Introduction aux Systèmes d'Exploitation

Unit 4: Regex and useful commands

François Taïani

References

- **Regular Expressions In grep**
  - ➔ http://www.cyberciti.biz/faq/grep-regular-expressions/
- **Advanced Bash-Scripting Guide: Chapter 18.**
  - ➔ http://tldp.org/LDP/abs/html/regexp.html
- **man -s7 regex / man re_format (OS X)**
- **Getting started with SSH**
  - ➔ http://kimmo.suominen.com/docs/ssh/

Unit Outline

- **Some useful commands**
  - ➔ which, find, uniq, sort, grep
- **Regular expressions**
  - ➔ Variants, operators, boundaries
- **ssh**

which and find

- **which**
  - ➔ print location of executable

- **find <directory> -name “<somename>”**
  - ➔ look recursively for a file
  - ➔ <somename> can contain wildcards
  - ➔ e.g. **find . -name "*.txt"**
  - ➔ many other conditions: **man find**

uniq & sort

- **uniq**
  - ➔ Remove contiguous duplicate lines
  - ➔ with -c print a count how many in original file / stdin
- **sort**
  - ➔ Sort lines of files, using dictionary order by default
  - ➔ -r : reverse order
  - ➔ -n : numerical order
  - ➔ -k<x> : use field number 1 for sorting
- Both have many other options → man is your friend

grep : Searching in files

- **grep** (“global regular expression print”)
  - ➔ typical use “grep <somestring> <somefile(s)>”
  - ➔ print all lines where <somestring> occurs
- Numerous options (see ‘man grep’
  - ➔ -i ignore case (POSIX compliant)
  - ➔ -c count occurrences (POSIX compliant)
  - ➔ -v select lines where string does not occur (POSIX)
  - ➔ --color print matches in color (GNU)
  - ➔ -o print only matches strings
- Goes beyond mere string
  - ➔ can work with expression, known as regular expressions

Example

```
debianfrancoist ftaiani [SHELL_PROGRAMMING] $ grep --color creature
the creature came towards me!
the creature came towards me!
I let out a shriek: Ah!!!! The creature is on me.
I let out a shriek: Ah!!!! The creature is on me.
```

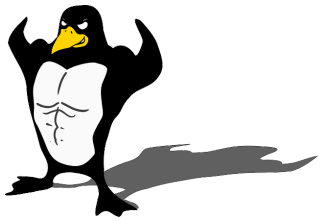
- If pg84.txt contains the text of Frankenstein
  - ➔ How would you count the number of lines containing the word 'creature', independently of case?

Regular expressions

- Same idea as globbing, but much more powerful
  - ➔ See shell globbing mechanism in Unit 3
- Not limited to shell command line
- Used in many unix tools and commands
  - ➔ grep
  - ➔ sed
  - ➔ awk

- But also key in scripting and web languages
  - ➔ perl, PHP, ruby
  - ➔ (and bash)

Regex variants



- Regular expressions come in several flavours
  - ➔ old regex ("basic regex")
  - ➔ modern regex ("extended regex")
  - ➔ many flavours depending on scripting language
- Some tools can do both, others only one
  - ➔ e.g. grep using -E or sed with -r for modern regex
  - ➔ ruby only follows the new one (or mostly)
- In the following: modern Regex

Regex operators

- **Matching characters**
  - . → any character
  - [ ] → set of characters, can use [x-z] for range
  - [^] → not these characters (can use [^x-z] as well)
  - use \ to "escape" ., [, and] and match actual character
- **Examples**
  - grep -E "[aeiou]": lines containing at least one vowel
  - grep -E "[^aeiou]": lines containing at least a non-vowel
  - grep -E "." : lines containing at least one character
  - grep -E "....." : lines containing at least 5 characters
- (don't forget quotes to prevent globbling!)

Regex operators (cont.)

- **Repeating sequences**
  - ( ) : group a pattern together
  - (X|Y) : X ou Y
  - X*: zero, one, or more times X
  - X+, one or more times X
- **Example**
  - a*b: matches b, ab, aab, aaab, etc.
  - (ac)*b: matches b, acb, acacb, etc.
  - a{1,2}b+: matches ab, aab, abb, but not aaab
- Quiz
  - will grep -E "piz+" match the line "pizzza" ?

