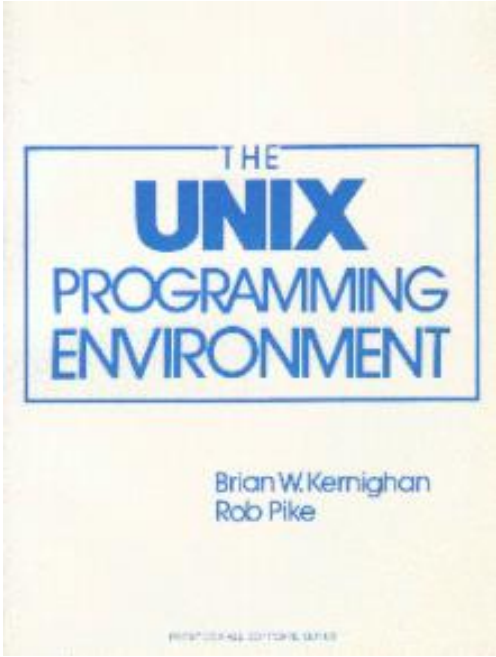
Laboratorio II

Corso A

Lezione 22

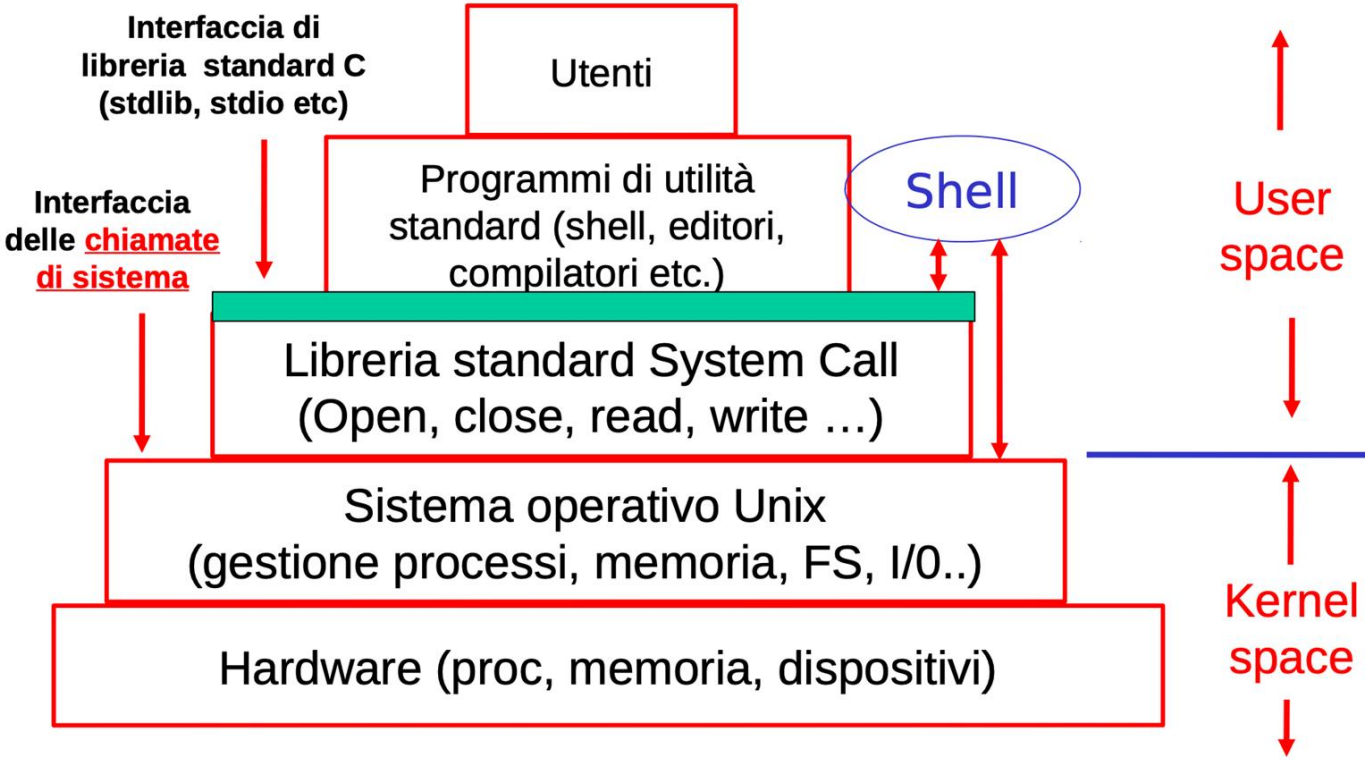
Programmazione Bash



Shell

- Shell
  - programma in cui l’utente può eseguire dei comandi (testuali) - interfaccia con la libreria di SC e Kernel
  - UNIX systems: shell, command line, terminal
  - è un interprete di comandi
    - interattivo - un comando alla volta
    - basato su script - programmi che utilizzano più comandi
      - comandi unix + strutture di controllo del flusso
      - tipicamente utilizzati per automatizzare operazioni di configurazione e operazioni su file
  - comandi
    - builtin - eseguiti nella stessa shell
    - eseguibili localizzati nel file system - eseguito in un processo shell separato (figlio) creato dalla shell corrente

