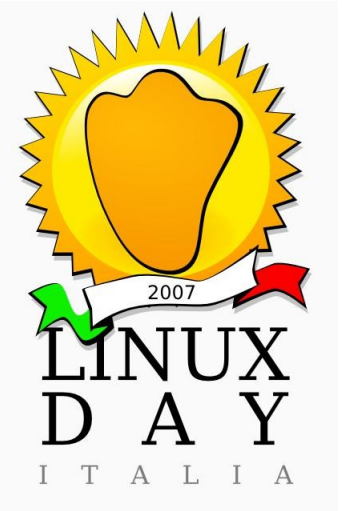
Lo scripting ben temperato

Ovvero un tocco di eleganza nella programmazione della shell

Ing. Davide Bolcioni
Consulente Sistemi Informativi



Gli script vanno bene per ...

- Personalizzare le proprie macchine
  - invocati da **`cron`**, **`at`** ...
- Sistematizzare operazioni ricorrenti

  `risistema-foto [`*`directory`* **`[`*`camera`*`]]`

Gli script non vanno bene per ...

- Programmi da distribuire
  - meglio Perl, Python, Tcl ...
  - sottili differenze
- Algoritmi complessi
  - mancano strutture dati (Wirth)
  - difficoltà di debug
- Problemi di performance
  - relativi

Note di buona creanza - 1

- Non si mette l'estensione
  - si vede che venite da Windows
  - ha un altro significato, "source me"
- Si mette nel **`$PATH`**
  - in **`/usr/local/bin`** per alcuni
  - in **`$HOME/bin`** per altri

Note di buona creanza - 2

- Non si cambia la shell di root
  - usate `#!  /la/mia/shell` in cima
- Non scrivere per csh e derivati
  - cercare “csh programming considered harmful”
  - bastano ksh, bash, ash, dash, ...
- Funzionalità minime
  - opzione -h
  - codice di ritorno
  - componibilità

Funzioni di comodo

- Onde scrivere script di comodo utilizzo
  - vocabolario
- Semplici
  - 80% risultato, 20% sforzo
- Portabili
  - fino a un certo punto

Emettere avvisi

```
stem() { # basename
  echo "$1" | sed 's:^.*/::'
}
warn() {
  echo `stem "$0"`: "$@" >&2
}
```

Onde poter scrivere

```
[ -e "$cam" ] && warn "Trovata \"$cam\""
```

Uscita su errore

```
die() {
  warn "$@"
  exit 3
}
```

Onde poter scrivere

```
mount "$cam" || die "Non trovo \"$cam\""
```

Opzioni standard

```
talkative=:
laconic=:
while getopts 'hvq' option; do
  case $option in
    h) usage ;;
    v) talkative=: ; laconic=false ;;
    q) talkative=false ; laconic=: ;;
    *) usage "Invalid option \"$option\"" ;;
  esac
done
```

Messaggi e silenzi

- Il silenzio è la regola
- Avvisare in situazioni insolite

```
$talkative || warn "Nessuna immagine"
```

- Prolissi su richiesta (-v)

```
$laconic || echo "Trasformo foto $i di $n"
```

Aiuto

```
exit_usage() {
  echo 'usage:' `stem "$0"` '[-v][-q]' ...
  echo '       ' `stem "$0"` '-h'
  ...
  exit "$1"
}
usage() {
  [ -z "$1" ] && exit_usage 0
  warn "$@"
  exit_usage 2 >&2
}
```

Piccole attenzioni - 1

- Di fronte agli ostacoli ...
  - sintassi `$(...)` non garantita
  - sintassi `` `...` `` scomoda
- ... si fa un giro attorno

```
echo "usage: $(stem "$0") [-v][-q]"
echo usage: `stem "$0"` '[-v][-q]'
```

- Attenzione agli apici ...
  - prima servivano per allineare gli spazi
  - altrimenti non servono

Piccole attenzioni - 2

```
mounted() {
   [ 1 = `mount | sed -n "s|^.*$1.*$|1|p"`]
}
```

Onde scrivere

```
mounted /tmp || warn "molto strano"
```

restituisce "`missing ]`"

Piccole attenzioni - 3

```
mounted() {
   [ 1 = `mount | sed -n "s|^.*$1.*$|1|p"` ]
}
```

con

```
mounted /tmp || warn "molto strano"
```

funziona ma con

```
mounted /xxx || warn "normale"
```

restituisce "**test: argument expected**"

Piccole attenzioni - 4

```
mounted() {
   [ x1 = x`mount | sed -n "s|^.*$1.*$|1|p"` ]
}
```

si sacrifica l'eleganza al risultato

nascondendolo dentro una funzione

Funzioni e variabili locali

- Differenze nelle funzioni

```
function f
f()
```

- Differenze nelle variabili locali

```
typeset
local
```

- Morale: usa **f()** senza variabili locali

Uso di comandi esterni - 1

```
check() {
  eval "$1" 2>/dev/null && return 0
  warn "$2"
  exit 3
}
```

Onde poter scrivere

```
check "tail </dev/null" "No tail(1) in $PATH"
```

Uso di comandi esterni - 2

- A volte occorre industriarsi ...

```
check "sed q </dev/null" "No sed"
check "gzip </dev/null >/dev/null" "No gzip"
check "tar cf - /dev/null >/dev/null" "No tar"
check "echo a | grep a >/dev/null" "No grep"
check "echo a | tr -d a >/dev/null" "No tr"
```

- ... meglio non farsi male da soli

```
check "registra-foto -h >/dev/null" "Eh ?"
```

Uno scheletro di script

```
#! /usr/bin/ksh

# funzioni: warn, die, check, ...

# variabili: talkative, laconic, ...

# while getopts ...

# check ...

# istruzioni modificanti ...
```

Considerazioni finali

- Componibilità
  - script che invocano script (`check`)
  - script che invocano la grafica (`kdialog`)
  - grafica che invoca gli script (Tcl/Tk)
- La "Advanced Bash Scripting Guide"
  - una botte di trucchi
  - non suggerisce dove **fermarsi**