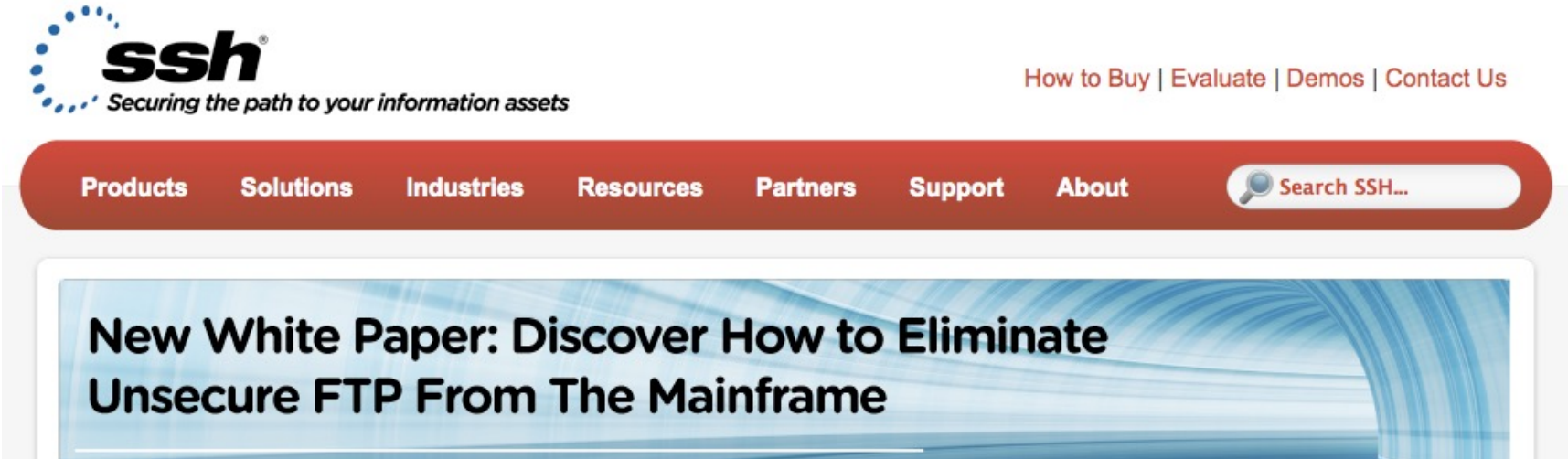
Regex in scripting language

- Often separated by / /
- Uses =~ or ~ to match a string against a regex

- For instance: awk
  - ➔ awk '$2~/^S/' student-list.txt
  - ➔ print all students whose family name starts by S

SSH

- 1995, Tatu Ylönen, Helsinki University of Technology
  - ➔ to prevent plain-text password sniffing with old rsh
- Ylönen's implementation
  - ➔ first Open Source, then increasingly proprietary
  - ➔ prompted alternative implementations. e.g. OpenSSH
- Ylönen's company still around and active
  - ➔ http://www.ssh.com/



SSH

- SSH = secure shell
  - ➔ network protocol (i.e. standard, with reference implem)
  - ➔ for secure remote connection (encrypted)
  - ➔ by default connection textual (remote shell)

- Several implementations
  - ➔ OpenSSH (http://www.openssh.com/)
  - ➔ Putty (client-only for windows, http://www.chiark.greenend.org.uk/~sgtatham/putty/)

- Two pieces to SSH
  - ➔ ssh client (to make connections)
  - ➔ ssh server (aka daemon): sshd to receive them

SSH

- Goal: Two-way authentication + encryption
  - ➔ each server a pair of cryptographic keys: their "ID"
  - ➔ SSH will complain if this server key change
  - ➔ more secure techniques possible (certificates, DNS)
- user can authenticate via various methods
  - ➔ default username / password: not the best
  - ➔ cryptographic keys (ssh-keygen): much better (but important to protect private key with password)
  - ➔ connection to higher systems (PAM, Kerberos)
- Tagline: very flexible and adaptable

SSH

- Basic working
  - ➔ ssh <username>@<machine>
  - ➔ or ssh <machine> if username is same everywhere
- Examples