Shell


Interfaccia di
libreria standard C
(stdlib, stdio etc)
Utenti
Programmi di utilità
standard (shell, editori,
compilatori etc.)
Shell
Interfaccia
delle chiamate
di sistema
Libreria standard System Call
(Open, close, read, write …)
Sistema operativo Unix
(gestione processi, memoria, FS, I/0..)
Hardware (proc, memoria, dispositivi)
User
space
Kernel
space

Shell

- Un sistema Unix tipicamente include vari shell - vari interpreti
- Tradizionali:
  - Bourne shell (sh) - sviluppato da Steven Bourne (Bell) per Unix 7 (1979)
    - sh - compatibility è diventato uno standard per sistemi Unix
  - Bourne Again Shell (bash) - sviluppato da Brian Fox (GNU) per Linux (1989)
  - Ancora utilizzati (a volte sh è in realtà un link ad un interprete più moderno)
- Alcuni più moderni
  - Dash - 1997 per Debian
- Al login viene avviato un shell di default (su laboratorio2 è bash)

Shell scripting

- Uno script è un file contenente vari comandi Unix - un programma - definisce un nuovo comando
- Per avviarlo:
  - invocare il shell giusto se non è quello di default, con il filename dello script come parametro
  - includere il nome dello shell nel file sulla prima riga (#!) - il file deve essere eseguibile

Esempio Hello World

Variabili di shell

- nome a cui associamo un valore
- i valori sono ***stringhe***
- il valore può essere acceduto con la sintassi `$nomeVar` o `${nomeVar}`
- una variabile può essere cancellata con `unset`

```
a029688@laboratorio2:~$ myVar=5
a029688@laboratorio2:~$ echo $myVar
5
a029688@laboratorio2:~$ myVar=alina
a029688@laboratorio2:~$ echo $myVar
alina
```

```
a029688@laboratorio2:~$ unset myVar
a029688@laboratorio2:~$ echo $myVar

a029688@laboratorio2:~$
```

Stringhe nella shell

- con o senza virgolette ? (quoted vs unquoted)
  - prima di eseguire i comandi, la shell esegue un processo di cosiddetta ***espansione di metacaratteri***
    - metacaratteri: *, ?, [], |,>,<, $ altri elementi del linguaggio shell
  - le virgolette e apici ‘proteggono’ la stringa contro l’espansione
    - le apici sono più ‘forti’
    - le virgolette permettono alcuni tipi di espansione - e.g. espansione di variabili
    - si può utilizzare anche “\” per proteggere ogni metacarattere nella stringa

Espansione di metacaratteri

```
a029688@laboratorio2:~/bashScripting$ echo *
hw.sh
a029688@laboratorio2:~/bashScripting$ echo '*'
*
```

espansione di percorso (globbing)

```
a029688@laboratorio2:~$ echo * PATH=$PATH *
Desktop Documents Downloads Music Pictures Public Templates Videos bas
hScripting head src PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr
/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin Desktop Document
s Downloads Music Pictures Public Templates Videos bashScripting head
src
a029688@laboratorio2:~$ echo "* PATH=$PATH *"
* PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/u
sr/games:/usr/local/games:/snap/bin *
a029688@laboratorio2:~$ echo '* PATH=$PATH *'
* PATH=$PATH *
```

espansione di variabili

Espansione di metacaratteri

```
a029688@laboratorio2:~$ var=*
a029688@laboratorio2:~$ echo $var
Desktop Documents Downloads Music Pictures Public Templates Videos bas
hScripting head src
a029688@laboratorio2:~$ echo '$var'
$var
a029688@laboratorio2:~$ echo "$var"
*
```

Variabili di shell predefinite

- nomi già definiti all'apertura della shell
- tengono traccia dello stato dell'istanza della shell e configurazioni di sistema (e.g. working directory, PATH)
- possono essere visualizzati col comando `set`
- il loro valore può essere cambiato

```
a029688@laboratorio2:~$ echo $SHELL
/bin/bash
a029688@laboratorio2:~$ echo $HOSTTYPE
x86_64
a029688@laboratorio2:~$ echo $HISTSIZE
1000
```

```
a029688@laboratorio2:~$ set| head -20
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:expand_alia
iranges:histappend:interactive_comments:login_shell:progcomp
BASH_ALIASES=()
BASH_ARGC=([0]="0")
BASH_ARGV=()
BASH_CMDS=()
BASH_COMPLETION_VERSINFO=([0]="2" [1]="10")
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="5" [1]="0" [2]="17" [3]="1" [4]="release
BASH_VERSION='5.0.17(1)-release'
COLUMNS=103
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/896179407/bus
DEFAULT_XDG_CONFIG_DIRS=/etc/xdg
DEFAULT_XDG_DATA_DIRS=/usr/local/share/:/usr/share/
DIRSTACK=()
EUID=896179407
GROUPS=()
HISTCONTROL=ignoreboth
HISTFILE=/home/local/ADUNIPI/a029688/.bash_history
a029688@laboratorio2:~$
```

