rest of the world)

Three types :

- Read
- Write
- eXecution

Two ways of specifying :

- `{u|g|o|a} {+|-|=} {r|w|x}`
- Octal : `r`=4, `w`=2, `x`=1

Example (1/2)

Let the following be the caracteristics of a file :

```
-rw------- 1 smith prof 158979 Aug 13 21:32 document.ps
```

This means that :

- This is a regular file
- Only the owner can read and/or modify the file
- There is no other link to this file
- It is owned by `smith`
- It belongs to group `prof`
- It is composed of 158,979 bytes

Example (2/2)

Let the following be the caracteristics of a file :

```
-rw------- 1 smith prof 158979 Aug 13 21:32 document.ps
```

This means that :

- It was last modified on August 13 at 21:32
- Its name is `document.ps`

To make this file readable by everybody :

- `chmod 644 document.ps`
- `chmod a+r document.ps`

Directory

Structure :

- Files containing couples (name ; i-node)

Special directories :

- `/` : the root of the filesystem
- `.` : the current directory
- `..` : the parent directory
- `~`: the home directory

Access :

- Absolute : from the root of the filesystem
- Relative : from the current directory

BASIC COMMANDS

Disk and directory

Disk :

- `df`
- `du [-{s|k}] [<dir>]`

Directory :

- `cd [<dir>]`
- `ls [-{a|l|r|t}] [<file>]`
- `mkdir <dir>`
- `pwd`
- `rmdir <dir>`

File

Information :

- `file <file>`
- `wc [-{c|l|w}] [<file>]`

Management :

- `cp [-R] <file> <file>`
- `ln [-s] <file> <file>`
- `mv <file> <file>`
- `rm [-{r|f}] <file>`
- `touch <file>`

Filters

Content :

- `cat [<file>]`
- `more [<file>]`

Selection :

- `cut {-c|[-d'<sep>'] -f<list>} [<file>]`
- `head [-{c|n} <number>] [<file>]`
- `tail [-{c|n} [+-]<number>] [<file>]`
- `uniq [-c] [<file>]`

Security and name

Security :

- `chmod [-R] <mode> <file>`
- `chown [-R] <user>[:<group>] <file>`
- `chgrp [-R] <group> <file>`

Operations on filenames :

- `basename <string> [<ext>]`
- `dirname <string>`

ADVANCED COMMANDS

`sort`

Synopsis :

- `sort [-{n|r}] [-t'<sep>'] [-k<list>] [<file>]`

Examples :

- Sort `/etc/passwd` file using the GECOS field

```
sort -t':'  -k5 /etc/passwd
```

- Sort `/etc/group` by GID (third field)

```
sort -n -t':'  -k3 /etc/group
```

grep (1/2)

Synopsis :

- `grep [-{l|n|v}] <pattern> [<file>]`

Special characters for the pattern :

- `^` : at the beginning of the line
- `$` : at the end of the line
- `*` : the previous character repeated from 0 to n times
- `.` : any character
- `[<list>]` : one character in the list
- `[^<list>]` : one character out of the list

grep (2/2)

Examples (using file `/etc/passwd`) :

- Display the lines beginning with an 'n'
  ```
  grep '^n' /etc/passwd
  ```
- Display the lines which do not begin an 'n'
  ```
  grep -v '^n' /etc/passwd
  ```
- Display the number of lines containing 'var'
  ```
  grep -n 'var' /etc/passwd
  ```
- Display the lines containing 'var' and then 'nologin'
  ```
  grep 'var.*nologin' /etc/passwd
  ```


find (1/2)

Synopsis :

- `find <dir> [<predicate>]`

Main predicates (see online manual) :

- `-exec <cmd> \;` : execute a cmd (current file : {})
- `-group <group>`: file depending group
- `-name <pattern>` : file name
- `-print` : display file name
- `-type <type>` : type of file
- `-user <user>` : file owner

find (2/2)

Examples :

