Introducing Bash : Ding

Pierre Reinbold
pre@info.ucl.ac.be

Original Ding package by Stéphane Delcroix (2003)

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Plan

1. Introduction (quick)
2. Scripting a bit
3. Standard flows
4. Stream & regexp
5. Loops
6. Functions

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

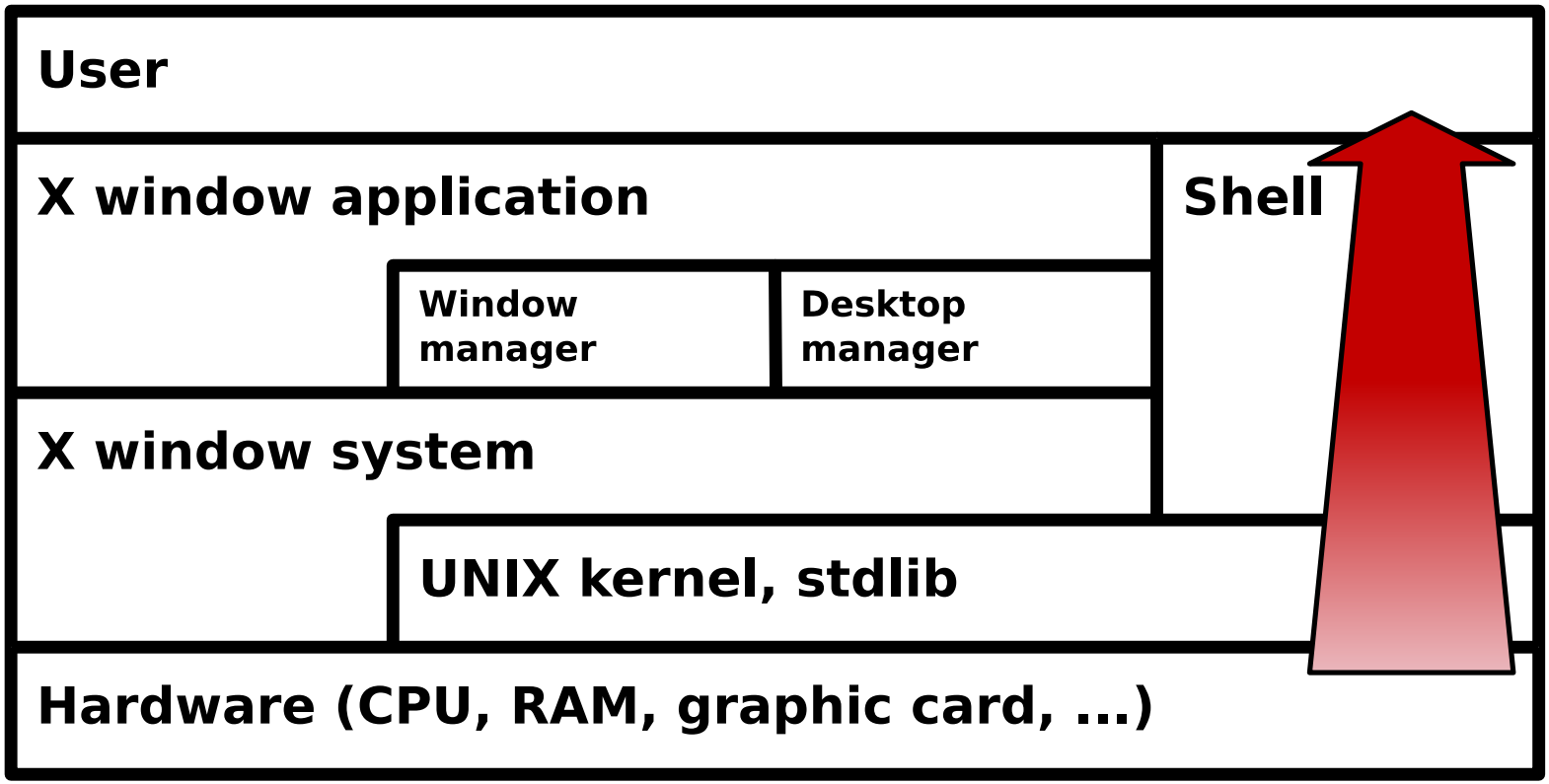
Introduction and objectives

- Tutorial introduction to (some features of) the **Bash** shell
- Ding : Ding Is Not Google
- Learning by doing (sort of ☺)
- Some prerequisites :
  - basic Unix notions
  - basic SQL (MySQL)
  - basic HTML (HTTP)

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Unix, Shell and Bourne Shell

- Unix Tetris ®



→ **The most efficient way to use the machine ressources**

pass through the shell ! ☺

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Unix, Shell and Bourne Shell

