

**[Software Development]**

# ***Linux ToolSet (part B)***

*Davide Balzarotti*

Eurecom – Sophia Antipolis, France

## Locate the source of Information

ls  
find  
locate  
shell wildcards

---

## Locate the source of Information



```
graph TD; A[Locate the source of Information] --> B[ ]; B -.-> C[ls  
find  
locate  
shell wildcards]; B -.-> D[for, while  
find -exec  
xargs];
```

```
ls  
find  
locate  
shell wildcards
```

```
for, while  
find -exec  
xargs
```

**Locate the source of Information**

```
ls  
find  
locate  
shell wildcards
```

```
for, while  
find -exec  
xargs
```

**Extract the Data**

```
cat  
extract  
pdftotext, ...  
wc
```

**Locate the source of Information**

```
ls  
find  
locate  
shell wildcards
```

**Extract the Data**

```
for, while  
find -exec  
xargs
```

```
cat  
extract  
pdftotext, ...  
wc
```

**Filter the Data**

```
grep  
head, tail  
sort, uniq
```

**Locate the source of Information**

```
ls  
find  
locate  
shell wildcards
```

**Extract the Data**

```
for, while  
find -exec  
xargs
```

**Filter the Data**

```
cat  
extract  
pdftotext, ...  
wc
```

**Modify the Data**

```
grep  
head, tail  
sort, uniq
```

Print all the words in the English dictionary that contain all five vowels



You have a folder with all the pictures from your recent vacation. Count how many of them were taken with the flash



Find all the broken links in your system

Print the 100 most frequent words contained in a .txt document



Print the 100 most frequent words contained in a .txt document, excluding the stop words (listed in a separate file)

# Your turn

```
1.find ~ -atime 100 | grep -v photo
```

# Your turn

```
1.find ~ -atime 100 | grep -v photo
```

```
 2.find . -iname "*.txt" -exec egrep '.{81}' {} \;
```

# Your turn

1. `find ~ -atime 100 | grep -v photo`

2. `find . -iname "*.txt" -exec egrep '.{81}' {} \;`

3. `ls -l | egrep -o "^[^ ]+" | egrep 'x$'`

# Your turn

1. `find ~ -atime 100 | grep -v photo`
2. `find . -iname "*.txt" -exec egrep '.{81}' {} \;`
3. `ls -l | egrep -o "^[^ ]+" | egrep 'x$'`
4. 

```
for f in original/*; do
    echo $f | egrep -o "[^/]+$" >> /tmp/or; done

for f in data/*; do
    echo $f | egrep -o "[^/]+$" >> /tmp/cur; done

sort /tmp/cur /tmp/cur /tmp/or | uniq -u
```

# Uniforming the End of Line

- Unix and Unix-like systems terminate lines with a **line break** character (`\n` or `0x0A`)
- DOS and Windows terminate line with a sequence of **two** characters: **carriage return** (`\r` or `0x0D`) and line break
- This little difference makes text files difficult to exchange

```
In the Beginning was the Command Line^M
^M
by Neal Stephenson^M
^M
^M
    About twenty years ago Jobs and Wozniak, the founders of Apple, came up
```

- `dos2unix` and `unix2dos` commands can be used to translate the end of the line characters from one world to the other



# Format

```
fmt [options] file
```

- Re-format each paragraph of a text to better fix a certain line length
  - It can break and join lines, but it preserves words
- Options
  - `-w width` – reformat each paragraph to no more than `width` characters
  - `-c` – preserve the indentation of the first two lines (the rest of the paragraph is then aligned with the second line)
  - `-s` – split long lines but don't join short ones (useful to prevent joining lines of code)

# Translate

```
tr [options] set1 [set2]
```

- Translate the characters specified in `set1` to the corresponding characters in `set2`
  - Sets of characters may be abbreviated by using ranges (`[a-z]`)
  - `set2` is automatically extended to the length of `set1` by repeating its last character as necessary
- Options:
  - `-s` – replace each instance of a sequence of repeated characters listed in `set1` with a single instance of the character specified in `set2`
  - `-c` – complement the set of characters (substitute all the character that are NOT specified in the set)
  - `-d` – delete every instance of the characters instead of replacing it (with this option you must specify only one set of characters)