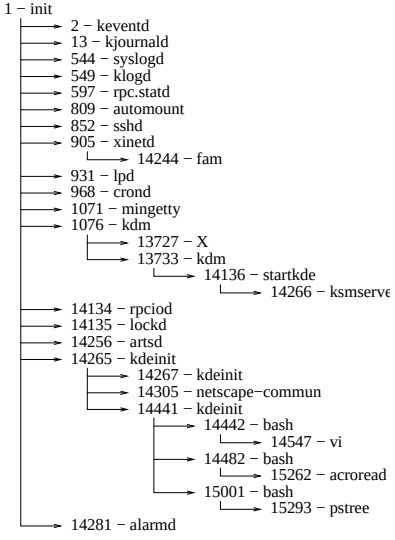
- Where are all the `.c` files of the filesystem ?
  ```
  find / -name '*.c' -print
  ```
- Where is directory `toto` in my home directory ?
  ```
  find ~ -type d -name 'toto' -print
  ```
- What is the number of lines of `.h` files in `/usr` ?
  ```
  find /usr -name '*.h' -exec wc -l {} \;
  ```
- Which files, in my home directory, are owned by another user ?
  ```
  find ~ -not -user 20319 -print
  ```
  where `20319` is my user ID ...

PROCESS MANAGEMENT

Processes

- Program in execution (or waiting for execution)
- One process per processor at a given time
- *Ticks* are usually lasting around 10 ms
- Several processes per user ; one user per process
- Identified by a PID (Process id)
- Main state :
  - `Run` : in execution
  - `Sleep` : waiting for an event
  - `Stop` : do not request for execution
  - `Zombie` : waiting for an ack. from the father

Process tree



- The system starts with process 1
- To create a new process, the system duplicates an existing one
- The original process is called the father of the new process
- The new process is called the son

Commands

Commands :

- `bg [%<job>]`
- `fg [%<job>]`
- `jobs`
- `kill [-<signal>] {<pid>|%<job>}`
- `ps [-{a|u|x|w}]`
- `sleep <number>`

Keyboard :

- Terminate current process : `^C`
- Stop current process : `^Z`

SEPARATORS AND REDIRECTIONS

Separators

Separators :

- Sequential : `<cmd> ; [<cmd>]`
- Parallel : `<cmd> & [<cmd>]`

Conditional executions :

- Success : `<cmd> && <cmd>`
- Fail : `<cmd> || <cmd>`

Composition :

- Creation of a sub-shell : `( <cmd> ; <cmd> )`

Input / output redirection

Goal :

- Diverting information flow

Communication with a process :

- one standard input (`IN`) : `0`
- one standard ouput (`OUT`) : `1`
- one standard error output (`ERR`) : `2`

Default attachment :

- `IN` : keyboard ; `OUT` and `ERR` : window

Use of redirection (1/2)

To the process :

- One “less than” symbol

  `<cmd> n< <file>`

From the process :

- One “greater than” symbol (with creation)

  `<cmd> n> <file>`
- Two “greater than” symbols (with appending)

  `<cmd> n>> <file>`

Use of redirection (2/2)

From a flow to another one :

- Both `>` and `&` symbols

```
<cmd> n>&p
```

Default values :

- Input : `0`
- Output : `1`

Example :

- `myprog < file1 > file2 2>&1`

Pipes

- In order to allow a process to use the results from another process, one must use a temporary file :

```
<cmd1> > <file>
<cmd2> < <file>
rm <file>
```

- This is long and annoying
- Unix proposes a shortcut : the pipe `|`

```
<cmd1> | <cmd2>
```

- The standard output of `<cmd1>` is diverted to the standard input of `<cmd2>` ; the standard error output does not change

COMMUNICATIONS

Remote access

Remote login :

- `telnet <host>`
- `rlogin [-l <user>] <host>`

- Remote execution :
- `rsh [-l <user>] <host> [<cmd>]`

  → The `.rhosts` file on the remote account must allow connection without password
- `ssh [<user>@]<host> [<cmd>]`

Transfer

Transfer :

- `[s]ftp <host>`
  → `bi`, `cd`, `get`, `prompt`, `put`
- `scp [<user>@]<host>:<path> <path>`

Localization :

- `rusers`

Load :

- `rup`

Communication

Interactive :

- `talk <user>`

Remote write :

- `write [<user>]`
- Close the message with `^D`

Protection :

- `mesg [n|y]`

Asynchronous :

- `mail [<user>]`
- Close the message with a `.` on the first character of a line

Other available mailer :

- `elm`, `mutt`, netscape, ...