Variabili di shell predefinite

- PS1 - prompt string per il prompt principale
- Può essere cambiato in qualsiasi stringa
  - Può contenere variabili speciale
    - \u - username
    - \s - shell
    - \w - working dir
    - \h - host

```
a029688@laboratorio2:~$ PS1="alina:>"
alina:>pwd
/home/local/ADUNIPI/a029688
alina:>echo $PS1
alina:>
alina:>echo ciao
ciao
alina:>
```

Variabili di shell predefinite

- PATH - lista di directory dove si trovano i comandi utilizzati dalla shell, separati da “:”
- Possiamo aggiungere altri directory al path

```
a029688@laboratorio2:~/bashScripting$ ./hw.sh
Hello world
a029688@laboratorio2:~/bashScripting$
```

```
a029688@laboratorio2:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin
:/bin:/usr/games:/usr/local/games:/snap/bin
a029688@laboratorio2:~$ PATH=~/bashScripting/:$PATH
a029688@laboratorio2:~$ echo $PATH
/home/local/ADUNIPI/a029688/bashScripting/:/usr/local/s
bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/g
ames:/usr/local/games:/snap/bin
a029688@laboratorio2:~$
```

```
a029688@laboratorio2:~$ hw.sh
Hello world
```

Variabili di ambiente

- le variabili di shell descrivono l'ambiente locale dell'istanza di shell
- possiamo definire delle variabili 'globali' visibili anche nei processi che la shell crea - "environment variables" - variabili di ambiente
- create con il comando `export nomevar`
- possiamo controllare la lista con comando `export`
- alcune variabili sono esportate di default (HOME, PATH, PWD)

```
a029688@laboratorio2:~/bashScripting$ MYVAR=~
a029688@laboratorio2:~/bashScripting$ echo $MYVAR
/home/local/ADUNIPI/a029688
a029688@laboratorio2:~/bashScripting$ export MYVAR
```

```
a029688@laboratorio2:~/bashScripting$ export | tail -17
declare -x MOTD_SHOWN="pam"
declare -x MYVAR="/home/local/ADUNIPI/a029688"
declare -x OLDPWD="/home/local/ADUNIPI/a029688"
declare -x PATH="/home/local/ADUNIPI/a029688/bashScripting/:/usr/lo
usr/local/games:/snap/bin"
declare -x PWD="/home/local/ADUNIPI/a029688/bashScripting"
declare -x SHELL="/bin/bash"
declare -x SHLVL="2"
```

Parametri

- Quando lanciamo uno script possiamo specificare dei parametri come negli altri comandi
- Accediamo al parametro *i* utilizzando `$i` (`$1`, `$2`, etc)
- Parametri speciali:
  - $0 - nome dello script
  - $* - tutti i parametri come unica stringa
  - $@ - lista di tutti i parametri - lista di stringhe
  - $# - numero di parametri
  - $$ - PID della shell

Esempio con parametri

Controllo del flusso

- strutture condizionali
  - `if-then-else, case`
- strutture iterative
  - `for, while, until`

if-then-else

```
if <condition> ; then
        lista comandi
elif <condition>; then
                lista comandi
else
                lista comandi
fi
```

condition - qualsiasi comando - ha esito 0 - (successo, true) o !=0 (errore, falso)

possiamo combinare comandi con &&, ||, !

```
 GNU nano 4.8                    hw.sh
#!/bin/bash

if diff -q  $1 $2; then
        echo file $1 e $2 identici
else
        echo file $1 e $2 diversi
fi
```

```
[a029688@laboratorio2:~/bashScripting$ ./hw.sh hw.sh hw.sh
file hw.sh e hw.sh identici
[a029688@laboratorio2:~/bashScripting$ ./hw.sh hw.sh ../head
Files hw.sh and ../head differ
file hw.sh e ../head diversi
a029688@laboratorio2:~/bashScripting$
```

case

```
case <expr> in
      <pattern>)
            lista comandi;;
      <pattern>)
            lista comandi;;
      …
esac
```

expr è in genere una variabile o una stringa da espandere

pattern è un valore o lista di valori separati da |

```
 GNU nano 4.8                          case.sh
#!/bin/bash

case $# in
        0 ) echo Ecco i contenuti del directory corrente $PWD; ls;;
        1 ) head $1;;
        2 ) if diff $1 $2 > /dev/null; then
                echo  i due file sono identici
        else
                echo i due file sono diversi
        fi ;;
        * ) echo  $0: troppi parametri ;;
esac
```

case

```
case <expr> in
    <pattern>)
        lista comandi;;
    <pattern>)
        lista comandi;;
    …
esac
```