- 1970's : Unix == stable OS Kernel + Standard C library
  → was (just) a programming thing !
- **Stephen Bourne** quickly comes with **sh** :
  - a shell around the kernel
  - a small, on-the-fly compiler
  - one command at a time
  - → **interpreter**
  - Bourne Shell : **sh**
- 1987 : **Brian Fox** created **Bash** for the GNU project
  - syntax == superset of sh
  - many improvements and ideas from Ksh and Csh
  - command completion
  - Bourne Again Shell : **Bash** (born again shell)
  - maintained by **Chet Ramey** since 1990

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Ding Is Not Google

- Ding is a simple (dumb?) bash web indexer
- Features :
  - manage links and keywords
  - link URL and keywords, with word occurence
  - manage last time visit for links
  - allow searching the DB with keywords for URLs
- Some dependencies :
  - MySQL : DB backend
  - Wget : very efficient web crawler
  - standard Unix tools : grep, sed, and many more

→ OpenSource softwares, available on all Linux distro.

The most simple Ding DB

Using MySQL :

```
CREATE TABLE ‘ding_mot‘ (
  ‘mot‘ varchar(40) NOT NULL default ’’,
  PRIMARY KEY  (‘mot‘)
) TYPE=MyISAM;
# --------------------------------------------------------
CREATE TABLE ‘ding_occurence‘ (
  ‘url‘ varchar(255) NOT NULL default ’’,
  ‘mot‘ varchar(40) NOT NULL default ’’,
  ‘nbr‘ int(4) NOT NULL default ’1’,
  PRIMARY KEY  (‘url‘,‘mot‘)
) TYPE=MyISAM;
# --------------------------------------------------------
CREATE TABLE ‘ding_url‘ (
  ‘url‘ varchar(255) NOT NULL default ’’,
  ‘last_update‘ date default NULL,
  PRIMARY KEY  (‘url‘)
) TYPE=MyISAM;
```

UCL - INGI
Département d’ingénérie informatique
www.info.ucl.ac.be

Writing a bash script

- a bash script is just a file with bash commands
- each command is executed in turn
- bash has many programming features :
  - control flow (if, else, ...)
  - loops (for, while, until)
  - functions
  - and more !

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Writing a bash script (2)

- Saying "Hello World !" with bash

```
$ cat helloworld.sh
#!/bin/bash
echo "Hello world !"
$ chmod +x helloworld.sh
$ ./helloworld.sh
Hello world !
$
```

- The **shebang**, SHarp (#) - BANG(!), followed by a path, is use to specify the command interpreter in Unix (sh, bash, php, perl, python, . . .) see `http://en.wikipedia.org/wiki/Shebang_(Unix)`
- "#" is also the comment marker in bash
- do not forget to make it executable !

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Grab

- **Grab** is our main script to index URLs
- How it should work :
  - take en URL from command line
  - get this from the web
  - isolate words
  - update the DB
- a starting point :

```
#!/bin/bash
# Grab (c) 2007 me (free software blabla.)

echo "This is Grab, the Ding indexer !"
```

Grab : arg management

- we must capture the command line argument of **Grab**

```
$ ./grab http://www.info.ucl.ac.be
```

```
#!/bin/bash
# Grab (c) 2007 me (free software blabla.)

echo "This is Grab, the Ding indexer !"

# Simple args management : check for $1
if [ $# -eq 1 ]; then
    URL="$1"
else
    echo "usage : $0 <URL>"
    exit 1
fi

echo "Grabbing $URL"
echo
```


UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Grab : arg management

- what it does :

```
$ ./grab-v1.sh
This is Grab, the Ding indexer !
usage : ./grab-v1.sh <URL>
$
$ ./grab-v1.sh http://www.info.ucl.ac.be
This is Grab, the Ding indexer !
Grabbing http://www.info.ucl.ac.be

$
```

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Arguments and variables

- args are stored in `$1` to `$9` (use `shift` for more)
  - `$0` script name
  - $# args number
  - $@ args list (see also `$*`)
  - $$ script PID
- variables are dynamically typed
  - no declaration, just assignation
  - get the value with "$"

```
URL="$1"
echo "Grabbing $URL"
```

  - try black magic with `${VARNAME op }` Ex. substitution :

```
$ STRING="toto tutu b212"
$ echo "[${STRING// /, }]"
[toto, tutu, b212]
```

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Arguments and variables

- Read variables from users too !

```
echo -n "What's your name ? "
read name
echo "Hello $name !"
echo -n "What's your quest ? "
read quest
echo "Really ?"
echo -n "What's your favorite color ? "
read color
echo "Right. Off you go."
```

Gives :

```
$ ./readit.sh
What's your name ? Lancelot of Camelot
Hello Lancelot of Camelot !
What's your quest ? To find the Holy Grail
Really ?
What's your favorite color ? Blue
Right. Off you go.
```