PRINTER AND ARCHIVES

Printer

List :

- `lpstat`

Queue :

- `lpq`

Print :

- `lp [-d <printer>] [-n <number>] <file>`
- `lpr [-P <printer>] [-# <number>] <file>`

- Cancel :
- `lprm [-P <printer>] [<job>]`
- `cancel [<job>]`

Data

Save / Extract :

- `tar [ctx][v][z]f <archive> <file>`

Compress :

- `gzip [-{0-9|d}] <file>`
- `[un]compress <file>`

`vi`

`vi` text editor (1/4)

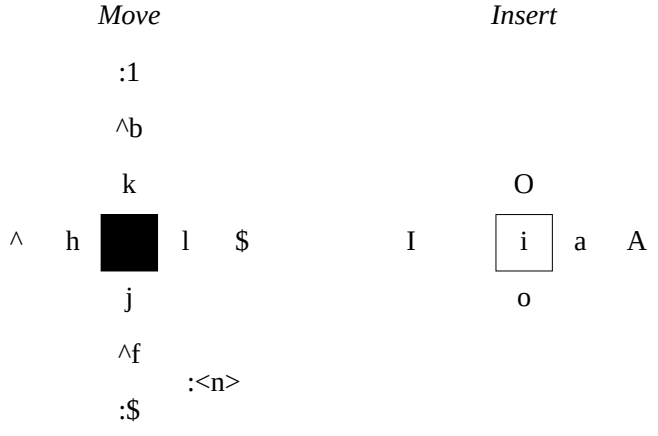
Caracteristics :

- Standard over UNIX
- Very powerful
- Not a full-screen editor

Comparison :

- Allows to perform the same operations as the other text editor (especially graphic ones)
- Can use regular expressions

`vi` text editor (2/4)



Small online help : `viusage`

vi text editor (3/4)

Cut :

- Characters : `[<n>]x`
- Words : `[<n>]dw`
- Lines : `[<n>]dd`

Copy :

- Lines : `[<n>]yy`

Paste :

- Before : `P`
- After : `p`

Replace :

- One character : `r`
- Several characters : `R`
  → Stop with `<esc>`

vi text editor (4/4)

Search :

- Asc : `?[<pattern>]`
- Desc : `/[<pattern>]`
- Repeat : `n`

Commands :

- Edit : `:e <file>`
- Write : `:w[!] [<file>]`
- Read : `:r <file>`
- Quit : `:q[!]`

Substitution :

- `:[<zone>]s/<pattern>/[<pattern>]/[g]`
- with `<zone>` equal to `<n>[,<m>]` or `%`

SCRIPT SHELLS

Script Shells

Variables :

- `$[0-9]` : i^{th} parameter → `shift`
- `$*` : all the parameters
- `$#` : number of parameters
- `$?` : returned value of the last command
- `$!` : PID associated to the last command
- `$$` : script PID

Comments :

- From character `#` to the end of the line
- `#!<path>` specifies the interpreter to use (1^{st} line)

Command substitution

Principle :

- `<var>=‘<cmd>‘`

Caracteristics :

- Divert the standard output of the command to the interpreter
- Newlines are replaced by spaces

Example :

- `CFiles=‘ls *.c‘`

Loops (1/2)

In extension

```
for <var> [in <list>]
do
    <cmd>
done
```

In comprehension

```
while <cond>
do
    <cmd>
done
```

Default for `<list>` is `$*`

Loops (2/2)