expr è in genere una variabile o una stringa da espandere

pattern è un valore o lista di valori separati da |

```
a029688@laboratorio2:~/bashScripting$ ./case.sh
Ecco i contenuti del directory corrente /home/local/ADUNIPI/a029688/bas
hScripting
case.sh  hw.sh
a029688@laboratorio2:~/bashScripting$ ./case.sh case.sh
#!/bin/bash

case $# in
        0 ) echo Ecco i contenuti del directory corrente $PWD; ls;;
        1 ) head $1;;
        2 ) if diff $1 $2 > /dev/null; then
                echo  i due file sono identici
        else
                echo i due file sono diversi
        fi ;;
a029688@laboratorio2:~/bashScripting$ ./case.sh case.sh case.sh
i due file sono identici
a029688@laboratorio2:~/bashScripting$ ./case.sh case.sh hw.sh
i due file sono diversi
a029688@laboratorio2:~/bashScripting$ ./case.sh case.sh hw.sh case.sh
./case.sh: troppi parametri
a029688@laboratorio2:~/bashScripting$
```

for

```
for <var> [in <list>]; do
    <comandi>
done
```

```
 GNU nano 4.8                    headCat.sh
echo "questo script stampa le prime 3 righe"
echo " dei file in input "

for file in "$@"; do
        if  test -e "$file" && ! test -d "$file"; then
                head -3 "$file"
        fi
done
```

```
a029688@laboratorio2:~/bashScripting$ ./headCat.sh
questo script stampa le prime 3 righe
 dei file in input
a029688@laboratorio2:~/bashScripting$ ./headCat.sh  case.sh ciao
questo script stampa le prime 3 righe
 dei file in input
#!/bin/bash

case $# in
a029688@laboratorio2:~/bashScripting$ ./headCat.sh  *
questo script stampa le prime 3 righe
 dei file in input
#!/bin/bash

case $# in
echo "questo script stampa le prime 3 righe"
echo " dei file in input "

#!/bin/bash

if diff -q  $1 $2; then
#!/bin/bash

echo "questo script stampa solo i suoi parametri"
a029688@laboratorio2:~/bashScripting$
```

for

```
for <var> [in <list>]; do
    <comandi>
done
```

se <list> è omessa si itera tra i parametri dello script ($@)

```
  GNU nano 4.8                    showParams.sh
#!/bin/bash

echo "questo script stampa solo i suoi parametri"
for p ; do
        echo "$p"
done
```

```
a029688@laboratorio2:~/bashScripting$ ./showParams.sh
questo script stampa solo i suoi parametri
a029688@laboratorio2:~/bashScripting$ ./showParams.sh  *
questo script stampa solo i suoi parametri
case.sh
hw.sh
showParams.sh
a029688@laboratorio2:~/bashScripting$ ./showParams.sh  "*"
questo script stampa solo i suoi parametri
*
a029688@laboratorio2:~/bashScripting$ ./showParams.sh  ciao come stai
questo script stampa solo i suoi parametri
ciao
come
stai
a029688@laboratorio2:~/bashScripting$ ./showParams.sh  "ciao come stai"
questo script stampa solo i suoi parametri
ciao come stai
a029688@laboratorio2:~/bashScripting$
```

for

for <var> [in <list>]; do

    <comandi>

done

se <list> è omessa si itera tra i parametri dello script ($@)

```
  GNU nano 4.8                                  headCat1.sh
#!/bin/bash

echo "questo script stampa le prime 3 righe"
echo " dei file in $PWD "

for file in *; do
        if  ! test -d "$file"; then
                echo "first 3 lines of $file are: "
                head -3 "$file"
        fi
done
```

while

```
while <condition> ; do
        lista comandi
done
```

```
  GNU nano 4.8                          while.sh
#!/bin/bash

echo "questo script aspetta fin che l'utente non sia loggato"

while ! who | grep $1 > /dev/null ; do
        sleep 60
done

echo "l'utente $1 si è loggato"
```

```
[a029688@laboratorio2:~/bashScripting$ ./while.sh a029688
questo script aspetta fin che l'utente non sia loggato
l'utente a029688 si è loggato
[a029688@laboratorio2:~/bashScripting$ ./while.sh a029689
questo script aspetta fin che l'utente non sia loggato
^C
a029688@laboratorio2:~/bashScripting$
```

until

```
until <condition> ; do
        lista comandi
done
```

```
 GNU nano 4.8                    until.sh
#!/bin/bash

echo "questo script aspetta fin che l'utente non sia loggato"

until  who | grep $1 > /dev/null ; do
        sleep 60
done

echo "l'utente $1 si è loggato"
```

Condizioni - comando `test`

- `if <condition> ; while <condition> ; until <condition>` possono usare l'exit value di qualsiasi comando per controllare il flusso
- il comando `test` - facilita vari tipi di test con valore 0 (true, success) o 1 (false, error)
  - confronto di valori - interi, stringhe
  - caratteristiche di file

```
man test
```

- Un altro modo di scrivere un test: [], [[]] - con gli stessi parametri del comando test
  - [] equivalente a test
  - [[]] - più flessibilità - globbing, etc - (https://acloudguru.com/blog/engineering/conditions-in-bash-scripting-if-statements)

