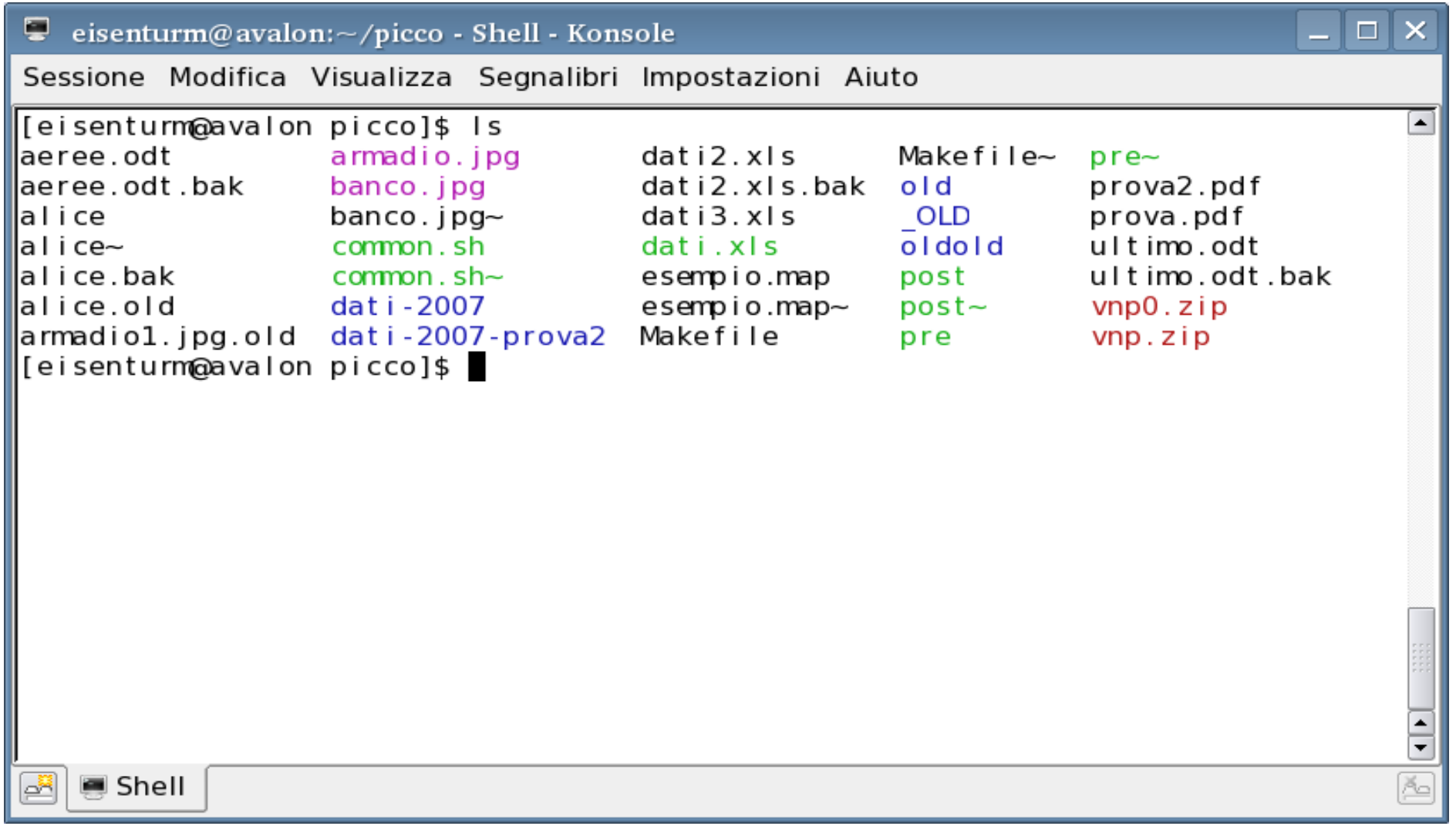
Taglia e Cuci, Bash e Dash ...

... tutto pulito e in ordine

Ing. Davide Bolcioni
Consulente Sistemi Informativi

Una tipica directory ...



... e i suoi problemi

- Il lavoro è in corso
  - non ci si sofferma su come farlo
- Niente controllo delle revisioni
  - manca la dimestichezza
  - formati non di testo
  - andrebbe installato e configurato
- Non si vuole perdere il lavoro fatto
  - copie selvagge a scopo di rassicurazione
  - disordine

Legge ed Ordine ?

