

1970s: Unix philosophy



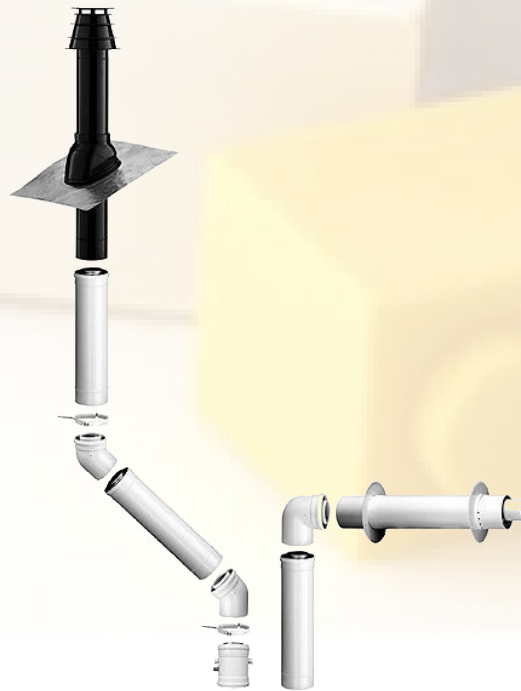
Or



- instead of a “do-it-all“ tool, the Unix designers thought of a set of specialized tools with some way to assemble them.
- each tool does only 1 task, but it does it well

the pipe metaphore

- the data (as ascii) is a flow. It starts at a source, goes through pipes from a device (filters, etc) to the next one. The flow can be split (using a Tee). Finally, it goes into a bucket.



Environnement

- All that follows is used through the command line (terminal window)
- use arrows to navigate up and down in history and back and forth in the line
- use [tab] key for autocompletion

Sources

these are like legos' base plate: you built on them

- *cat {file}: output file*
- *head -n 10 {file}: output the first 10 lines (bytes is -c)*
- *tail {file}: output the last lines/bytes*

Example: *head -25 my_data.dat*

Help

help available with *-h* or *-help* options or using
online manual: *man {command}*

head --help

Usage: head [OPTION]... [FILE]...

Print the first 10 lines of each FILE to standard output.

With more than one FILE, precede each with a header giving the file name.

With no FILE, or when FILE is -, read standard input.

Mandatory arguments to long options are mandatory for short options too.

-c, --bytes=[-]N	print the first N bytes of each file; with the leading '-', print all but the last N bytes of each file
-n, --lines=[-]N	print the first N lines instead of the first 10; with the leading '-', print all but the last N lines of each file
-q, --quiet, --silent	never print headers giving file names
-v, --verbose	always print headers giving file names
--help	display this help and exit
--version	output version information and exit

N may have a multiplier suffix: b 512, k 1024, m 1024*1024.

Advanced Sources

- *curl: download a file on the Internet*
- *find: finds a file on the disk (by name, size, owner, time of modification, several of these together...)*
- *any other tool that usually sends some data to the screen*

filters

these remove or keep only part of the data

- *grep {expression} {file}: keep only what matches (or the opposite with -v option)*
- *tr {expression} {translation}: replaces a character by another one*
- *sort {file}: sorts by various orders, reverses, keep only unique entries...*
- *cut: extract fields delimited by a given delimiter*

Regular Expressions

syntax to express some pattern. Used by Unix tools, Perl, PHP, AWK, available in C, ...

- 'slf' matches only 'slf'
- '.' matches any character
- '^' matches the beginning of the line
- '\$' matches the end of the line
- '[' matches any of the characters in the []. ex:
[abc] matches a or b or c
- '[^]' matches any character NOT in the []

Regular Expressions II

- 'a-z' matches any character between 'a' and 'z'
- '()' groups elements together
- '*': zero or more copies of what is before
- '+': one or more copies of what is before
- '?': zero or one copy...
- '{x,y}': matches what is before between x and y times
- '|': logical OR

Regular Expressions: examples

- '^#' matches a common way to do comments
- 'CH-[0-9]{4}' matches a zip code
- '[a-zA-Z]@slf\.ch' matches an slf email (the '.' is escaped)
- '^2007-01-[0-9]?[1-9]' matches a common timestamp

sometimes, the *-E* option has to be given to force Extended regular expressions

The assembly...

- each tool is connected to the next one by a piece of pipe: '|'
- so, to read from file 'test.dat', filter the comments out, extract field 1 (the time) and field 3 (the temperature), keep only one record every 20 minutes and sort in reverse order, we do:

```
cat test.dat | grep -v “^#” | tr -s ' ' | cut -d' ' -f1,3 |  
grep -E “[0-9]{1,2}:[024]0:00” | sort -n -r
```

Lines like '10:20:00 5.2365' are sent to the screen...

Redirections

- to send the output to a file: > file
- to append to a file: >> file
- there are in fact two streams: the standard output and the standard error
- to send the standard error to a file:>&2 file
- to merge stdout and stderr: ... &> file

Advanced flow control!

- to send to a file while keeping the processing going: use tee!
- counting words, characters, lines: wc
- to send data over the network: netcat, command 'nc'
- named pipes: the same as a pipe, but visible as a file on the disk (create with 'mkfifo {name}')

Executing commands

xargs: takes input from stdin, replaces arguments by this input:

- *find . -type f | xargs -i chmod u+x {}* finds all files from the current location and does a *chmod u+x {file}* on each one of these (the '{}' is replaced by the line given on the input, looped for all lines).

Saving a script to a file

- such a command line can be written into a file for repeated execution
- use an extension like '.sh' to remind you that this is not a binary
- do a *chmod +x {file}* to have it executable

Example

Count the number of unique words used in a LaTeX file:

```
cat task1.tex | tr -s ' ' '\n' | grep -v "^%" | grep -vE  
"^(\\|\\)" | tr -d ',.:)(;' | sort -u | wc -l
```

Examples

kill all 'Acrobat' processes belonging to bavay or mathias

```
ps aux | grep -E “^(bavay/mathias)” | grep  
“Acrobat” | tr -s ' ' | cut -d' ' -f2 | xargs -i kill -9 {}
```

Examples

Computer A runs an acquisition program that writes to a file. We want to send it to computer B in real time, removing comments and replacing NaN values by '0'.

On computer A:

```
mkfifo data  
acquisition_program.bin -file data  
cat data | nc 192.168.1.2 4000
```

On computer B:

```
nc -l 4000 | grep -vE "^#" | tr 'Na' '@' | tr -s  
'@' '0'
```

Examples

From a data file, extract all measurements made in January 2007, keep comments but convert them to lowercase

```
cat data.dat | grep -E "(^2007-01-[0-9]?[1-9])|(^#)"  
| tr [A-Z] [a-z]
```

More complex processing

- Combine pipes with shell scripting
- Combine pipes with AWK (actions associated with patterns)