Condizioni - comando test

```
  GNU nano 4.8                                  test.sh
#!/bin/bash

if test $# = 1 ; then
        if test -e $1 && ! test -d $1; then
                head -5 $1
        else
                echo "Argument is not a file"
        fi
else
        echo  "usage: $0 filename "
fi
```

Condizioni - comando test

```
a029688@laboratorio2:~/bashScripting$ ./test.sh
usage: ./test.sh filename
a029688@laboratorio2:~/bashScripting$ ls
case.sh  headCat.sh  headCat1.sh  hw.sh  showParams.sh  test.sh  until.sh  while.sh
a029688@laboratorio2:~/bashScripting$ ./test.sh case.sh
#!/bin/bash

case $# in
        0 ) echo Ecco i contenuti del directory corrente $PWD; ls;;
        1 ) head $1;;
a029688@laboratorio2:~/bashScripting$ ./test.sh ciao
Argument is not a file
a029688@laboratorio2:~/bashScripting$ ./test.sh ciao ciao
usage: ./test.sh filename
```

Condizioni - comando test

```
  GNU nano 4.8                                  test.sh
#!/bin/bash

if [ $# = 1 ] ; then
        if [ -e $1 ] && ! [ -d $1 ] ; then
                head -5 $1
        else
                echo "Argument is not a file"
        fi
else
        echo  "usage: $0 filename "
fi
```

Operazioni aritmetiche

`man expr`

- Comando `expr` - permette di effettuare operazioni su interi
- Per ottenere il valore di un comando e assegnarlo ad una variabile: sostituzione di comandi (command substitution) - *un tipo di espansione*
  - `$(comando)`
  - `` `comando` ``

```
a029688@laboratorio2:~$ a=6
a029688@laboratorio2:~$ b=7
a029688@laboratorio2:~$ c=$(expr $a + $b)
a029688@laboratorio2:~$ echo $a+$b=$c
6+7=13
a029688@laboratorio2:~$
```

```
a029688@laboratorio2:~/bashScripting$ a=$(expr 3 + 9)
a029688@laboratorio2:~/bashScripting$ echo $a
12
a029688@laboratorio2:~/bashScripting$ a=`expr 5 + 8`
a029688@laboratorio2:~/bashScripting$ echo $a
13
```

Operazioni aritmetiche

Comando bc - operazioni su numeri reali

```
man bc
```

```
a029688@laboratorio2:~/bashScripting$ a=$(echo "5.4+3.2"|bc)
a029688@laboratorio2:~/bashScripting$ echo $a
8.6
a029688@laboratorio2:~/bashScripting$ a=$(echo "5/2"|bc)
a029688@laboratorio2:~/bashScripting$ echo $a
2
a029688@laboratorio2:~/bashScripting$ a=$(echo "scale=2;5/2"|bc)
a029688@laboratorio2:~/bashScripting$ echo $a
2.50
a029688@laboratorio2:~/bashScripting$
```

Operazioni aritmetiche

- *Espansione aritmetica* - (( ))
  - Permette di scrivere codice in stile C.

```
a029688@laboratorio2:~$ a=6
a029688@laboratorio2:~$ b=7
a029688@laboratorio2:~$ c=$(expr $a + $b)
a029688@laboratorio2:~$ echo $a+$b=$c
6+7=13
a029688@laboratorio2:~$ d=$((a+b))
a029688@laboratorio2:~$ echo $a+$b=$d
6+7=13
a029688@laboratorio2:~$ ((e=a+b))
a029688@laboratorio2:~$ echo $a+$b=$e
6+7=13
a029688@laboratorio2:~$
```

for C-like

```
for ((i=0;i<n;i++)); do
    <comandi>
done
```

```
  GNU nano 4.8                                  powers.sh
#!/bin/bash

for ((i=1; i<=$1; i++)) ; do
        echo $((i*i))
done
```

Array

- Collezione di stringhe indicizzate da interi (da 0) - possono essere sparse
- Creazione
  - arr=(file1 file2 file3); arr[10]=file4; arr=($(<command>))
- Indicizzazione
  - echo ${arr[0]}
  - echo ${arr[*]}, echo ${arr[@]}
- Numero di elementi (non vuoti)
- echo ${#arr[@]}
- Possiamo iterare con for
  - for f in ${arr[@]}; do <comando>; done

```
 GNU nano 4.8                    arrays.sh
arr=(file1 file2 file3)
arr[10]=file4

echo ${arr[0]}
echo ${arr[*]}
echo ${arr[@]}
echo ${#arr[@]}


for f in  ${arr[@]}; do
        echo $f
done
```

```
[a029688@laboratorio2:~/bashScripting$ ./arrays.sh
file1
file1 file2 file3 file4
file1 file2 file3 file4
4
file1
file2
file3
file4
a029688@laboratorio2:~/bashScripting$
```

Funzioni

- Associano a un nome un programma di shell
  - Può essere chiamato come comando builtin (senza attivare un altro thread)
- Sintassi per definire una funzione:

```
<nome> () {
    <lista comandi>
}
```

