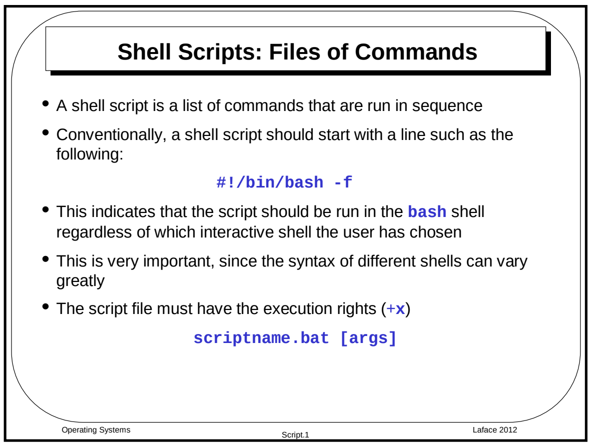


Shell Scripts: Files of Commands

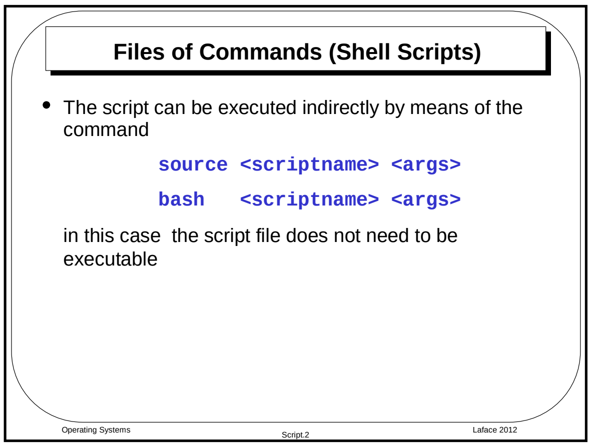
- A shell script is a list of commands that are run in sequence
- Conventionally, a shell script should start with a line such as the following:

```
#!/bin/bash -f
```

- This indicates that the script should be run in the `bash` shell regardless of which interactive shell the user has chosen
- This is very important, since the syntax of different shells can vary greatly
- The script file must have the execution rights (`+x`)

```
scriptname.bat [args]
```

Operating Systems Script.1 Laface 2012



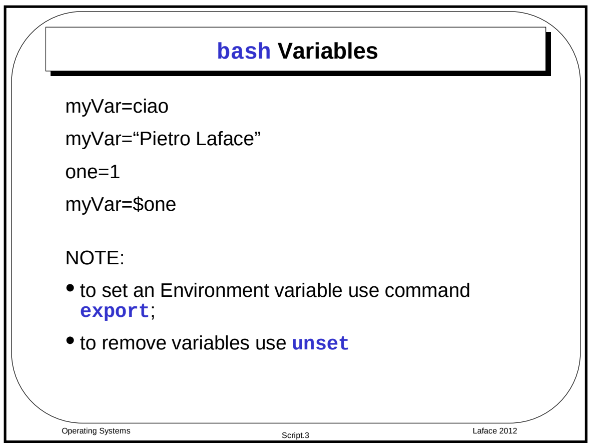
Files of Commands (Shell Scripts)

- The script can be executed indirectly by means of the command

```
source <scriptname> <args>
bash   <scriptname> <args>
```

in this case the script file does not need to be executable

Operating Systems Script.2 Laface 2012



bash Variables

myVar=ciao

myVar="Pietro Laface"

one=1

myVar=$one

NOTE:

- to set an Environment variable use command `export`;
- to remove variables use `unset`

Operating Systems Script.3 Laface 2012

Predefined Variables

Command line parameters:

– **$1**	first parameter
– **$#**	number of parameters
– **$0**	command name
– **$***	string including all the parameters
– **$?**	exit code of the last executed process
– **$$**	process PID

Interactive Input (`read`)

```
          read var1 var2 … varn
#!/bin/bash
#
echo –n ”Write a sentence: ”
#     (–n means no newline)
read one two other
echo ” The first word is: $one”
echo ” The second word is: $two”
echo ” The rest of the line is: $other”
```

Arrays

- Definition
  – a list of values between parentheses
- Access
  – using C notation
  – but also ranges are allowed

Arrays - Example

```
v=(a b ”Pietro Laface”)
echo ${v[2]}
echo ${v[*]}
echo ${v[@]}
unset v[subscript]
```

Special Characters: Quoting

`’’` Enclosing characters in single quotes preserves the literal value of each character within the quotes

- the shell does not expand the variables within `’’`

```
# myVar=”Ciao”
# echo ’Content of myVar: $myVar’
Content of myVar: $myVar
```

