

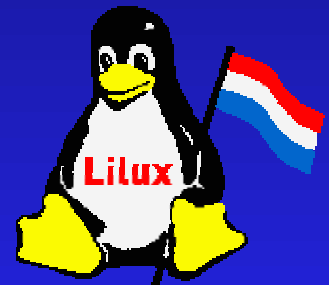
# Advanced training

Linux components

Command shell

LiLux a.s.b.l.

*alexw@linux.lu*



# Kernel

- Interface between devices and hardware
- Monolithic kernel
- Micro kernel
- Supports dynamics loading of modules
- Support for different networks and file systems

# Configuration of the kernel

## Under X

- Login as root
- Go to `/usr/src/linux`
- `make xconfig`

## In command line

- Login as root
- Go to `/usr/src/linux`
- `make menuconfig`

## Tips

- `make oldconfig`
- The kernel configuration file `.config`

# How to compile the kernel

- 1) make dep
- 2) Make
- 3) make install
  - Using lilo
    - /etc/lilo.conf
    - lilo
  - Using grub
    - /etc/grub.conf
    - Grub
      - root(hd0,0)
      - setup(hd0)
- 4) make modules
- 5) make modules\_install

# File systems

- ☞ ext2 / ext3
- ☞ reiserfs
- ☞ inodes
- ☞ raidtools
- ☞ Partition types (fdisk)
- ☞ SWAP
- ☞ mk2fs, mkreiserfs, mount, umount

# Network file systems

☞ NFS

☞ Samba

☞ Connectivity with Windows machine

☞ Samba client

☞ Samba server

# Network

- ☞ TCP / IP
- ☞ IP Tables
- ☞ Schedules (QoS)
- ☞ Other protocol supported
- ☞ Ethernet + VLANs
- ☞ Bridging
- ☞ Routing

# Command shell

- Several Shell commands make it easy to select information (by row and col) and prepare it for printing or processing.
- All files are simple flat files, but with delimiters like *tab* they are transformed into a spreadsheet or relational database.
- To get data from a spreadsheet or RDBMS you need to be able to select by row (**grep**, **head**, **sed**, **tail**, **uniq**) and col (**cut**).



# cat

- ☞ Takes one or more filenames as arguments, opens the file(s) and copies them to *stdout* (usually the terminal, but can be redirected).
- ☞ The real power of cat is being able to stream multiple files to *stdout*.
- ☞ Note: cat is not good at handling files with non-printable control characters, which can produce gibberish or lock your terminal.

# Streams, redirection, pipes

- ☞ Streams are the foundation for input and output
- ☞ shell provides it with three standard streams, with numeric identities 0, 1 and 2
  - Standard input
  - Standard output
  - Error messages
- ☞ keyboard --> it provides data for the standard input
- ☞ default output --> streams to your screen

# Streams redirections

👉 Redirection = file descriptor manipulation

- `my_program < input_data.file > output_data.file`

👉 `>>` is used to append data to a file

- `#!/bin/sh`
- `echo "Hello" > log.file`
- `date >> log.file`
- `echo "end of test" >> log.file`

# Pipes

- Two programs can be linked
  - the (standard) output from the first is sent in as the (standard) input to the second
- small utilities each of which performs just one simple task
- `cat input.file | tee log.file | my_program`

# List of Selection Commands

- awk pattern scanning and processing language
- cut – Select columns
- diff – compare and select differences in two files
- grep – select lines or rows
- head – select header lines
- sed – edit streams of data
- tail – select tailing lines
- uniq – select unique lines or rows
- wc – count characters, words, or lines in a file

# Line or Row Commands 1/2


👉 **head** allows previewing of files; to see the first ten lines:

- head file

👉 **tail** allows viewing of the last ten lines:

- tail file

## Line or Row Commands 2/2

-  To see data being written to a file, use the -f option with **tail**.
  - Tail gives the usual last ten lines, then sleeps
  - Every few secs it wakes and displays added lines
  - End with the break command (Delete key or Ctrl-C)

# Grep 1/2

- ☞ globally look for a regular exp and print.”
- ☞ Looks for character strings in files and writes the requested info on stdout.
- ☞ To get the entire lines containing a word:
  - `grep PATH /etc/profile`
- ☞ Use the `-l` option to just get filenames:
  - `grep -l PATH /etc/*`
- ☞ `-s` will remove the warnings/error messages

-



## Grep 2/2

- ☞ Use the `-n` option to learn the lines:
  - `grep -n PATH /etc/profile`
- ☞ For strings of more than one word use `""`:
  - `grep -"export PATH" /etc/*`
- ☞ To ignore differences in case use `-i`:
  - `grep -i pAtH /etc/profile`
- ☞ The `-v` option returns lines that do not contain the string or pattern specified:
  - `grep -v PATH /etc/profile`

# Regular expressions 1/2

- ☞ a,b,.. . general characters
  - \ before special characters mentioned below
- ☞ A B Concatenating regular expressions
- ☞ ( A ) (y) Groups sub-parts of a complicated expression
- ☞ A | B Alternation is written using a vertical bar, which may be read as OR
- ☞ A\*(y) Matches zero or more instances of it;
- ☞ A+(y) like the star operator, but accepts one or more instances