- Si impone la seguente procedura
  - prima di modificare, fate una copia
  - se non esistono copie, la prima ha suffisso `.1`
  - se esistono, ruotare i suffissi di una posizione
    - `.1` diventi `.2`
    - `.2` diventi `.3`
    - ... ricordatevi di cominciare dal fondo :-)
    - tenere al massimo 9 copie
    - infine creare `.1`

UNIX pensaci tu

- Le procedure vanno bene per i computer
  - gli umani si dedichino ad attività più costruttive
  - per esempio realizzare delle procedure
    - che costituiscano un comodo strumento di lavoro
    - che non richiedano impegno spropositato
    - costruite con elementi già conosciuti
- La nostra procedura manipola file
  - niente di meglio della shell per questo lavoro
    - magari esiste già ... ma supponiamo di no

30 secondi di analisi ...

- Difetti della procedura proposta
  - non riduce il disordine
    - ne automatizza il diffondersi
  - si applica file per file
    - i numeri non sono necessariamente allineati
  - un pò complicata da spiegare

... non sono bastati, ma quasi

- Facciamo un comando che
  - fa una fotografia di una directory
    - tutti i file sono colti in un momento coerente
    - può essere un pò pesante
  - la mette via da qualche parte
    - di solito queste fotografie non sono poi usate davvero
    - le mettiamo in `$HOME/.archive`
  - usando un nome che suggerisca i contenuti
    - ad esempio *`directory-AAAAMMGG-HHMM`*
  - ... e passa la paura

Prima bozza

```
#! /bin/sh
# keep-a-copy - quick personal peace of mind

die() {
  echo "${0##*/}: $1" 1>&2
  exit "${2:-1}"
}

ad="$HOME/.archive"
[ -d "$ad" ] || mkdir "$ad" || exit 1

die "finito" 0
```

Prima bozza - note

- La funzione `die` serve per uscire
  - emettendo un messagio in standard error
  - con un codice di ritorno diverso da zero
- La directory d'archivio
  - se manca la creo
  - se la creazione fallisce esco
    - il messaggio d'errore lo mette `mkdir`

Seconda bozza

```
#! /bin/sh
# keep-a-copy - quick personal peace of mind

die() { ... }

ad="$HOME/.archive"
[ -d "$ad" ] || mkdir "$ad" || exit 1

if [ -d "$1" ]; then
  name="${1##*/}"
fi
echo $name
```

Seconda bozza - note

- Argomenti del comando
  - la directory di cui tenere una copia
  - se ad esempio invocherò

    ```
    keep-a-copy picco
    ```

    avrò che `$1` vale `picco`
  - ma se invocassi

    ```
    keep-a-copy progetti/copia
    ```

    avrei che `$1` vale `progetti/copia`
  - `./copia ?`

Terza bozza

```
...

full_path() {
  case $1 in
  /*) echo $1 ;;
  ./*) echo "$PWD/${1##./}" ;;
  *) echo "$PWD/$1" ;;
  esac
}

ad="$HOME/.archive"
[ -d "$ad" ] || mkdir "$ad" || exit 1
[ -d "$1" ] || die "not a directory $1"

dir=$(full_path "$1")
where=$(dirname "$dir")
what=$(basename "$dir")

echo $dir
echo $where
echo $what
```

Terza bozza - note

- La funzione `full_path`
  - trasforma l'argomento in un percorso assoluto
    - l'ultimo componente è la directory che voglio
  - se comincia con `/` va già bene
  - se non ha niente davanti premetto `$PWD`
  - se comincia con `./` lo tolgo
  - non ho pensato al caso `../copia`

Migliorare full_path ...

```
...

full_path() {
  case $1 in
  /*) echo $1 ;;
  ./*) echo "$PWD/${1##./}" ;;
  ../*) echo "$(dirname "$PWD")/${1##../}" ;;
  *) echo "$PWD/$1" ;;
  esac
}

...
```

... non è così facile ...

- Rimuovere `../`
  - richiede di consumare un componente di `$PWD`
    - lo consumo con `dirname`
- non basta
  - potrei avere `../../copia`

... si riuscirebbe anche ...

```
...

munge_dot() {
  case $2 in
  ./*) echo $(munge_dot "$1" "${2##./}") ;;
  ../*) echo $(munge_dot "$(dirname "$1")" "${2##../}") ;;
  *) echo "$1/$2" ;;
  esac
}

full_path() {
  case $1 in
  /*) echo $1 ;;
  ./*) echo $(munge_dot "$PWD" "${1##./}") ;;
  ../*) echo $(munge_dot "$(dirname "$PWD")" "${1##../}") ;;
  *) echo "$PWD/$1" ;;
  esac
}

...
```

... ma è quasi programmare

- Ho usato una funzione ricorsiva
  - Taglia dal fondo di `$PWD`
  - Cuce insieme i pezzi e torna indietro

```
/home/db/progetti/altro/ld ../../picco
/home/db/progetti/altro ../picco
/home/db/progetti picco
/home/db/progetti/picco
```