A single quote may not occur between single quotes, even when preceded by a `\` `($’\047’)`

Special Characters `””` : Quoting

Enclosing characters in double quotes preserves the literal value of all characters within the quotes, with the exception of `$`, `` ` ``, and `\`

The characters `$` and `` ` `` retain their special meaning within double quotes

The `\` retains its special meaning only when followed by one of the following characters: `$`, `` ` ``, `”`, `\`, or `newline`

A `”` may be quoted within double quotes by preceding it with a `\`

Special Characters: Quoting

- the shell expands the variables within ””

```
$ myVar=”Ciao”
$ echo ”Content of myVar: $myVar”
Content of myVar: Ciao
```

Quoting: Escape Character \

`\` removes the special meaning of a character or word to the shell

```
$ myVar=\$Ciao
$ echo ”Content of myVar: $myVar”
Content of myVar: $Ciao
```

`\` followed by `newline` continues the command on the next line

```
find / -name \* -exec wc –c {} \;  | \
awk –f istogram.awk | gnuplot isto.plt
```

Command substitutions: Special character `

Command substitution allows the output of a command (with any trailing `newline` deleted) to replace the command name

`` `commands` ``	`\` retains its literal meaning except when followed by `$`, `` ` ``, or `\`
`$(commands)`	no character in command has special meaning

```
$ myVar=`awk ’BEGIN{print sqrt(4)}’`
$ echo ”Content of myVar: $myVar”
Content of myVar: 2
```

Arithmetic Expressions

Evaluation is done in long integers with no check for overflow, though division by 0 is trapped and flagged as an error

`$[expr] , $((expr))` or `command expr`

```
num1=5
num2=$[$num1*3+1]
```
(NO blanks around `=`)

```
num2=$(($num1*3+1))
```

command `expr`

```
echo `expr $num1 \* 3 + 1`
```

Escape or quote for shell

Arithmetic Expressions

All the arithmetic and logical operators of C are allowed in arithmetic expressions,

plus exponentiation `**` `((i++))`

```
$ echo $((2**30 % 6))
4
$ i=4;j=2;
$ (( j+=$((++i)) )); echo $j
7
$ echo $((--j))
6
$ echo $((j==i))
0
```

Conditional expressions

`[ expr ]`

notice the blanks in `[_expr_]`

`[` is a reserved word, not a token like `(`

Conditional expressions

```
#!/bin/bash –f
# this is file test.bat
str1=”abc”
str2=”abd”
echo ”str1 = $str1”; str2=$str2
if [ $str1 == $str2 ]
then
  echo ”str1 equal to str2”
else
  echo ”str1 not equal to str2”
fi

$ bash test.bat
str1 = abc; str2 = abd
str1 not equal to str2
```

String Comparison Operators

String relational operators:

`==`	equal
`!=`	not equal
`-n`	length > 0 ; string exist
`-z`	length = 0 ; string doesn’t exist
`s1 < s2`	s1 precedes s2
`s1 > s2`	s1 follows s2

Arithmetic Comparison

`-eq`	equal
`-ge`	>=
`-le`	<=
`-ne`	not equal
`-gt`	>
`-lt`	<

```
if [ $n1 -ge $n2 ]
then
   . . .
fi
```

File Operators

`-d`	directory
`-f`	regular file
`-e`	file exists
`-r`	read permission
`-w`	write permission
`-x`	exec permission
`-s`	file dimension > 0
`-L`	is a symbolic link

```
if [ -d $myVar ]
then
  . . .
fi

if [ -e file ]; then echo exists; else echo no file; fi
```

`if` Construct

```
if [ expr ];
then
    command list;
elif [ expr ]; then
    command list;
else
    command list;
fi
```

`for` Construct

```
for myVar in list
  do
    command list
  done
```

Examples:

```
for x in 1 2 3 4
  do
   echo $x
  done
for x in `ls`
  do
   mv $x ../backup
  done
```

while Construct

```
while [ expr ]
  do
    command list
  done
```

Example:

```
#/bin/bash -f
# creates 15 empty files
ind=0
while [ $ind -lt 15 ]
do
  touch ”xxx$ind”    # creates an empty file
  ((ind++))          # ind=$[$ind+1]
                     # ind=`expr $ind + 1`
done
```

case Construct

```
case str in
   str1)
         command list ;;
   str2 | str3)
          command list;;
   *)
          command list ;;
  esac
```

Notice !!

case - Example

```
case $1 in
   01 | 1) echo ”January”;;
   02 | 2) echo ”February”;;
   ...
   12) echo ”December”;;
   *) echo ”Invalid”;;
  esac
```

case - Menu Example

```
echo -n ”Enter the name of an animal: ”
read ANIMAL
echo -n ”The $ANIMAL has ”
case $ANIMAL in
   horse | dog | cat)  echo -n ”four”;;
   man | kangaroo )     echo -n ”two”;;
   *)        echo -n ”an unknown number of”;;
 esac
 echo " legs"
```

cut Command

```
cut [OPTION]... [FILE]...
```

Print selected parts of lines from each FILE to standard output.