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Using variables

- some usefull variables (or not)
  - The Environment : a set of available variables
    - `$PATH`
    - `$HOME`
    - `$USER`
    - `...`
  - other vars, set by Bash
    - `$UID` : the real UserID of the script
    - `$EUID` : the current UserID (! ex. `su`)
    - `$SECONDS` : time elapsed since the beginning of the script (in seconds ☺)
    - `$RANDOM` : random integer between 0 et 32767
    - `$IFS` : separator (default : blanks or tabs)

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Testing, truth value and return code

```
if test $# -eq 1; then
    URL="$1"
else
    echo "usage : $0 <URL>"
    exit 1
fi
```

```
if [ $# -eq 1 ]; then
    URL="$1"
else
    echo "usage : $0 <URL>"
    exit 1
fi
```

- Testing operators :

Integer	-eq	-ne	-lt	-gt	-le	-ge
Strings	==	!=	<	>	<=	>=

- Boolean operators (cf. **C**)
  - "&&" : AND, "||" : OR, "!" : NOT
  - Lazy example :

    ```
    ! grep newuser /etc/passwd && useradd newuser
    ```

    Add a user only if it is not found in **/etc/passwd**.

Testing, truth value and return code

- Testing operators for files (some) :

```
if test <file test> <file name> ; then
    <do something>
fi
```

- Some examples for `file test`:
  - Exists : `-e`
  - Directory : `-d`
  - Standard file : `-f`
  - Readable : `-r`
  - Writable : `-w`
  - Exec : `-x`
- Example : backup some file

```
SUFFIX=".bak"
FICHIER=$1
if test -f $FICHIER ; then
    cp $FICHIER $FICHIER$SUFFIX
fi
```


UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Testing, truth value and return code

- the return code **is** the truth value !
  - what is tested ? **The return code of the `test` command !**
    (`[...]` just an alias)
  - not testing with **`test`** ?

```
$ if cat /toto; then echo TRUE; else echo FALSE; fi
cat: /toto: No such file or directory
FALSE
```

- arg of **exit** (or return code of last command)

```
echo "usage : $0 <URL>"
exit 1
```

  - the return code == end status of a command
    - 0 : success
    - >0 : failure

⇒ in bash : **0 is true !**

- stored in $?

```
$ cat /toto
cat: /toto: No such file or directory
$ echo $?
1
```

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Grab : get on with it !

- Where we are :

```
#!/bin/bash
# Grab (c) 2007 me (free software blabla.)

echo "This is Grab, the Ding indexer !"

# Simple args management : check for $1
if [ $# -eq 1 ]; then
    URL="$1"
else
    echo "usage : $0 <URL>"
    exit 1
fi

echo "Grabbing $URL"
echo
```

- Now getting the data from the web !

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Grab : get on with it !

```
echo "Grabbing $URL"
echo

TMPURL="/tmp/tmpurl.$$.$RANDOM"
echo "copy page into temporary file $TMPURL"
rm -f $TMPURL
wget $URL -O $TMPURL
```

- Using **wget** to retreive URL (`man wget`)
- Good temporary filename provided by some ramdomness : `TMPURL="/tmp/tmpurl.$$.$RANDOM"`
- Now we have the source code of the URL into our temp file

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Grab : words management

We have to find words into the source code and to update the DB accordingly

- strip the html tags
- isolate words
- count occurences
- update the DB

⇒ some fun to come !

Let's begin with words finding, assuming english ascii text...

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Grab : words management

To get the sorted list of ascii words (lower case) from $TMPURL :

```
cat $TMPURL | \
    tr -d "\r\n" | \
    sed -e "s/<[^<>]*>//g" | \
    tr "[.;,: ]" "\n" | \
    grep "^[a-zA-Z]\{3,40\}$" | \
    tr "[A-Z]" "[a-z]" | \
    sort | uniq
```

- a single bash line, continued by "\"
- "|" and so-called input-ouptput redirection
- `tr`, `sed` and `grep`, powerfull stream management tools handling **regular expressions**
- `sort` and `uniq`, standard tools to sort and remove duplicate entries within word lists
- encoding for other languages must be handled with care and introduces a bunch of tricky issues ! ☺

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

IO Redirection

- Standard flows (from the C)

`stdin`	:	standard input	⟷	keyboard
`stdout`	:	standard output	⟷	screen
`stderr`	:	standard error	⟷	screen

default

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

IO Redirection

Each flow may be redirected :

`<command> <redirection symbol> <file name>`

- Redirect stdout : ">" or ">>"

  What was printed to the screen is now written into a file

  - `$ ls -alsh /root > out`

    `$ echo 'me:x:505:100::/home/me:/bin/bash' >> /etc/passwd`
  - ">" overwrites the file,
  - ">>" appends stdout to the file (creating it if necessary)
  - Errors are not redirected (stderr still on screen)

- Redirect stderr : "2>" ou "2>>"

  - `$ ls -alsh /root 2> out`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

IO Redirection

- Redirect stdout **and** stderr to overwrite : "&>"

```
$ ls -alsh /root &> out
```