- Per cancellare una funzione:

```
unset -f <nome>
```

```
a029688@laboratorio2:~/bashScripting$ f1 (){
>         echo 'Sono nella funzione f1'
> }
a029688@laboratorio2:~/bashScripting$ f1
Sono nella funzione f1
```

Funzioni

- Possono prendere parametri posizionali e speciali come tutti i comandi (e script)

  `$0, $1….`

  `$@, $#, $*`

- Per visualizzare informazioni sulle funzioni definite nella shell corrente:
  - comando `declare`

    -f : funzioni e codice

    -F : solo nomi di funzioni

  - comando `type -all`

```
a029688@laboratorio2:~/bashScripting$ f1 (){
> echo 'Sono nella funzione f1'
> echo "Ho ricevuto $# parametri: $*"
> }
a029688@laboratorio2:~/bashScripting$ f1
Sono nella funzione f1
Ho ricevuto 0 parametri:
a029688@laboratorio2:~/bashScripting$ f1 ciao hello
Sono nella funzione f1
Ho ricevuto 2 parametri: ciao hello
a029688@laboratorio2:~/bashScripting$
```

```
a029688@laboratorio2:~/bashScripting$ f1 (){
>         echo 'Sono nella funzione f1'
> }
a029688@laboratorio2:~/bashScripting$ f1
Sono nella funzione f1
a029688@laboratorio2:~/bashScripting$ declare -F | grep f1
declare -f f1
a029688@laboratorio2:~/bashScripting$ type -all f1
f1 is a function
f1 ()
{
    echo 'Sono nella funzione f1'
}
a029688@laboratorio2:~/bashScripting$
```

Funzioni

- Possiamo ‘importare’ nella shell corrente funzioni da un file
- Due comandi diversi che eseguono ***nella shell corrente*** il codice contenente in un file:
  - `. script` (diverso da `./script` !!!!)
  - `source script`

```
  GNU nano 4.8                    functions.sh
f1 (){
        echo 'Sono nella funzione f1'
}
```

```
[a029688@laboratorio2:~/bashScripting$ f1
f1: command not found
[a029688@laboratorio2:~/bashScripting$ ./functions.sh
[a029688@laboratorio2:~/bashScripting$ f1
f1: command not found
[a029688@laboratorio2:~/bashScripting$ . functions.sh
[a029688@laboratorio2:~/bashScripting$ f1
Sono nella funzione f1
a029688@laboratorio2:~/bashScripting$
```

```
[a029688@laboratorio2:~/bashScripting$ f1
f1: command not found
[a029688@laboratorio2:~/bashScripting$ source functions.sh
[a029688@laboratorio2:~/bashScripting$ f1
Sono nella funzione f1
a029688@laboratorio2:~/bashScripting$
```

Funzioni

- Le variabili definite in una funzione sono sempre globali
- Possono essere limitate allo scope della funzione con `local`

```
 GNU nano 4.8                    functions.sh
f1 (){
        var='valore f1'
        echo "In f1: var=$var"
}
```

```
[a029688@laboratorio2:~/bashScripting$ source functions.sh
[a029688@laboratorio2:~/bashScripting$ echo $var

[a029688@laboratorio2:~/bashScripting$ f1
In f1: var=valore f1
[a029688@laboratorio2:~/bashScripting$ echo $var
valore f1
a029688@laboratorio2:~/bashScripting$
```

```
 GNU nano 4.8                    functions.sh
f1 (){
        local var
        var='valore f1'
        echo "In f1: var=$var"
}
```

```
[a029688@laboratorio2:~/bashScripting$ source functions.sh
[a029688@laboratorio2:~/bashScripting$ echo $var

[a029688@laboratorio2:~/bashScripting$ f1
 In f1: var=valore f1
[a029688@laboratorio2:~/bashScripting$ echo $var

 a029688@laboratorio2:~/bashScripting$
```

Operazioni su stringhe

```
a029688@laboratorio2:~$ var=laboratorio2
a029688@laboratorio2:~$ echo ${#var}
12
```

- Lunghezza di una stringa
  - ${#<var>}
- Sottostringa
  - ${<var>:<offset>}, ${<var>:<offset>:<length>}

```
a029688@laboratorio2:~$ var=laboratorio2
a029688@laboratorio2:~$ echo $var
laboratorio2
a029688@laboratorio2:~$ echo ${var:2}
boratorio2
a029688@laboratorio2:~$ echo ${var:2:2}
bo
a029688@laboratorio2:~$
```