- Sto manipolando delle sequenze di testo
  - La shell non è lo strumento giusto

UNIX pensaci tu – di nuovo

```
...

full_path() {
  readlink -f "$1"
}

...
```

Quarta bozza

```
...
usage() {
  [ -n "$2" ] && echo "$2"
  echo "usage: ${0##*/} directory"
  echo "       ${0##*/} -h"
  exit "${1:-2}"
}

while getopts :h opt ; do
case $opt in
  h) usage 0 ;;
  ?) usage 2 "invalid option $OPTARG" 1>&2 ;;
esac
done
shift $(($OPTIND - 1))

ad="$HOME/.archive"
[ -d "$1" ] || usage 2 "arg must be a directory, not $1" 1>&2
[ -d "$ad" ] || mkdir "$ad" || exit 1
...
```

Le tasse sono bellissime

- La funzione `usage` è una “tassa”
  - non serve a me che la scrivo
  - serve agli altri che usano il programma
    - compreso il futuro me stesso che non si ricorderà
  - Altre “tasse” che non capisco
    - ✔ corretta gestione dei codici di ritorno
    - ✔ *standard output* e *standard error*
    - ✗ la pagina di manuale
    - ✗ il pacchetto d'installazione
    - ✗ la *policy* SELinux
    - ✗ la *bash completion*

Quinta bozza

```
...

while getopts :h opt ; do
case $opt in
  h) usage 0 ;;
  ?) usage 2 "invalid option $OPTARG" 1>&2 ;;
esac
done
shift $(($OPTIND - 1))

ad="$HOME/.archive"
[ -d "$1" ] || usage 2 "arg must be a directory, not $1" 1>&2
[ -d "$ad" ] || mkdir "$ad" || exit 1

dir=$(full_path "$1")
where=$(dirname "$dir")
what=$(basename "$dir")

tar -c -j -f "$ad"/"$what".tar.bz2 -C "$where" "$what"
```

Tanto rumore per nulla ?

- Esiste già il comando `tar` che
  - fa un archivio data una directory
  - lo comprime
- Va bene così ...
  - abbiamo programmato poco
    - la shell serve per collegare strumenti esistenti
  - usando una funzionalità generica
    - costruiamo un nostro strumento specifico di lavoro
      - invece di imparare quello con cui ha lavorato un altro
  - lo combineremo con altri strumenti

UNIX, che ore sono ?

```
...

dir=$(full_path "$1")
where=$(dirname "$dir")
what=$(basename "$dir")
ca="$what-$(date '+%Y%m%d-%H%M').tar.bz2"

tar -c -j -f "$ad"/"$ca" -C "$where" "$what"
```

In caso di guai

```
...

dir=$(full_path "$1")
where=$(dirname "$dir")
what=$(basename "$dir")
ca="$what-$(date '+%Y%m%d-%H%M').tar.bz2"

tar -c -j -f "$ad"/"$ca" -C "$where" "$what" || rm -f “$ca”
```

Ancora le “tasse”

```
...

dir=$(full_path "$1")
where=$(dirname "$dir")
what=$(basename "$dir")
ca="$what-$(date '+%Y%m%d-%H%M').tar.bz2"

if ! tar -c -j -f "$ad"/"$ca" -C "$where" "$what" ; then
  rm -f “$ca”
  exit 1
fi
```