- `continue` :
  Step to the next iteration
- `break` :
  Step to instruction that follows the loop
- `return` :
  Quit the script or the function (depending upon the case)
- `exit [<value>]` :
  Quit the script

Tests

By comparison

```
  if <cond>
  then
      <cmd>
[ elif <cond>
  then
      <cmd> ]
[ else
      <cmd> ]
  fi
```

By analogy

```
  case <pattern> in
[ <pattern>)
      <cmd>
      ;; ]
  esac
```

Default analogy : `*`

The `test` command

- Two ways to use it : `test` or `[]`
- Allows to (see the online manual for options) :
  - test integers and strings
  - determine file state
  - use logic operator
- Example :

```
if [ -r $1 ] && [ -w $1 ]
then
    echo "$1 is readable and writable"
fi
```


Last commands

File substitution :

- `sed -e 's/<pattern>/<pattern>/[g]' [<file>]`

Data from the keyboard :

- `read <var>`

LaTeX

LaTeX (1/7)

- Text compiler
- Generates high-quality documents (postscript, PDF...)
- Focus on the content, LaTeX focuses on the form
- Automatic label, reference and bibliography generation
- Possible to cut a document in several parts
- Can be used to create slides (aren't they nice ?)
- Really easy to use
- Lots of documentation and tutorials

LaTeX (2/7)

- Structure of a LaTeX document:

```
\documentclass[options]{type}
    preambule
\begin{document}
    body
\end{document}
```

- Comments : from % to the end of the line

LaTeX (3/7)

- *type* : document type (e.g. `article`, `book`, `report`...)
- *option* : options related to the document type
- *preambule* :
  - package inclusion (e.g. `\usepackage{a4}`)
  - definition of new commands

  eg. `\newcommand{\me}{Éric}` replaces each "`\me`" in the body by `Éric`
  - new environments, new counters...

LaTeX (4/7)

- *body* : the document body

Include a title :

```
\title{TheTitle}
\author{TheAuthor}
\date{TheDate}
\maketitle
```

Hierarchical structure :

```
\part{Name}
\chapter{Name}
\section{Name}
\subsection{Name}
\subsubsection{Name}
\paragraph{Name}
```


LaTeX (5/7)

```
\documentclass[12pt]{article}
    \usepackage{a4}
\begin{document}
    \title{My beautiful article}
    \author{Myself in.  Person}
    \maketitle
    Hello world !  The doc is beginning...
    \section{First}
    This is the first section...
\end{document}
```

LaTeX (6/7)

How to do lists :

```
\begin{list}
      \item element 1
      \item element 2
      \item element 3...
\end{list}
```

where *list* may be `description`, `enumerate` or `itemize`

LaTeX (7/7)

Compilation steps :

1. Generate the `.dvi` document from the `.tex` file
   ```
   latex document.tex
   ```

Nota : Do it twice to include cross-references

2. Generate the `.ps` document from the `.dvi`
   ```
   dvips -o document.ps document.dvi
   ```
3. Generate the `.pdf` document from the `.ps`
   ```
   ps2pdf document.ps document.pdf
   ```

This may be a little bit more complicate when generating the bibliography automatically

MAKEFILE

Makefile (1/2)

- Used to perform operations automatically
- Operations are performed according to rules (explicit or implicit)
- The structure of a rule is :
  ```
  target :  src1 src2 src3 ...
        cmd
  ```
- If the last modification date of at least one of the `src` is newer than target, the (not up-to-date) target is rebuilt using `cmd` (may be more than one line)
- If a `src` is a target, the operation is done recursively
- Rules are in `makefile` or `Makefile`

Makefile (2/2)

- Explicit rules :
  - File names are given literaly

```
document.dvi :  document.tex
	latex document.tex
```

- Implicit rules :
  - Only extensions are specified : each file with the same extension can be applied the rule

```
%.dvi :  %.tex
	latex $<
```

  - Specific variables : `$<` for the sources, `$@` for the target...