# Examples

```
balzarot> echo "12345" | tr "2-4" "xyz"  
1xyz5
```

```
balzarot> echo "12345" | tr "2-4" "x"  
1xxx5
```

```
balzarot> echo "12345" | tr -s "2-4" "x"  
1x5
```

```
balzarot> echo "12345" | tr -d "2-4"  
15
```

# Useful commands:

# Convert to single space


```
balzarot> tr -s "[[:space:]]" " "
```

# Convert to lowercase

```
balzarot> tr "A-Z" "a-z"
```

# The Stream Editor


```
sed [options] "location action" [files]
```

- `sed` is one of the first Unix commands built for command line processing of text files
- It is a complex tool, but still not a **full fledged** programming language 
  - No variables
  - No branching (only labels and GOTO)
  - But it is still Turing-complete!
- `sed` performs a certain action to the specified locations
  - It is line oriented, so everything is repeated for each line in the input

# SED

- Default behavior:
  - Read next line
  - Check if it matches one of the specified **locations**.
  - If yes, apply the corresponding **action**
  - Print the line to standard output
- Options:
  - n** don't print the lines if not specifically instructed to
  - e** combine together multiple expressions (location+action)  
example: `-e "2,3p" -e "7,8p"`
  - i** edit the file in place
  - r** use extended regex

# Locations

- Locations specify to which lines an action must be applied
  - If the location is not specified, the action is executed for **each line**
  - If a **single value** is specified, the action is executed only for the line matching that value
  - If a **range** is specified (two locations separated by comma), the action is executed from the line matching the first location to the line matching the second location
    - The two location cannot match the same line
    - Ranges always  span at least two lines (unless the file terminates after the first one)

# Locations

- Locations
  - N line number N
  - \$ last line
  - /regex/ line matching the regular expression
  - +N next N lines (when used as a second value in a range)
- A location can be negated by appending an exclamation mark after it
  - 7, 9! – every line except 7, 8 and 9

# Actions

- `p` – print the line (usually used in conjunction with the `-n` option)
- `d` – delete the line
- `q` – quit without processing the remaining lines
- `a text` – append the text after the matching line
- `i text` – insert the text before the matching line
- `c text` – change the line with text



# The power of S

`s/regex1/replacement/flag`

- substitute whatever matches “regex1” with “replacement”
  - If no flags are specified, it only substitutes one occurrence per line
  - By adding a number N, it will only do N substitutions per line
  - By adding the g flag, all the occurrences will be substituted
  - The / characters can be uniformly replaced by any other single character (if the character also appears in the regex, it has to be escaped)
- Groups and back references
  - In the replacement, the & character can be used to indicate the text matched by the regex
  - In the regex, part of the pattern can be grouped between parenthesis ()
  - In the replacement, \1,\2,.. \9 are substituted with the corresponding groups in the regex

```
sed -r '10,$ s-<a href="(.*)">(.*</a>- \2: \1-g'
```

```
sed -r '10,$ s-<a href=" (.*)" > (.*) </a>-\2: \1-g'
```

Option  
(use extended regex)

Location  
(from line 10 to the end of the file)

Action  
(substitute)

RegEx to  
substitute

Replacement

Flag  
(global substitute)


# Examples

Delete the second line	<code>sed '2d'</code>
Print the 10 <sup>th</sup> line	<code>sed -n '10p'</code>
Comment out the 12 <sup>th</sup> lines	<code>sed '12 s,^,/,/, '</code>
Delete all the empty lines	<code>sed '/^\$/d'</code>
Print the lines longer than 80 characters	<code>sed -n '/.{81}/p'</code>
Add “Comment” after the 7 <sup>th</sup> line	<code>sed '7a Comment'</code>
Delete leading whitespaces from each line	<code>sed 's/^[ \t]*//'</code>
Delete everything after the first blank line	<code>sed '/^\$/q'</code>


# AWK

- **AWK is a programming language designed to process text files**
  - First created at Bell Labs in the 1970s
  - It is an acronym of its authors' names but it is usually pronounced like the bird (i.e., auk, not hawk)
  - Beside the shell itself, AWK is the only other scripting language available in a standard Unix environment
  - It is one of the oldest tools to perform **mathematical operations** on files of numerical data
- awk is somehow similar to sed: for each record that matches a pattern, it executes the corresponding action

