A Gentle Introduction to Bash
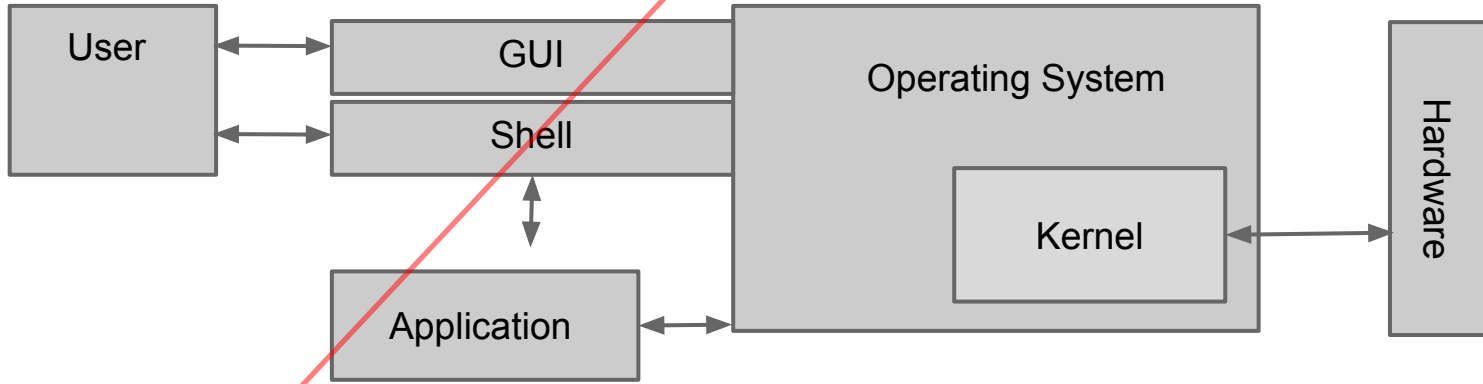
Incontri Informatici 2015
MOX 22/01/2015
Carlo de Falco

What is Bash?

- a UNIX shell program is a command interpreter that enables the user to interact with application software, the operating system (OS), and with the Kernel that controls Hardware (HW).
- **sh** a.k.a "Bourne Shell" was the default Unix shell of Unix Version 7
- **bash** "Bourne Again Shell" is the GNU shell. On GNU/Linux and most modern UNIX-like systems /bin/sh is a *symlink* to /bin/bash
- **bash** is Free Software, it is a compatible and standard conforming extension to **sh**.

Bash as command line interpreter

- When logging-in to a computer on [console](#) or via a [terminal emulator](#), the the OS starts a `bash` process in *interactive* mode.

```
Last login: Fri Jan 16 18:56:00 2015 from tosca.mate.polimi.it
(default)carlo@mox33 ~ $
```

- Enter a command, then press `enter`. The command is *interpreted* and executed then the shell waits to read a new command.
- `bash` built-in commands:

```
(default)carlo@mox33 ~ $ help
GNU bash, version 4.1.2(1)-release (x86_64-redhat-linux-gnu)
These shell commands are defined internally.  Type `help' to see this list.
Type `help name' to find out more about the function `name'.
Use `info bash' to find out more about the shell in general.
Use `man -k' or `info' to find out more about commands not in this list.
```

Shell commands

- **`bash`** is a full-fledged interpreted programming language

- shell commands may contain

    - conditionals
      ```
      if, test, case, [ … ]
      ```

    - cycles
      ```
      for, while, until
      ```

    - arithmetic expressions
      ```
      let, (( … ))
      ```

- example :
  ```
  select pippo in `ls`; file $pippo; done
  ```

Starting an application

- To start a (binary) executable type the name of the file and press enter

```
(default)carlo@mox33 home $ ls
my_directory
(default)carlo@mox33 home $ cd my_directory/
(default)carlo@mox33 my_directory $ ls
(default)carlo@mox33 my_directory $ ls -a
.  ..
(default)carlo@mox33 my_directory $ pwd
/u/carlo/home/my_directory
(default)carlo@mox33 my_directory $ which pwd
/bin/pwd
(default)carlo@mox33 my_directory $ which ls
alias ls='ls --color=auto'
        /bin/ls
(default)carlo@mox33 my_directory $ 
```