- Redirect stdout **and** stderr to append : ">> 2>&1"

```
$ ls -alsh /root >> out 2>&1
```

using a more general redirect syntax

- Redirect stdin (yes!) : "<"

  What should have been readed from keyboard is readed from a file

```
$ write root pts/5 < message.txt
```

Instead of interactively prompting for message, `write` read it from `message.txt` and send it to root on console `pts/5`

IO Redirection

- **Here Document** (<<) constiture a more elaborate form of stdin redirect

```
#!/bin/bash
HOST="$1"
LANG="fr"

nc $HOST 80 << _EOF_
GET / HTTP/1.0
Host: $HOST
Accept-Language: $LANG

_EOF_
# Note: telnet may be tempting to use but it does read in
# raw mode directly from tty rather than from stdin.
# It can be done however, but it's a bit more tricky ;-)
```

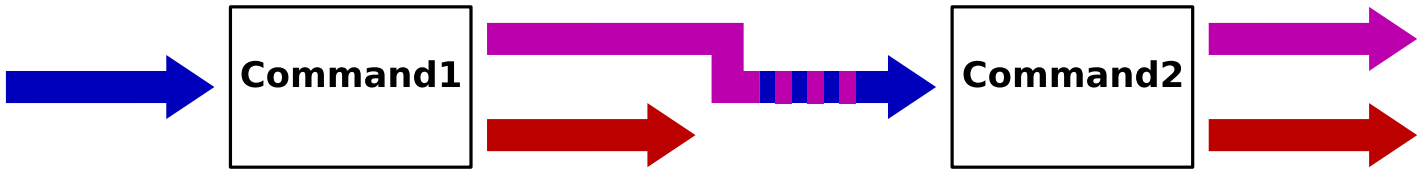
- Get the belgian french page of Google:

```
www-session.sh www.google.be
```

IO Redirection

- Pipe : "|"

  Idea : using stdout of a command as stdin for another command



`<command1> | <command2>`

The output of `command1` is used as input for `command2`.

- `$ cat /etc/passwd | grep prof`
- `$ users | grep student`
- `$ w | grep tty | grep prof`

One of the most powerful Unix feature !

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Grab : words management

Back to **Grab** words management:

```
cat $TMPURL | \
    tr -d "\r\n" | \
    sed -e "s/<[^<>]*>//g" | \
    tr "[.;,: ]" "\n" | \
    grep "^[a-zA-Z]\{3,40\}$" | \
    tr "[A-Z]" "[a-z]" | \
    sort | uniq
```

- 7 pipes to filter and modify on the fly the content of $TMPURL
- the file $TMPURL remains unchanged
- First step: join all lines with **tr** !

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

TR stands for TRanslate

- **tr** is a very simple but usefull character translation tools
- read input text stream from **stdin**, write the result on **stdout**
- works line by line
- some mode:
  - default : translate from a class to another
    - `tr ";" " "` translates `";"` to blank
    - `tr "[.;,: ]" "\n"` translates any char from `".;,:"` or blank to `"\n"`
    - `tr "[A-Z]" "[a-z]"` translates uppercase to lowercase (by matching char ranges)
  - `-d` : delete any char from an arg class

    `tr -d "\r\n"` removes line ends
  - `-s` : squeeze multiple occurrences of the arg class

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Grab : words management

Back to **Grab** words management:

```
cat $TMPURL | \
    tr -d "\r\n" | \
```

At this point, we output $TMPURL into a single long line

```
    sed -e "s/<[^<>]*>//g" | \
    tr "[.;,: ]" "\n" | \
```

After some magic, we cut the line again at the punctuation chars

```
    grep "^[a-zA-Z]\{3,40\}$" | \
    tr "[A-Z]" "[a-z]" | \
```

Some other magic and we translate everything to lowercase

```
    sort | uniq
```

The final sort and delete of duplicate entries

- We need now to understand the magic : **regular expression** used with **sed** and **grep**

Regular expressions (for dummies)

- Regular expressions are used to describe sets of text strings in smart way. **Let's just scratch the surface !**
- Some examples :
  - Any word starting with "a"
  - Any line containing "`totolulub212`"
    → definition of a **pattern**
- any string corresponding to the pattern definition, is said to **match** the pattern
- Widely used with **`(e)grep`** and **`sed`**
  - **`(e)grep`** : search input line by line for matching strings
  - **`sed`** : handle input in various ways, based on patterns, ex. substitutions.