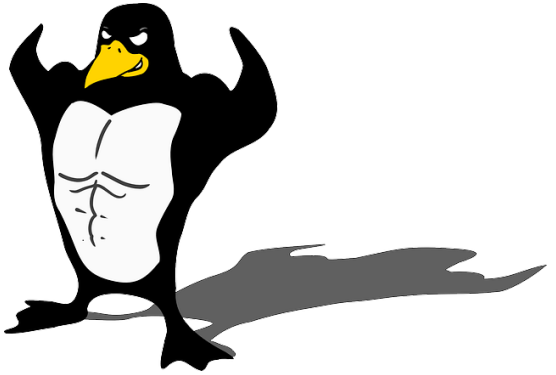
```
borrowdale ftaiani [SCRIPTING] $ ssh welcome1.istic.univ-rennes1.fr
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-81-generic x86_64)
[..]
Last login: Wed Sep 15 21:48:25 2021 from 148.60.9.1
$ uname -a
Linux welcome1 5.4.0-81-generic #91-Ubuntu SMP Thu Jul 15 19:09:17 UTC
2021 x86_64 x86_64 x86_64 GNU/Linux
$ exit
Connection to welcome1.istic.univ-rennes1.fr closed.
borrowdale ftaiani [SCRIPTING] $
```

- Note: VPN needed (see https://istic.univ-rennes1.fr/services#p-246)

SSH – more advanced

- Executing remote command:
  - ➔ ssh ftaiani@welcome1.istic.univ-rennes1.fr uname –a

- Forwarding ports / services
  - ➔ ssh -vvv -L 4000:hackerspaces.org:80 ftaiani@welcome1.istic.univ-rennes1.fr