- Command line options are passed to the application as `(int argc, char **argv)`
- The OS consists of the Kernel plus many small utility provided as separate executables
- If a *directory* is in the `$PATH` then one does not need to type the full *path*

```
(default)carlo@mox33 ~$ printenv PATH
/usr/bin:/bin:/usr/sbin:/sbin:/usr/local/bin
```

man and info

- `help` works for shell built-in commands
- to get info for a <mark>command</mark> that is <mark>installed as a separate executable</mark> use `man`

  ```
  $ man ls
  ```

- some complex applications provide a more detailed manual that can be read usin `info`

  ```
  info latex

  info octave
  ```

- if a detailed manual is not available `info` still displays manpages

- to search manpages use

  ```
  man -k string
  ```

Environment variables

- Environment variables are used to set options for the behaviour of the shell or other applications
- List environment variables:

```
tosca.mate.polimi.it > env
HOSTNAME=tosca.mate.polimi.it
TERM=xterm-256color
SHELL=/bin/bash
HISTSIZE=1000
SSH_CLIENT=10.48.137.75 63111 22
SSH_TTY=/dev/pts/4
USER=carlo
LS_COLORS=
```

export NOME_VARIABILE=valore:
per settare il valore della variabile di ambiente
printenv NOME_VARIABILE: stampa il valore della
variabile (non serve $, solo variabili di ambiente)
echo $NOME_VARIABILE: stampa il valore della
variabile (serve $, qualsiasi variabile)
echo NOME_VARIABILE: stampa "NOME_VARIABILE"
($ serve in sostanza per deferenziare, per mostrare il
valore della variabile e non il nome)
unset NOME_VARIABILE: cancelliamo il valore della
variabile (ora è vuota)

- Set, print, unset value of a variable

```
tosca.mate.polimi.it > export PIPPO=cane
tosca.mate.polimi.it > printenv PIPPO
cane
tosca.mate.polimi.it > echo $PIPPO
cane
tosca.mate.polimi.it > unset PIPPO
tosca.mate.polimi.it > printenv PIPPO
tosca.mate.polimi.it >
```

- "$" indicates *variable expansion*

```
tosca.mate.polimi.it > echo PIPPO e"'" un $PIPPO
PIPPO e' un cane
```

Return values

- Each command executed returns a 'return value'

```
$ cat pippo.c
int main (int argc, char
**argv)
{ return 0;}
$ gcc -o pippo pippo.c
$ ./pippo
$ echo $?
0
```

```
$ cat topolino.c
int main (int argc, char **argv)
{ return 2;}
$ gcc -o topolino topolino.c
$ ./topolino
$ echo $?
2

$ cat pluto.c
int main (int argc, char **argv)
{ return -1;}
$ gcc -o pluto pluto.c
$ ./pluto
$ echo $?
255
```

- '$?' is a special variable containing the return value of the last executed command
- There is a convention that 0 indicates success, nonzero indicates failure

In Linux, Unix and other POSIX-compatible systems, the wait system call sets a *status* value of type int packed as a bitfield with various types of child termination information. If the child terminated by exiting (as determined by the WIFEXITED macro; the usual alternative being that it died from an uncaught signal), SUS specifies that the low-order 8 bits of the exit status can be retrieved from the status value using the WEXITSTATUS macro in wait.h; As such, POSIX-compatible exit statuses are restricted to values 0-255, the range of an unsigned 8-bit integer.