# Records and Fields

- AWK treats the input as a sequence of **records**  
(by default each line is a record)
- Each record is then split into a sequence of **fields**  
(by default each word is a field)
- Fields and records are specified by special variables:
  - \$0 refers to the current record (line)
  - \$1-\$n are the single fields of the current record  
(any expression can be used instead of numbers)
  - NF is the number of fields in the current record
  -  NR is the record number
  - FS and RS are the field and record separator  
(they can be a character, a word, or a regular expression)
  - OFS and ORS are the separators used in the program output

# AWK Programs

- Unless many other languages, programs in AWK are not procedural but **data-driven** 
- A program in AWK is composed by a sequence of rules
  - Each rule specifies one pattern to search for and one action to perform when the pattern is found

```
record-pattern1 {action1}  
record-pattern2 {action2}
```

...

...

- In a rule, either the pattern or the action can be omitted, but not both
  - If the pattern is omitted, the action is performed for every record
  - If the action is omitted, the default action is to print all records that match the pattern

# Patterns

- `BEGIN, END` – special values used to specify actions to be executed at the beginning or at the end of the program (before and after the input is processed)
  - Useful to initialize variables (such as the field separator) and to print final results
- `pattern1, pattern2` – every line from the one that matches `pattern1` to the one that matches `pattern2`
- `/regex/` – each record that matches the regular expression
- `Condition` – each record for which the condition is true
  - `NR == 10` – at the 10<sup>th</sup> record of the input
  - `/regex1/ && /regex2/` – true if both are true
  - `$1 ~ /regex/` – true if the first field match the regex



# Actions

- Each action can be a single instruction or a sequence of instructions separated by semicolon
- Control statements:
  - `if (condition) {body} else {body2}`
  - `for (initialization; condition; increment){ body }`
  - `while (condition) {body}`
  - `break` and `continue`
- AWK supports floating point arithmetic and **automatically converts** strings to numbers
- AWK supports most of the mathematical operators that you have in C (`+`, `-`, `*`, `/`, `%`, `^`, `++`, `--`, `+=` ...)

# Useful Functions

- String manipulation:
  - `length` - return the length of a string (without parameters returns the length of the current record)
  - `tolower`, `toupper` – convert strings to lower/upper case
  - `substr(string, start, nchar)` – extract a substring
  - `gsub(regex, replacement, string)` – regex replacement
- Output
  - `printf` – like in C
  - `print item1 item2 item3` – print the concatenation of the three items
  - `print item1, item2, item3` – print the three items separated by the output field separator (OFS)

# Variables

- Modern awk implementations do not require variables to be initialized
  - Integer variables automatically initialized to 0, strings to ""
- Arrays are automatically created and resized
  - Arrays are "associative", meaning that the index can be any string:
    - `array["txt"] = value`
    - `array[50]` is equivalent to `array["50"]`
  - `index in array` – tests if the array contains the specified index
  - Array can be scanned using a special for loop
    - `for (variable in array) { body }`

# Examples

Change the first field to ">"	<code>awk '{ \$1="&gt;"; print }'</code>
Print every line with more than 4 fields	<code>awk 'NF &gt; 4'</code>
Right align all text on a 79-column width	<code>awk '{ printf "%79s\n", \$0 }'</code>
Print the even-numbered lines	<code>awk 'NR % 2 == 0'</code>
Swap the first two fields	<code>awk '{ t=\$1; \$1=\$2; \$2=t; print }'</code>
Add a new field at the end of each line	<code>awk '{ \$(NF+1)="new"; print }'</code>
Print the sum of each field	<code>awk '{ for (i=1; i&lt;=NF; i=i+1) s+=\$i; print s }'</code>

# Working on Multiline Records

Jimmy the Weasel  
100 Pleasant Drive  
San Francisco, CA 12345

Big Tony  
200 Incognito Ave.  
Suburbia, WA 67890

```
awk 'BEGIN {RS=""; FS="\n", OFS=","}
      {print $1,$2,$3}'
```

Jimmy the Weasel, 100 Pleasant Drive, San Francisco, CA 12345  
Big Tony, 200 Incognito Ave., Suburbia, WA 67890