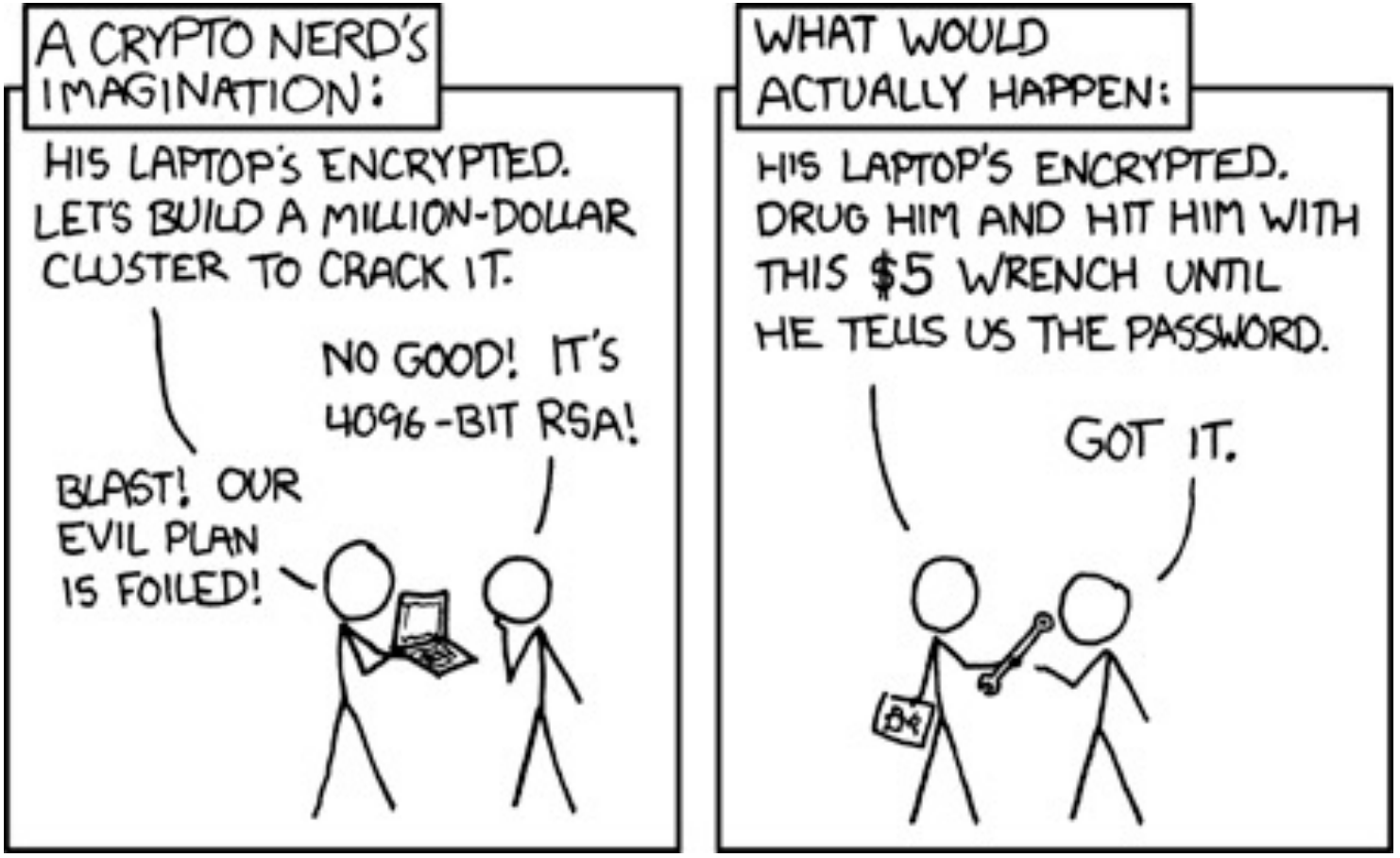
Practicing All You've Learnt

- If you have access to a Unix machine
  - ➔ Use the console!
  - ➔ Play with grep
  - ➔ Try to use each of the capabilities we have seen
- If you don't have access to a Unix machine
  - ➔ If fact you do ! Use PuTTY / `ssh` to connect to `welcome1.istic.univ-rennes1.fr`

Homework: How would you check which Unix OS welcome1 uses ?