STDOUT STDERR STDIN

$ cat pippo.cc

#include <iostream>

int main (int argc, char **argv)

{

  std::cout << "this is the standard output strem"
          << std::endl;

  std::cerr << "this is the standard error stream"
          << std::endl;

  return 0;
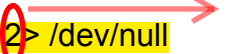
}

$ g++ -o pippo pippo.cc

$ ./pippo

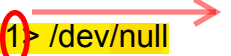this is the standard output strem

this is the standard error stream

$ ./pippo 2> /dev/null → Indica std::err

this is the standard output strem

$ ./pippo 1> /dev/null → Indica std::out

this is the standard error stream

$ ./pippo > /dev/null 2>&1

$

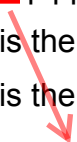$ ./pippo > pippo.txt 2>&1

$ cat pippo.txt

this is the standard output strem

this is the standard error stream

Serve per mostrare il contenuto di pippo.txt
Il file non deve essere necessariamente un .txt

Redirection

- stdout to file   Anzichè essere stampato a video, viene stampato su file

  ```
  echo "something" > filename
  ```

- stderr to file   std::err non vengono stampati a video ma su file

  ```
  ./pippo 2> filename
  ```

- stdout to stderr   std::out viene trasformato in std::err e entrambi vengono stampati su file

  ```
  ./pippo 2> filename 1>&2
  ```

- stderr to stdout   std::err viene trasformato in std::out

  ```
  ./pippo > filename 2>&1
  ```

- stderr and stdout to file

  ```
  /pippo &> filename
  ```

Piping

You can redirect the STDOUT/STDERR of a program to another program:

```
$ ls
pippo       pippo.c         pippo.cc      pippo.txt
$ ls | grep txt
pippo.txt
$
```
grep serve per cercare la stringa "txt" all'interno della directory

the "piping" can include multiple stages

```
$ $ ls | grep txt | sed 's/txt/png/g'
pippo.png
```
Setta delle opzioni di visualizzazione dei file

Saving STDOUT into a variable

$(command) and `command` expand to the output of command

```
$ for filename in `ls | grep txt`
do
  echo "change name of ${filename} to $(basename $filename .txt).ascii"
done

change name of pippo.txt to pippo.ascii

$
```

The output can be assigned to a variable:

```
export NEWNAME=$(basename pippo.txt .txt).ascii
```

Initialization files

- Global configuration files

```
/etc/profile
/etc/bashrc
```

- User configuration files

```
~/.bash_profile
~/.profile
~/.bashrc
```

- hint : to avoid confusion set one file as a *symlink* to the other so you have only one startup file to debug.

Scripts

- Scripts are text files containing a list of shell commands

```
tosca.mate.polimi.it > cat helloworld
echo 'Hello, World!'

tosca.mate.polimi.it > bash helloworld
Hello, World!
```

- Scripts can be made *executable* using a shebang

```
tosca.mate.polimi.it > cat helloworld
#!/bin/sh
echo 'Hello, World!'

tosca.mate.polimi.it > chmod u+x helloworld
tosca.mate.polimi.it > ./helloworld
Hello, World!
tosca.mate.polimi.it > 
```

Bash functions

```
!/bin/bash
# BASH FUNCTIONS CAN BE DECLARED IN ANY ORDER
function function_B {
    echo Function B.
}
function function_A {
    echo $1
}
function function_D {
    echo Function D.
}
function function_C {
    echo $1
}
```

```
# FUNCTION CALLS
# Pass parameter to function A
function_A "Function A."
function_B
# Pass parameter to function C
function_C "Function C."
function_D
```

Variables in scripts

```bash
#!/bin/bash
#Define bash global variable
#This variable is global and can be used anywhere in this bash script
VAR="global variable"
function bash {
#Define bash local variable
#This variable is local to bash function only
local VAR="local variable"
echo $VAR
}
echo $VAR
bash
# Note the bash global variable did not change
# "local" is bash reserved word
echo $VAR
```

More resources

- GNU Bash Website

- Wikipedia page about Bash

- Bash Programming HOWTO

- Advanced Bash Scripting guide

- Bash Guide for Beginners