# Piping in AWK

- The output of the print and printf instructions can be redirected or piped to other tools

```
print $0 > "filename"
```

```
print $1 | "command"
```

- **Important!** AWK opens a file or a pipe only if that particular file or command has not already been written to by the program or if it has been closed since it was last written to
  - `awk '{print $1 | "sort"}'` - sort is executed once, and the outputs of all the print instructions are piped to it
  - `awk {print $1 > "file.dat"}` - the file is open (the previous content is lost) but then each line is added to it
  - `awk {print $1 > "file.dat"; close("file.dat")}`  
the file will contains only the output of the last print

# Internet from the Command Line

- Many network- and web-related command lines tools
  - Text-based web browsers (`w3m`, `lynx`, `links`)
  - Non-interactive downloaders (`wget`, `curl`)
  - General input/output through sockets (`netcat`)
  - DNS query (`host`, `dig`)
  - Pinging, scanning, trace routing (`ping`, `arping`, `nmap`, `traceroute`, `hping3`)
  - Remote shell, secure copy (`ssh`, `scp`)
  - ....

# wget

- wget download files from the web using either HTTP, HTTPS, or the FTP protocol

**wget [options] url**

- It can retrieve a single document or recursively download everything that is referred to from a remote resource
- Parameters
  - `-m` – mirror the entire web site
  - `-r` – recursively download all the linked documents
  - `-l n` – specify the max recursion depth to n
  - `-nd` – don't replicate the directory structure  
(saves all the file in the same directory)
  - `-k` – fix all the the links in the downloaded pages to make them suitable for offline browsing




# wget

- `-c` – continue the download of a partially-downloaded document
  - `--spider` – don't download the pages, just check that they are there
  - `-w sec` – wait `sec` seconds between each download
  - `-U agentstring` – send `agentstring` as user agent
  - `-A pattern` – download only files matching the pattern
  - `-R pattern` – do not download files matching the pattern
  - `--limit-rate nk` – limit the speed to `n` KB/s
- 
- **Example: download all the slides for the first lecture:**

```
wget -r -l1 -nd -A"1_*.pdf"  
http://www.iseclab.org/softdev/material.html
```

# CURL

- Very similar to wget but support more protocols:
  - FTP, FTPS, HTTP, HTTPS, SCP, SFTP, TFTP, TELNET, DICT, FILE, LDAP and LDAPS
- Curl fetches just the URLs that the user specifies
  - It does not contain any recursive downloading capability
  - It does not  parse the HTML
- It supports file upload in HTTP and FTP
- It has a pretty complete cookie management

# CURL



- `-s` – silent (suppress the progress meter)
- `-d data` – send the data in a POST request
- `-A useragent` – specify the given useragent
- `-F "field=@filename"` – upload a file
- `-I` – fetch only the HTTP headers
- `-c file` – write the cookies to file (in netscape format)
- `-b file` – load the cookies from file
- `-u username:password` – specify the authentication credentials

- **Example:** submit a POST form, pretending to be a Mozilla browser

```
curl -s -A "Mozilla/5.0 (compatible; MSIE 7.01;
Windows NT 5.0)" -d 'rid=value&submit=SUBMIT'
```

# URL De-Shortening with CURL

```
curl -sI http://tinyurl.com/m3q2xt
```



```
HTTP/1.1 301 Moved Permanently
Location: http://en.wikipedia.org/wiki/URL_shortening
X-tiny: cache 0.00096607208252
Content-type: text/html
Date: Wed, 20 Oct 2010 09:09:13 GMT
Server: TinyURL/1.6
```



```
sed -n 's/Location: //p'
```

# The TCP/IP Swiss Army Knife

- netcat (nc) is a tool to read and write data across TCP or UDP network connections
- It can be used as a generic network client:
  - `nc host port`
  - The standard input is redirected to the remote host
  - The data received from the network is sent to the standard output
- It can be used as a generic network server:
  - `nc -l -p port`
  - It listens for inbound connections to port `port` and then forwards the data as in client mode
  - It manages only one connection and then terminates

# Copying Files with NetCat

- On the sender side:

```
tar czf - dir | nc -q 10 -l 3333
```

- On the receiver side:

```
nc -w 10 remotehost 3333 | tar xzv
```

- The same can be done by having the receiver that listens and the sender that opens the connection