Operazioni su stringhe

- Pattern matching
  - Trova e rimuove pattern da stringhe
    - pattern - espressioni con caratteri più wildcard *,?, []
  - Occorrenze iniziali
    - ${<var>#<pattern>}, ${<var>##<pattern>} : trova pattern all'inizio e rimuove l'occorrenza più corta/lunga
  - Occorrenze finali
    - ${<var>%<pattern>}, ${<var>%%<pattern>} : trova pattern alla fine e rimuove l'occorrenza più corta/lunga

```
[a029688@laboratorio2:~$ var=laboratorio2
```

```
[a029688@laboratorio2:~$ echo ${var#[a-z]}
aboratorio2
[a029688@laboratorio2:~$ echo ${var##[a-z]}
aboratorio2
[a029688@laboratorio2:~$ echo ${var#*([a-z])}
laboratorio2
[a029688@laboratorio2:~$ echo ${var##*([a-z])}
2
[a029688@laboratorio2:~$ echo ${var#+([a-z])}
aboratorio2
[a029688@laboratorio2:~$ echo ${var##+([a-z])}
2
[a029688@laboratorio2:~$ echo ${var#?([a-z])}
laboratorio2
[a029688@laboratorio2:~$ echo ${var##?([a-z])}
aboratorio2
```

```
[a029688@laboratorio2:~$ echo ${var%%?([1-9])}
laboratorio
[a029688@laboratorio2:~$ echo ${var%?([1-9])}
laboratorio2
```

Operazioni su stringhe

```
[a029688@laboratorio2:~$ var=laboratorio2
```

- Sostituzione di sottostringhe
  - ${<var>/<pattern>/<string>}: trova pattern e sostituisce la prima occorrenza più lunga con string
  - ${<var>//<pattern>/<string>} : sostituisce tutte le occorrenze
  - se pattern comincia con # - sostituisce l'occorrenza iniziale
  - se pattern comincia con % - sostituisce occorrenza finale

```
[a029688@laboratorio2:~$ echo $var
laboratorio2
[a029688@laboratorio2:~$ echo ${var/a/A}
lAboratorio2
[a029688@laboratorio2:~$ echo ${var//a/A}
lAborAtorio2
[a029688@laboratorio2:~$ echo ${var/#?a/AA}
AAboratorio2
[a029688@laboratorio2:~$ echo ${var/%a*/AA}
lAA
a029688@laboratorio2:~$
```

Dettagli su espansione

Vari tipi di espansione, in ordine preciso

1. Espansione degli alias e della history
   - alias - nomi per comandi esistenti (e.g. sh può essere un alias verso una shell diversa)
   - !n - espanso come l'ennesimo comando nella history
   - !! - espanso come l'ultimo comando nella history
2. Espansione delle parentesi graffe
   - Permette la creazione di stringhe usando pattern
   - <prefisso>{<elenco>}<suffisso>
3. Espansione della tilde (~)
   - ~utente sostituito con la home di utente
   - ~ sostituito con la home dell'utente loggato

```
a029688@laboratorio2:~$ for f in  user_{u1,u2,u3}_file.txt; do echo $f; done
user_u1_file.txt
user_u2_file.txt
user_u3_file.txt
a029688@laboratorio2:~$
```

```
a029688@laboratorio2:~$ echo ~
/home/local/ADUNIPI/a029688
a029688@laboratorio2:~$ echo ~adminpolo2
/home/adminpolo2
a029688@laboratorio2:~$
```

Dettagli su espansione

Vari tipi di espansione, in ordine preciso

4. Espansione delle variabili
    - ${<variabile>}
5. Sostituzione dei comandi
    - $(<comando>)
6. Espansione delle espressioni aritmetiche intere
    - $((<espressione>)) , $[<espressione>]
7. Suddivisione in parole
    - I delimitatori sono contenuti nella variabile IFS (Internal Field Separator) che per default contiene spazio, tab e newline (‘ ’, ‘\t’, ‘\n’)
    - La suddivisione avviene solo per stringhe senza quote (unquoted)

Dettagli su espansione

Vari tipi di espansione, in ordine preciso

8. Espansione di percorso o *globbing*
    - Pattern con wildcard *, ?, [...] vengono espansi a nomi di file nel folder corrente (PWD)

```
a029688@laboratorio2:~/bashScripting$ ls *.sh
arrays.sh  es4.sh        helloWorld.sh  powers.sh      test.sh        while.sh
case.sh    functions.sh  hw.sh          safeDelete.sh  undoDelete.sh
es1.sh     headCat1.sh   log.sh         showParams.sh  until.sh
a029688@laboratorio2:~/bashScripting$ ls a*.sh
arrays.sh
a029688@laboratorio2:~/bashScripting$ ls *[1-9].sh
es1.sh  es4.sh  headCat1.sh
a029688@laboratorio2:~/bashScripting$ ls ?a*.sh
case.sh  safeDelete.sh
a029688@laboratorio2:~/bashScripting$
```

Dettagli su espansione

- Quoting
  - ‘ ’ - proteggono da tutti i tipi di espansione
  - “ ” - proteggono da espansione di percorso e suddivisione in parole (tutte le altre vengono ancora eseguite)
- Escaping
  - \ - protegge solo il prossimo carattere da tutti i tipi di espansione

Redirezione output in file

- Inizialmente ogni processo ha 3 descrittori di file aperti: 0,1,2
- Possiamo aprire altri file aggiungendo altri descrittori (n)
- Operatore > associa un descrittore *n* a un file aperto per la scrittura
  - <comando> [n]> file
  - se n manca si associa lo stdout (1)
    - e.g. cat file.txt > file_copy.txt
  - possiamo usarlo per aprire un file per scrittura
    - exec 3>file.txt - descrittore 3 associato a file.txt aperto per scrittura
    - per chiudere file: exec n>&-
  - >> - modalità append
  - &> - redirezione stderr & stdout simultanea