## Regular expressions 2/2

- ☞ `A?` (y) Zero or one matches for the given item
- ☞ `A\{n,m\}` From n to m repetitions
- ☞ `[a-z]` Matches a single character, which must be one of the ones listed within the brackets
  - The mark `^` can be used at the start of a pattern to negate the sense of a match
- ☞ `.` Matches any single character except a newline.
  - Thus `.*` matches any string of characters not including newlines
- ☞ `^` and `$` `^` matches at the start of a
- ☞ line, while a `$` at the end ensures that matches are only accepted at the end of a line

## Some examples 1/3

- ☞ A pattern that matches words that start with a capital letter but where the rest of the characters (if any) are lower case letters and digits or underscores
  - `[A-Z][a-z0-9_]*`
- ☞ The string `#include` at the start of a line, apart from possible leading blanks
  - `^ *#include`
- ☞ A line consisting of just the single word `END`
  - `^END$`
- ☞ A line with at least two equals signs on it with at least one character between them
  - `=.+=`

## Some examples 2/3

- 👉 Find which file (and which line within it) the string class LostIt is in, given that it is either in the current directory or in one called extras
  - `grep 'class LostIt' *.java extras/*.java`
- 👉 Count the number of lines on which the word if occurs in each file whose name is of the form \*.txt.
  - `grep -c "\<if\>" *.txt`

## Some examples 3/3

- As above, but then use `grep` again on the output to select out the lines that end with `:0`, ie those which give the names of les that do not contain the word `if`. This also illustrates that if no les are specified `grep` scans the standard input.
  - `grep -c "\<if\>" *.txt | grep :0\$\`
- Start the editor passing it the names of all your source les that mention some variable, presumably because you want to review or change just those ones. You could obviously use the same sort of construct to print out just those les, or perform any other plausible operation on them.
  - `emacs `grep -l some_variable *.java``

# wc command

- Can count the number of characters, words and lines in a file (useful to check the outcome of prior processing – often from a pipe).
- The three common options:
  - -l returns the number of lines
  - -w returns the number of “words”
  - -c returns the number of characters

# uniq command

- Removes identical lines from a file (gets rid of redundancy). For **uniq** to work the file must be sorted, so use `sort` followed by `uniq`.
- If filenames are not given, `uniq` uses *stdin* and *stdout*.
- The `-d` option reports the duplicate lines that exist in the file (just gives the line once).
- To get a count of how often lines repeated use the `-c` option.



# Column Commands

- ❏ **cut** actually cuts files into pieces that can be pasted back together in some other usable fashion.
- ❏ **cut** can operate on a char-by-char or a field-by-field basis or some combination of both.
- ❏ The **-f** option indicates the fields (start at 1 and can be used for ranges) and **-d** can be followed by a delimiter (tab is default):
  - `cut -f1,5 -d: <using stdin from a pipe>`

# sort command

- ☞ Sort keys the order fields used by sort
  - Fields can be delimited by white space (or others)
  - Fields can also be defined by character position
  - The order can be ascending (default) or descending
- ☞ **sort** is used most efficiently after **grep** and **cut** have done their selections (less rows).
- ☞ Sorted data often contains duplicate rows; **uniq** can remove or display those duplicate lines.

# Sort examples

- Sort the password file by user ID and then extract all users under the file /home:
  - `sort -t: +0 -1 /etc/passwd | grep home | cut -f1 -d:`
- It would be more efficient to extract the data first and then sort:
  - `grep home /etc/passwd | cut -f1 -d: | sort`
- To sort a long listing of a directory by the file size in bytes (descending numeric order):
  - `ls -al | sort +4nr`

# sed transformer command

- **sed** (stream editor) transforms incoming data by executing editor commands on *stdin*. It is used in pipes in place of the standard line editor *ed*.
- The original file is never altered, to save the output you must redirect to another file.
- For simple substitutions the editor commands can be put in the command line:
  - `sed -e "s/PATH/chemin/" /etc/profile`
  - `sed --e "s/PATH/chemin/g" /etc/profile`
  - `sed --e "s/PATH/chemin/g"`

# Some common sed edit commands

👉 Delete lines from a file

- “1,5d”

👉 Insert text in the output file:

- “i/this is an inserted line/”

👉 Append text in the output file:

- “a/This is an extra line/”

👉 Change lines:

- “c/This is a changed line/”

👉 Print (select) certain lines from a file:

- “1,5p”

👉 Write changed rows to a file:

- “w outputfile”

## More examples

👉 **sed** can work like a selection command; the `-n` option tells it to print just the lines that were selected with the `p` option:

- `sed --n "1,10p" file1 > outfile`

👉 **sed** can also be used to remove certain characters from a file:

- `... | sed -e "s/[ \ta-zA-Z] [ \ta-zA-Z]*// g`

# Questions & Answers