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Regular expressions (2)

- Using it
  - Unix text tools works (mostly) line by line
  - By default patterns are resolved with the longest matching string (greedy match)
  - Some examples with **`grep`** (`-w` to search for complete words):
    - "`egrep -w 't[a-i]e'`" → `tae`, `tbe`, `... tie`
    - "`egrep '^.*Totolulub212.*$'`" → any line containing `Totolulub212`
    - "`egrep -w '(toto|lulu|b212)'`" → `toto`, `lulu` or `b212`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Regular expressions (3)

- canonical wildcard : "`.`" matches with any single char
  - "`a.b`" → `a1b`, `arb`, `...`
- "`*`" allows to match a sequence of 0 or more matches of the preceding char (or sub-expression)
  - "`.*`" → any string (even empty)
  - "`.*toto.*`" → any string containing "`toto`"
  - "`ba*b`" → "bb", "bab", "baab", `...`
- escape char: "\" (take care !)
  - "`\.`" → "`.`"

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Regular expressions (4)

- More general quantifier : "{}"

Quantifier	Repeat
"{x,y}"	from x to y times
"{x,}"	x times or more
"{x}"	exactly x times
"{,y}"	less or equal than y times

- Usefull aliases :
  - "*" == "{0,}"
  - "?" == "{0,1}"
  - "+" == "{1,}"
- Examples :
  - "ab{1,3}c" → abc, abbc and abbbc
  - "ab?c" → ac and abc

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Regular expressions (5)

- Ranges : "`[a-n]`" matches with any char between "a" and "n"
- Class : "`[abc123%@]`" matches with any char into the brackets (to include "`-`", put it in at the end)
- Not Class : "`[^abc123%@]`" matches with any char that are not into the brackets (after "`^`")

  - "`[a-n]*`" → any sequence of chars between a et n
  - "`[aeiou]`" → a, e, i, o or u
  - "`[^aeiou]`" → any char but a, e, i, o or u

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Regular expressions (6)

- Standard character classes :
  - `"[[:alpha:]]"` == `"[a-zA-Z]"`
  - `"[[:upper:]]"` == `"[A-Z]"`
  - `"[[:lower:]]"` == `"[a-z]"`
  - `"[[:digit:]]"` == `"[0-9]"`
  - `"[[:alnum:]]"` == `"[0-9a-zA-Z]"`
  - `"[[:space:]]"` → any blank space, including tabs
- "^" → line start
- "$" → line end (take care !)

Regular expressions (7)

- Alternative "`(  |  )`" :
  - "`(toto|lulu|b212)`" → `toto`, `lulu` or `b212`

- Grouping "`(  )`" :
  - sub-expressions may be grouped within parentheses so that they can be applied a quantifier
  - "`(t[a-r]*b212)*`" → any sequence of matches for "`t[a-r]*b212`"

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Regular expressions (8)

grep: you do not want to live without !

- read stdin (or a file, or files in a directory recursively) line by line
- default output on stdout: lines containing a match for a pattern
  - "`-v`": invert, outputs only lines that does not match
  - "`-A <num_a>`", "`-B <num_b>`": context, outputs also `num_a` lines *after* (resp. `num_b` lines *before*) the matching lines
  - "`-H`" print file name containing matching lines ("`-l`" print only file names)
  - "`-n`" prefix lines with a line number within the input file
  - "`-o`" print only the part of line that matches the arg pattern
  - "`-i`" case insensitive
  - "`-r`" search all files under each directory recursively

```
$ ifconfig | grep -i -A 5 en1 | grep -i ether | \
grep -i -o "\([a-z0-9]\{2\}:\)\{5\}[a-z0-9]\{2\}"
00:16:cb:06:20:24
```


UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Regular expressions (9)

***sed**: geeks love it !*

- Very powerfull text stream tool (real geeks also use **awk**)
- read stdin and output on stdout (default)
- works line by line
- it has many modes but **Grab** uses it only for text **substitution**
  - `sed -e "s/<pattern>/<string>/[g]"` replace the first, (or all if g is there) matching of `pattern` with `string`
  - `$ echo "toto,lulu,b212" | sed -e "s/,/ /g"` gives `"toto lulu b212"`
  - Suppress HTML tags with :

    `$ echo '<H1>Heading 1</H1>' | sed -e "s/<[^<>]*>//g"` gives `"Heading 1"`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Regular expressions (10)

*Escaping chars with **sed** et **grep***

- **sed** and **grep** uses basic regular expressions.
- the following chars must be escaped with "\": "?", "+", "{", "}", "|", "(" and ")"
- **egrep** does not suffer such limitation, it uses extended regular expressions
- Examples:
  - `egrep '(toto|lulu)'`
  - `grep '\(toto\|lulu\)'`
  - `sed -e 's/\(toto\|lulu\)/xxxx/g'`

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Regular expressions (11)

- Example: using **`grep`**, **`sed`** and **`tr`** to get the users belonging to a Unix group by looking into **/etc/group**

  - a line of **/etc/group**:

```
certusers:*:29:root,jabber,postfix,cyrusimap
```

  - and the script:

```
#!/bin/bash

if test -z $1 ; then
    echo "usage: $0 <group name>"
    exit 1
fi

grep $1 /etc/group | \
        sed -e "s/^.*[0-9]://" | \
        tr "," "\n"
```


UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Grab : words management

Merlin's beard! We finally got the magic !

```
cat $TMPURL | \
    tr -d "\r\n" | \
    sed -e "s/<[^<>]*>//g" | \
```

So this is all about **stripping HTML tags** with **`sed`**. Making a single long line with the input allows us to also strip tags expanding on multiple lines.

```
    tr "[.;,: ]" "\n" | \
    grep "^[a-zA-Z]\{3,40\}$" | \
    tr "[A-Z]" "[a-z]" | \
```

We cut our long line into lines containing one word, allowing **`grep`** to select only those containing 3 to 40 chars. We translate then everything to lower case.

```
    sort | uniq
```

The final sort and delete of duplicate entries (note that **`uniq`** works correctly only after **`sort`**)

So simple after all ! ☺

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Grab : words management

Putting words into the DB

```
echo "Managing words in $TMPURL"
WORDS=$(cat $TMPURL | \
    tr -d "\r\n" | \
    sed -e "s/<[^<>]*>//g" | \
    tr "[.;,: ]" "\n" | \
    grep "^[a-zA-Z]\{3,40\}$" | \
    tr "[A-Z]" "[a-z]" | \
    sort | uniq)

MYSQL="mysql -u robot ding"
for WORD in $WORDS; do
    echo "insert into ding_mot VALUES ('$WORD');" | $MYSQL
    # and yes, we should check before insert ! ;-)
done
```

What we need to understand here is:

- storing our precious words into the variable `$WORDS`
- looping over these words and passing them to MySQL

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Command Substitution

- execute a command in a sub-shell and recover its ouput (stdout)
  - **very** handy feature, **very** widely used
  - ‘<commande>‘ is replaced by bash by the result of commande, executed into a sub-shell
  - Cleaner syntax: $(<commande>), permits nesting !
  - Example : produce a fancy log message

```
TMPFILE=/tmp/tmplog.$$.$RANDOM
DATEPREFIX="AT TIME"
DATEPREFIX="$DATEPREFIX "$(date "+%d/%m/%Y-%H:%M:%S")
...
echo "$DATEPREFIX : what a nice log message" >> $TMPFILE
...
```

- Note: backtick substitution is older and better supported by shells (even **sh**!), although it does not allow nesting.
- Sub-note: is readability a matter of concern speaking of bash ? No ?! seriously ?

UCL - INGI
Département d’ingénérie informatique
www.info.ucl.ac.be

Loops

Scene 34: the Bridge of Death

- The famous one: `for` loop, in another famous scene

```
cat << _EOF_
  STOP!
  He who would cross the Bridge of Death
  Must answer me
  These questions three
  Ere the other side he see

_EOF_

for ITEM in "your name" "your quest" "the capital of Assyria";
do
    echo -n "  What is $ITEM ? "
    read ANSWER
done
echo
echo "  Anyway, you are cast into the Gorge of Eternal Peril"
echo "  AAAAARRRRRRRRRRRRRRRGGGGGHHH!!!!!!!... héhéhé..."
```


UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Loops (2)

- More boring(?), looping over args:

```
echo "Args number: $#"
for I in "$@"; do
    echo $I
done
```

- Note: both $@ and `$*` both expand to positional args, but, when between "
  - `"$*"` expands to `"$1c$2c...c$N"`, where c is the first char of `$IFS`
  - `"$@"` expands to `"$1"c"$2"c ... c"$N"`
  - take care if args can contain space !

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Loops (3)

- `while` (see also `until`)

```
#!/bin/bash

N=1
while test "$N" -le "10"
do
    echo "Number $N"
    N=$[N+1]
done
```

- Example : dealing a file line by line

```
while read LINE; do
    # $LINE will contain each line of the file
    # with spaces and all
    ...
done < myfile.txt
```


UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Loops (4)

- Interrupt iteration:
  - **`break`** interrupt the loop and continue the script execution with the instruction following `done`
  - **`continue`** interrupt the current iteration, checking the test and possibly starting a new one
  - Example : globbing in for loops

```
for FILE in *.txt; do
    if ! test -r $FILE; then
        continue
    fi
    echo "=========================="
    echo "Content of $FILE"
    echo "=========================="
    cat $FILE
    echo "=========================="
    echo "End of $FILE"
    echo "=========================="
done
```


UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Did you notice this: N=$[$N+1] ?

Bash can do math !

- . . . integers only ! ☺
- a command: **`let`**
- `let "exp"` == `((expr))` or `$[expr]`

```
VAR=55                # Assign integer 55 to variable VAR.
((VAR = VAR + 1))     # Add one to variable VAR.
                      # Note the absence of the ’$’ character.
VAR=$((VAR + 1))      # Add one to variable VAR.
                      # Dereference as right value.
((++VAR))             # Another way to add one to VAR.
                      # Performs C-style pre-increment.
((VAR++))             # Another way to add one to VAR.
                      # Performs C-style post-increment.
echo $[VAR * 22]      # Multiply VAR by 22 and substitute
                      # the result into the command.
echo $((VAR * 22)) # Another way to do the above.
```


UCL - INGI
Département d’ingénérie informatique
www.info.ucl.ac.be

Bash can do math!