`-d=DELIM`	use **DELIM** instead of **TAB** for field delimiter
`-b=LIST`	output only these bytes
`-f=LIST`	output only these fields

paste Command

```
paste [OPTION]... [FILE]...
```

Write lines consisting of the sequentially corresponding lines from each **FILE**, separated by **TAB**s, to standard output.

-d=LIST
use **LIST** of chars instead of **TAB** for field delimiters

-s paste one file at a time instead of in parallel

Example using `cut` and `paste`

```
$ cal 1 2003
   January 2003
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

```
$ cal 1 2003 | cut -b1-2 > file_a
$ cal 1 2003 | cut -b19-20 > file_b
$ paste  file_a file_b
Su      Sa
         4
 5      11
12      18
19      25
26
$ paste -s file_a file_b
      Su          5     12     19     26
      Sa     4   11     18     25
```

tr

`tr [OPTION]... SET1`

Translate, squeeze, and/or delete characters from standard input, writing to standard output.

`-d` delete characters in **SET1**, do not translate

`-s` replace each input sequence of a repeated character that is listed in **SET1** with a single occurrence of that character

tr

Examples:

```
$ echo amico | tr a-z A-Z
AMICO
$ echo ancd | tr a-c m-z
mnod
$ echo abc | tr a-m n-o
abc
noo
```

wc

`wc [OPTION]... file`

prints the number of lines, words and bytes in `file`

Options:

`-l`	prints the number of lines only
`-w`	prints the number of words only
`-c`	prints the number of bytes only

wc

Examples:

```
$ wc triang.awk
 11     13     137 triang.awk
$ wc < triang.awk          # notice input
  redirection
 11     13     137          # no file name
$ wc –l < triang.awk
11
```

uniq

`uniq [OPTIONS] [INPUT] [OUTPUT]`

Discard all but one of successive identical lines from `INPUT` (or `stdin`), writing to `OUTPUT` (or `stdout`)

`-c`	prefix lines by the number of occurrences
`-d`	only print duplicate lines
`-i`	ignore differences in case when comparing

Example:

```
$ . . .| sort | uniq –i -d
```

dirname and basename

`dirname` returns the portion preceding the final slash of a pathname.

Example:

```
dirname  /usr/src/linux/mem.c
```

prints `/usr/src/linux`

`basename` returns the portion following the final slash of a pathname.

Example:

```
basename /usr/src/linux/mem.c
```

prints `mem.c`

```
basename /usr/src/linux/mem.c .c
```

prints `mem`

Signals

The kernel send a signal to a process to notify the occurrence of an event.

Processes react to signals

– Example:

A foreground process is terminated by pressing `CTRL-C` on the session terminal

`CTRL-C` is an event that the kernel notifies to the foreground running process

Users can send a signal to their running processes by means of the command `kill`

```
kill -sig pid
```

Signals List

```
$ kill -l
HUP INT QUIT ILL TRAP ABRT EMT FPE KILL BUS SEGV SYS
PIPE ALRM TERM URG STOP TSTP CONT CHLD TTIN TTOU POLL
XCPU XFSZ VTALRM PROF WINCH LOST USR1 USR2
```

This is the output of `cygwin`

Signal	Value	Notes
INT	2	produced by `CTRL-C`
KILL	9	forced termination
ALRM	14	produced by the kernel at the end of a time interval provided by the user using system call `alarm(number_of_seconds)`
USR1	30	Software signals available to users
USR2	31	

Process Reaction to Signals

When a process receives a signal its default behavior is to terminate

A process can define an alternative behavior for some signals :

- Ignore the signal (except signal `KILL`)
- Execute a user specified signal handler for that signal

Signal Management

Command `trap` is used to manage signal in shell scripts

`trap ”command” signal_list`

When a signal in `signal_list` occurs, execute `command`

`trap signal_list`

Reset to the default behavior the signals in `signal_list`

`trap ”” signal_list`

Ignores the signals in `signal_list`

Example

To remove an temporary file created by a script even if the script is interrupted before its natural end

```
trap ”rm -f /var/tmp/file.tmp; exit 0” 2
touch /var/tmp/file.tmp
. . . . . .
```