Redirezione input da file

- Operatore < associa un file aperto in modalità lettura a un descrittore di file *n*
  - <comando> [n]< file
  - se n manca si associa lo stdin (0)
    - e.g. grep alina <file.txt
  - possiamo usarlo per aprire un file in lettura
    - exec 4<file.txt - descrittore 4 associato a file.txt aperto per lettura
    - per chiudere file: exec n<&-

Leggere file

Se abbiamo un descrittore in lettura possiamo leggere riga per riga con comando `read`

`-u<n>` - specifica il descrittore da dove leggere

```
  GNU nano 4.8                    readFile.sh
exec 3<$1

while read -u3 line; do
        echo $line
done
```

```
a029688@laboratorio2:~/bashScripting$ cat f.txt
ciao
hello
ciao
hello
a029688@laboratorio2:~/bashScripting$ ./readFile.sh f.txt
ciao
hello
ciao
hello
```

Scrivere in un file

Se abbiamo un descrittore in scrittura possiamo scrivere usando la redirezione

```
  GNU nano 4.8                                readFile.sh
exec 3<$1
exec 4>$2

while read -u3 line; do
        echo $line >&4
done
```

```
[a029688@laboratorio2:~/bashScripting$ ./readFile.sh f.txt fcopy.txt
[a029688@laboratorio2:~/bashScripting$ cat fcopy.txt
ciao
hello
ciao
hello
[a029688@laboratorio2:~/bashScripting$ cat f.txt
ciao
hello
ciao
hello
a029688@laboratorio2:~/bashScripting$
```

Combinazione di comandi

- Sequenza di comandi:
  - <command1> ; <command2>
  - Viene eseguito command1 poi command2
  - Exit status è quello del command2
- Esecuzione in background:
  - <command> &
  - Comando viene avviato e controllo torna subito al prompt della shell corrente
- Operatori logici
  - <command1> && <command2>
    - esegue command1 poi solo se *ha avuto successo* esegue anche command2
    - exit status è successo solo se entrambi i comandi hanno avuto successo
  - <command1> || <command2>
    - esegue command1 poi solo se *non ha avuto successo* esegue anche command2
    - exit status è successo solo se almeno uno dei comandi ha avuto successo

Combinazione di comandi

- Pipelining
  - Combinare comandi con |
  - lo stdout (1) di un comando è redirezionato nel stdin (0) del comando successivo
  - Exit status è dato dall’ultimo comando eseguito
  - Ogni comando eseguito in un processo shell separato

Debugging

- Non eseguire script come **root** prima di essere sicuri che funzionano
- Non testare su file e folder importanti
- Attenti agli **spazi** tra vari elementi del linguaggio
- Attenti alle espansioni - e loro ordine
- Opzioni di shell per debugging - comando `set` (`help set`)
  - `-n` -noexec - verifica solo la sintassi senza eseguire
  - `-v` -verbose - stampa comando prima di eseguire
  - `-x` -xtrace - stampa comando dopo l'espansione prima di eseguire
- `$-` : variabile che mostra le opzioni settate
- usando `+` invece di `-` si può disattivare un'opzione

Opzioni Debugging

```
[a029688@laboratorio2:~/bashScripting$ echo $-
himBHs
[a029688@laboratorio2:~/bashScripting$ ls *.sh
case.sh  es4.sh       helloWorld.sh  log.sh     safeDelete.sh  test.sh        until.sh
es1.sh   headCat1.sh  hw.sh          powers.sh  showParams.sh  undoDelete.sh  while.sh
[a029688@laboratorio2:~/bashScripting$ set -x
[a029688@laboratorio2:~/bashScripting$ ls *.sh
+ ls --color=auto case.sh es1.sh es4.sh headCat1.sh helloWorld.sh hw.sh log.sh powers.sh safeDelete.sh s
howParams.sh test.sh undoDelete.sh until.sh while.sh
case.sh  es4.sh       helloWorld.sh  log.sh     safeDelete.sh  test.sh        until.sh
es1.sh   headCat1.sh  hw.sh          powers.sh  showParams.sh  undoDelete.sh  while.sh
[a029688@laboratorio2:~/bashScripting$ echo $-
+ echo himxBHs
himxBHs
[a029688@laboratorio2:~/bashScripting$ set +x
+ set +x
[a029688@laboratorio2:~/bashScripting$ echo $-
himBHs
[a029688@laboratorio2:~/bashScripting$ ls *.sh
case.sh  es4.sh       helloWorld.sh  log.sh     safeDelete.sh  test.sh        until.sh
es1.sh   headCat1.sh  hw.sh          powers.sh  showParams.sh  undoDelete.sh  while.sh
a029688@laboratorio2:~/bashScripting$
```

Domande?