- Supported operators (decreasing precedence order):

```
id++ id--       variable post-increment and post-decrement
++id --id       variable pre-increment and pre-decrement
- +            unary minus and plus
! ~            logical and bitwise negation
**             exponentiation
* / %          multiplication, division, remainder
+ -            addition, subtraction
<< >>          left and right bitwise shifts
<= >= < >      comparison
== !=          equality and inequality
&              bitwise AND
^              bitwise exclusive OR
|              bitwise OR
&&             logical AND
||             logical OR
expr?expr:expr conditional evaluation
= *= /= %= += -= <<= >>= &= ^= |=
               assignment
expr1 , expr2  comma (evaluation in order, last returned)
```

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Bash can do math!

- **`let`** or **`(( ... ))`** have a return code

  → may be used in conditionnal statements !

```
$ A=4
$ ((A == 2))
$ echo $?
1
$ if ((A == 2)); then echo 'What?'; else echo 'Cool!'; fi
Cool!
$ ((A == 4)) && echo 'Cool!'
Cool!
```

- use external command for floating point calculations

  ex. **dc**

```
$ dc -e "2 k 35 4 / p"
8.75
```

  - reverse-polish, `-e "exp"` eval an expression
  - `2 k`: set the precision to 2
  - `35 4 /`: compute 34.00 / 4.00 in top of the stack
  - `p`: pop the top of the stack

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Grab : nearly there…

We have now everething needed to understand this code:

```
WORDS=$(cat $TMPURL | \
    tr -d "\r\n" | \
    sed -e "s/<[^<>]*>//g" | \
    tr "[.;,: ]" "\n" | \
    grep "^[a-zA-Z]\{3,40\}$" | \
    tr "[A-Z]" "[a-z]" | \
    sort | uniq)

MYSQL="mysql -u robot ding"
for WORD in $WORDS; do
    echo "insert into ding_mot VALUES ('$WORD');" | $MYSQL
    # and yes, we should check before insert ! ;-)
done
```

Like many DBM, MySQL offers a command line tools: **`mysql`**.

Providing authentication, **you can simply pipe SQL directly into it!**

- `-u` set the username (no pass for this example)
- `ding` id the database name

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

A word about MySQL and Bash

- Although simple (and slow) the MySQL command line is **very powerfull**, as soon as the performances are not an issue (i.e. always with Bash, often in general)
- **`mysql`** takes auth tokens as arguments

```
mysql -u user -p password database
```

- **`mysql`** reads SQL commands from stdin

```
echo "select * from table;" | \
mysql -u user -p password database
```

- **`mysql`** writes results on stdout (tabbed columns layout)

⇒ no driver, no layered API, no ResultSet (or worse), no whatever: **simply use stdout/stdin i.e. read and write text !**

. . . and pipe it again ! ☺

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

A word about MySQL and Bash


```
$ MYSQL="mysql -u robot ding"
$ cat ding.sql | $MYSQL
$ echo "show tables;" | $MYSQL
Tables_in_ding
ding_mot
ding_occurence
ding_url
$ echo "select * from ding_occurence limit 6;" | $MYSQL
url     mot     nbr
http://en.wikipedia.org/wiki/Bash       after   3
http://en.wikipedia.org/wiki/Bash       another 4
http://en.wikipedia.org/wiki/Bash       assign  1
http://en.wikipedia.org/wiki/Bash       backslash       5
http://en.wikipedia.org/wiki/Bash       beginners       1
http://en.wikipedia.org/wiki/Bash       bourne  12
```

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Grab : world domination is at hand !

- We have all tools needed to complete the script :
  - count word occurencies
  - find external links
  - play around with MySQL
- We'll use functions to clean the code a bit (what?)

```
findwords () {
    tr -d "\r\n" | \
    sed -e "s/<[^<>]*>//g" | \
    tr "[.;,: ]" "\n" | \
    grep "^[a-zA-Z]\{3,40\}$" | \
    tr "[A-Z]" "[a-z]" | \
    sort | uniq
}

WORDS=$(cat $TMPURL | findwords)
```


UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Functions

- Two type of function declaration:

```
myfuncname () {
    # do something
    # ... or not ?
}
```

```
function myfuncname {
    # do something
    # ... really ?
}
```

- functions are like external scripts :
  - must be declared before called (simply by their name)
  - take positionnal arguments
  - exit status
    - `return <int>` terminate the function
    - `<int>` will be the exit status of the function
    - with no `return`, the exit status will be the one of the last command in the function
    - as for a command, get it with "$?"
  - have standard flows: redirection and substitutions are possible

```
WORDS=$(cat $TMPURL | findwords)
```


UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Grab : here we are !

```
MYSQL="mysql -u robot ding"
WORDS=$(cat $TMPURL | findwords)
# dumb DB management to remain simple
for WORD in $WORDS; do
  echo "insert into ding_mot VALUES ('$WORD');" | $MYSQL
  OCC=$(cat $TMPURL | occurs $WORD)
  echo "insert into ding_occurence values ('$URL', '$WORD', '$OCC')" |\
    $MYSQL
done

echo "Managing links in $TMPURL"
LINKS=$(cat $TMPURL | externlinks)
for LINK in $LINKS; do
    echo "insert into ding_url values ('$LINK', null);" | $MYSQL
done

echo "Updating DB for visited link"
echo "update ding_url set last_update=current_date() where url='$URL'" |\
  $MYSQL
```

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Grab : the functions


```
# Utility functions (very simple versions)
# ================================================
# find ascii words into HTML
findwords () {
    tr -d "\r\n" | \
    sed -e "s/<[^<>]*>//g" | \
    tr "[.;,: ]" "\n" | \
    grep "^[a-zA-Z]\{3,40\}$" | \
    tr "[A-Z]" "[a-z]" | \
    sort | uniq
}

# count ascii words into HTML
occurs () {
    tr -d "\r\n" | \
    sed -e "s/<[^<>]*>//g" | \
    tr "[.;,: ]" "\n" | \
    grep "^[a-zA-Z]\{3,40\}$" | \
    grep $1 | \
    wc -l
}
```


UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Grab : the functions (2)

```
# find external links into HTML
externlinks () {
    grep -i "a href" | \
    sed -e "s/^.*[Aa] [Hh][Rr][Ee][Ff]=\"//g" | \
    sed -e "s/\".*$//g" | \
    grep http
}
# ================================================
```

- Exercices:
  - improve the DB management with existence check and so on
  - Write the script allowing to **search** into the DB
  - Write a **slurper** that will index the web, using `grab` recusively
  - Buy Google and rule the world ! (good luck for this one, remember me in case of success)

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

More dark magic ?

A glimpse of what remains in the shadows

- Arrays:

```
$ A=(a 2 C)
$ echo ${A[0]}  # first element
a
$
$ echo ${A[*]}  # all elements
a 2 C
$ echo ${A[@]}  # idem
a 2 C
$ A[3]=4        # assign
$ echo ${A[@]}
a 2 C 4
$ echo ${#A[@]} # number of elements (id. ${#A[*]})
4
```

→ many (**many!**) subtle issues arise with Bash arrays. Use them with care!

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

More dark magic ?

- String manipulation with `${ ... }`

```
$ S=abcdef
$ echo ${#S}     # count chars
6
$ echo ${S:3}    # extract substring beginning at 3
def              # also note ${S:(-3)}
$ echo ${S:3:1} # extract substring  beginning at 3
d                # of 1 char length (idem ${S:(-3):1})
$ echo ${S#*c}  # glob removal at the beginning (greedy)
def              # for non-greedy, use ${S##*c}
$ echo ${S%c*}  # glob removal at the end (greedy)
ab               # for non-greedy, use ${S%%c*}
$ echo ${S/[a-z]/@}     # string substitution (first match)
@bcdef
$ echo ${S//[a-z]/@}    # all matches
@@@@@@
$ echo ${S/#[a-z]/@}    # first match at the beginning
@bcdef
$ echo ${S/%[a-z]/@}    # first match at the end
abcde@
```


UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

More dark magic ?

- Process substitution: `<(process)` or `>(process)`
- Restricted shell: `#!/bin/bash -r`
- Indirect reference: `eval var1=\$$var2`

```
$ A=var
$ var=3
$ echo $A
var
$ eval B=\$$A
$ echo $B
3
$ eval $A=4
$ echo $A
var
$ echo $var
4
```


UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

More dark magic ?

- Multihtreaded ? Multi-process ! → **Subshells**:

  `( command 1; command 2; ... command n; )`

  - run in parallel, with their own environment
  - ex. of synchronize via **`lockfile`**
  - utilities: **`wait`**, **`sleep`**, **`kill -SIG`**
  - pipe them: `ls -a | (command)`

- ":", the NOP that does more than other NOPs!

```
:
echo $?          # 0, : is TRUE !
: > myfile       # Deleting file contents, but preserving
                 # the file itself, with all permissions
                 # id. cat /dev/null > myfile
: This is a comment generating an error, ( if [ $x -eq 3] ).
: ${username=‘whoami‘}
# ${username=‘whoami‘}  without the leading : gives an error
#                       unless "username" is a command/builtin
```

- And many more ! But **Beware of the dark side! If once you start down the dark path, forever will it dominate your destiny.**

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be

Questions ?

No question ?

What a fine varlet thou art !

Go to your local Bash wizard,

Methinks you may now be knighted !

UCL - INGI
Département d'ingénérie informatique
www.info.ucl.ac.be