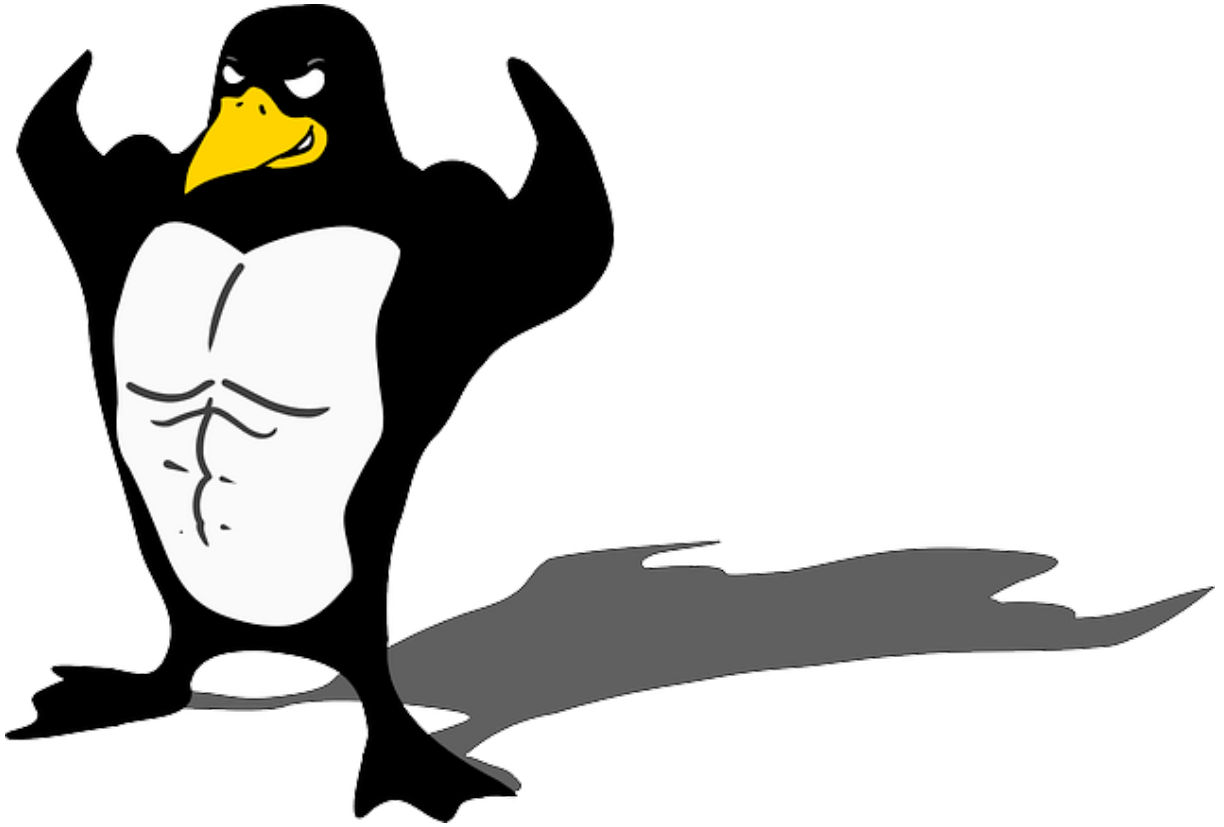
Physical Protection …

- from http://xkcd.com/538/

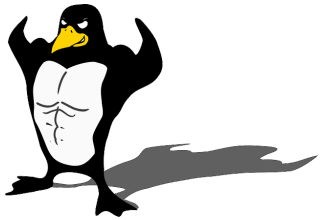


Bonus Material

- Not exam material

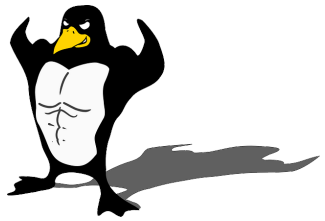


Regex boundaries



- Lines and words
  - ^: beginning of line
  - $: end of line
  - \< and \> : beginning, end of word (GNU regex)
- Example
  - ^free: matches lines starting with free
  - \<free\>: matches free, but not freedom
  - free\>: match words ending with free

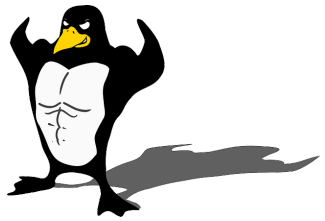
sed



- **S**tream **Ed**itor: **sed 'sed program' file**
- Most common command: s (substitution)

```
$ sed 's/day/night/'
It was a bright day.
It was a bright night.
What a nice day I said. On such a day, we should go and
have some fun.
What a nice night I said. On such a day, we should go and
have some fun.
```

sed (continued)

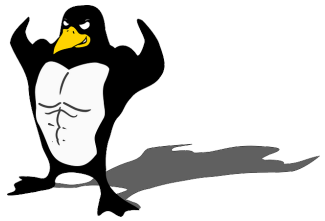


- Slightly more advanced substitution
  - ➔ g modifier (for global)
  - ➔ special character & in replacement (matched pattern) ...
- Example

```
$ sed -E 's/day|night/\(&\)/g'
It was a bright day.
It was a bright (day).
Such a nice day! I said. Let's wait for the night...
Such a nice (day)! I said. Let's wait for the (night)...
```

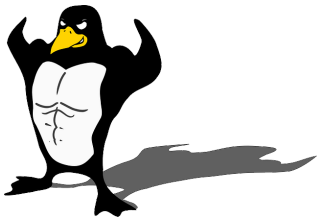
  - ➔ Note the -E option (modern regex) +the escaped brackets
- A lot more: e.g.
  - ➔ http://www.grymoire.com/Unix/Sed.html

More on awk



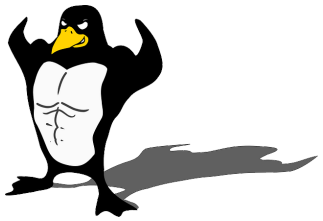
- Invented by Aho, Weinberger, & Kernighan at Bell Labs
  - ➔ the same place as Unix (not a coincidence)
  - ➔ (BTW Bell Labs also birthing place of C, C++)
  - ➔ in the 70's
- Line editor a bit like grep, but
  - ➔ works with fields: $0 = whole line, $1 first field, $2 second..
  - ➔ by default: fields assumed separated by blanks
  - ➔ full-fledge programming language
  - ➔ many implementation (gawk, mawk, ..)
- General form
  - ➔ **awk 'condition { commands }'** (both optional)

Refining our example



- awk '$2~/^S/ {print $1}' student-list.txt
  - ➔ prints the first name of students whose family name start with an S
- **Exercise**
  - ➔
```
awk '{count[substr($2,1,1)]++}
END{
   for(i in count)
     print i," : ", count[i]
}' student-list.txt
```
  - ➔ what does this do?

Exercise



- sudo head /var/log/syslog

```
Apr 22 01:02:56 debianfrancoist rsyslogd: [origin software="rsyslogd"
swVersion="4.6.4" x-pid="1011" x-info="http://www.rsyslog.com"] rsyslogd was HUPed,
type 'lightweight'.
Apr 22 01:02:56 debianfrancoist rsyslogd: [origin software="rsyslogd"
swVersion="4.6.4" x-pid="1011" x-info="http://www.rsyslog.com"] rsyslogd was HUPed,
type 'lightweight'.
Apr 22 01:03:00 debianfrancoist anacron[1056]: Job `cron.daily' terminated
Apr 22 01:03:00 debianfrancoist anacron[1056]: Job `cron.weekly' started
Apr 22 01:03:00 debianfrancoist anacron[2371]: Updated timestamp for job
`cron.weekly' to 2014-04-22
```

- Exercise
  - ➔ How would you extract the list of the programs that printed the messages
  - ➔ Hint : you can use awk & grep
    (more advanced version possible with awk alone)