Alcuni dettagli importanti

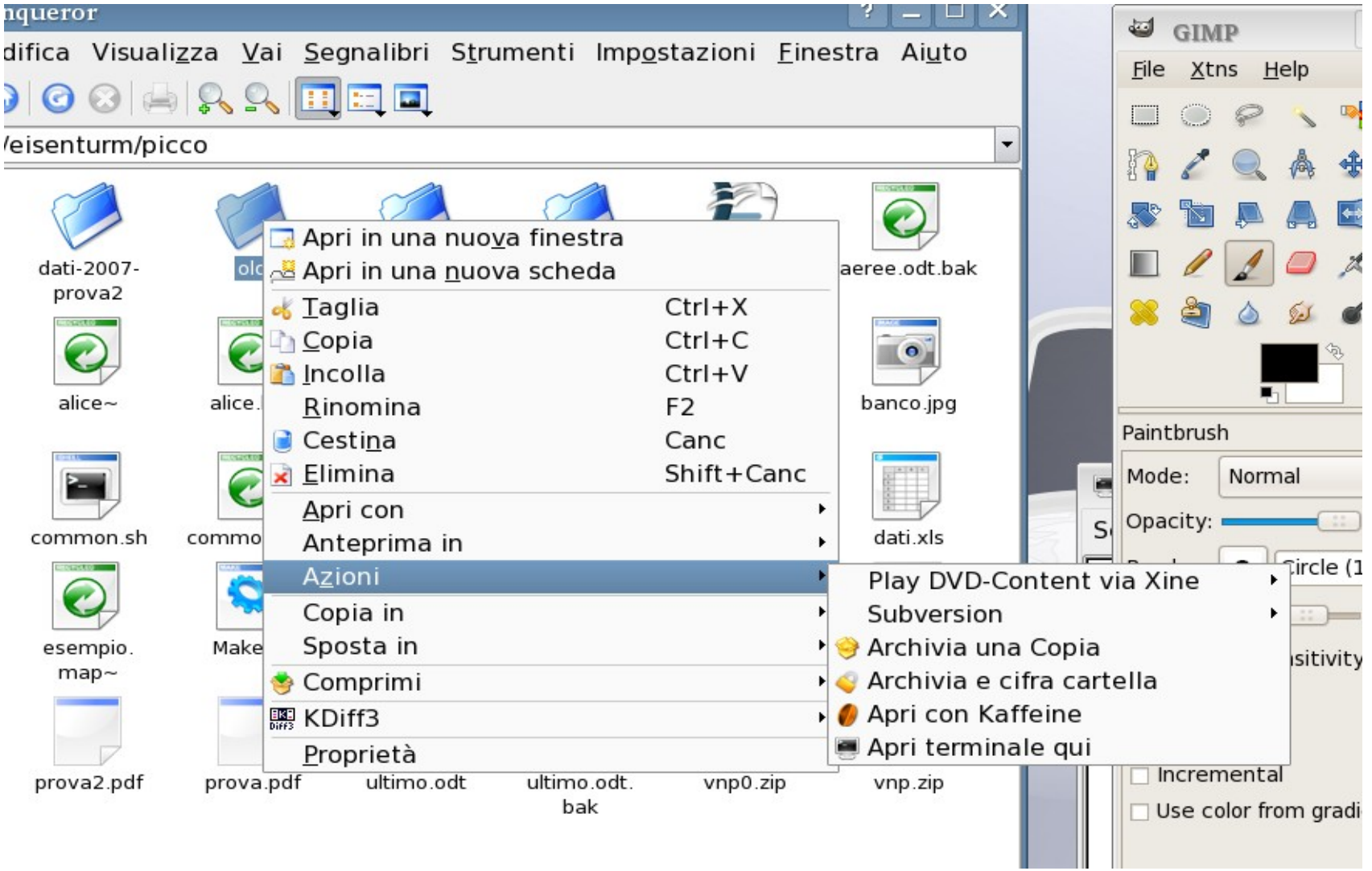
- se non ci passano nessun argomento ?
  - non abbiamo `$1`
  - facciamo stia per . ovvero la directory corrente
- non abbiamo escluso `$HOME/.archive`
  - riempiremmo il disco
- non tutte le shell sono uguali
  - famiglia delle “Bourne” shell: `bash`, `dash`, ...
  - altre famiglie
  - le differenze sono piccole
    - da nascondere dentro le funzioni

Una volta realizzato ...

```
$ cat .kde/share/apps/konqueror/servicemenus/Archivia.desktop
[Desktop Entry]
ServiceTypes=inode/directory
Actions=keepit

[Desktop Action keepit]
Name=Archive a Copy
Name[it]=Archivia una Copia
Exec=keep-a-copy %f
Icon=package
```

... si riusa in KDE



Ulteriori sviluppi

- Integrazione grafica migliorata
  - dialogo che segnali un errore
  - non modifiche ma composizione
    - avrò `kde-keep-a-copy` che chiama `keep-a-copy`
- Gestione delle copie archiviate
  - rimuovi le copie non più utilizzate

```
find $HOME/.archive -type f -atime +90 -delete
```

  - rimuovile automaticamente usando `cron`

Conclusioni

- Linux non è Windows
  - Linux è UNIX
    - occorre tornare indietro da un binario morto
    - strumenti invece di applicazioni
    - un sistema invece di un veicolo per applicazioni
  - realizzare i propri strumenti è facile
    - non occorre temere le personalizzazioni
      - funzionano e funzioneranno
  - se si pagano le “tasse”
    - aderendo a convenzioni che non si capiscono
      - gli strumenti si integrano nel sistema

Conclusioni

- Linux non è Windows
  - Linux è UNIX
    - occorre tornare indietro da un binario morto
    - strumenti invece di applicazioni
    - un sistema invece di un veicolo per applicazioni
  - realizzare i propri strumenti è facile
    - non occorre temere le personalizzazioni
      - funzionano e funzioneranno
  - se si pagano le “tasse”
    - aderendo a convenzioni che non si capiscono
      - gli strumenti si integrano nel sistema

Per approfondire

- Documentazione
  - Advanced Bash Scripting Guide
  - KDE Developer Documentation
- Pratica
  - Dimenticate Windows e installate Linux
  - Buttatevi
  - Ci vediamo l'anno prossimo
    - *mi stupirete*