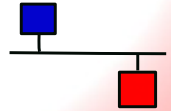
La Shell Lavora Per Noi

Come accelerare compiti comuni
Utilizzando la Shell, la sua programmazione ed i comandi di Linux

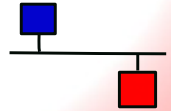


Memorizzare comandi frequenti

- Ho due monitor.
- Per configurare il secondo monitor a destra del principale, dovo usare il comando:

  `xrandr --output HDMI-1-0 --right-of eDP-1`
- Ricordarsi tutti i parametri non è semplice quindi cosa possiamo fare?
- Possiamo salvare questa riga in un semplice file di testo, magari coin il nome **`schermoADestra`**
- Per eseguirla potremo poi, al prompt, digitare:

  `bash schermoADestra`



Semplificare il richiamo

- Più semplice ma ancora complicato.
- Per semplificare dobbiamo fare due cose:
- Aggiungere una prima riga che indichi che programma eseguire (**bash**, nel nostro caso, o meglio **/bin/bash**, per indicare il percorso completo) in cima al file.
- Il file diventerà così:

```
#!/bin/bash
xrandr --output HDMI-1-0 --right-of eDP-1
```

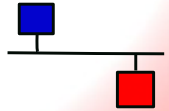
- Rendere il file eseguibile con il comando chmod:

```
chmod a+x schermoADestra
```

- Per eseguirlo ora (ammesso che sia nella nostra directory di lavoro) basterà scrivere al prompt:

```
./schermoADestra
```

- Se il file non è nella directory dove siamo, al posto di `./` dovremo scrivere il percorso per il file.

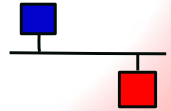


Richiamare il file senza il percorso

- Per richiamare il nostro file dobbiamo iserire il suo percorso perché non è in una delle directory che Linux usa per gli eseguibili.
- Quali sono queste directory? Sono elencate nella variabile d'ambiente **`$PATH`**
- Per visualizzarla useremo il comando echo:

  `echo $PATH`
- Che ci ritorna appunto l'elenco in questione:

  `/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games`
- Questo è un **elenco** di directory, separate da **`:`**
- Purtroppo nessuna di queste directory è scrivibile da noi, per cui non possiamo copiarci il nostro nuovo comando.

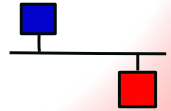


Aggiungere una directory per gli eseguibili

- Potremmo aggiungere una nostra directory agli eseguibili?
- Creiamo una directory bin nella nostra home:

  `mkdir ~/bin`
- Aggiungiamo questa directory alla variabile **`$PATH`** con il comando

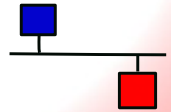
  `PATH=$PATH:~/bin`
- Notate che il **`PATH`** prima dell'uguale è senza il **$**, perché è la **variabile** (il contenitore) mentre quello a destra dell'uguale ha il **$**, perché ci serve il contenuto
- Occorre anche notare che la modifica che abbiamo fatto vale solo per il nostro utente, quindi è **locale** e che scomparirà al riavvio della macchina quini è **transitoria**



Rendere permanente il comando

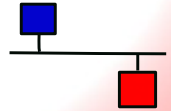
- Non possiamo rendere la modifica globale perché per farlo dovremmo essere root
- Possiamo però renderla permanente per noi.
- Basta aggiungere il comando **`PATH=$PATH:~/bin`** in fondo al file .bashrc nella nostra home directory
- Possiamo farlo con qualunque editor, anche grafico
- In alternativa potremmo usare il comando echo, con la **redirezione**:

  `echo PATH=$PATH:~/bin >> ~/.bashrc`
- Come sapete il comando **`echo`** produce come output la stringa passata per parametro
- Attenzione al **`>>`** che aggiunge in coda, perché se usate **`>`** sovrascrivete l'intero file



Redirezione

- La shell consente di scrivere l'uscita (quello che stampa a video) di un programma su di un file oppure di inviare ad un programma un file al posto di scrivere i dati da tastiera
- Questo si chiama **redirezione** e si ottiene inserendo i seguenti segni tra il comando ed il file:
  - **>** registra l'uscita su di un file, creandolo se manca o sovrascrivendolo se esiste
    ad esempio, **`ls > lista.txt`** salva l'elenco di file e directory della directory corrente nel file **`lista.txt`**
  - **>>** registra l'uscita su in coda ad un file, creandolo se manca
  - **<** invia il contenuto del file come ingresso al comando
    ad esempio **`cat < ~/.bashrc`** prende in ingresso il file **`.bashrc`** della home e lo stampa a video (cat però è in grado di prendere nomi di file di ingresso tra i parametri, quindi la redirezione è superflua in questo caso)

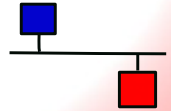


Pipe

- Un’altra caratteristica della shell è che è in grado di prendere l’uscita di un programma ed inviarla ad un altro; i due programmi vengono eseguiti simultaneamente.
- Per farlo si utilizza il segno **|** chiamato ***pipe***
- Ad esempio potremmo prendere la lista dei file della directory corrente e cercare in essa i soli file che contengono la parola README (maluscola) usando **`ls`** e **`grep`** nel seguente modo:

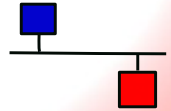
  ```
  ls | grep README
  ```

- Si possono in questo modo creare catene anche molto lunghe di comandi, collegandoli con delle ***pipe***



Espressioni regolari semplici

- Le ***espressioni regolari*** sono dei metodi per indicare una *famiglia di stringhe*
- Possono essere molto complesse ma nella loro forma più semplice usano due semplici caratteri per indicare:
  - ‘**.**’ un singolo carattere qualunque
  - ‘*****’ 0, 1 o più caratteri qualunque
- Questi caratteri possono essere messi ovunque in un nome

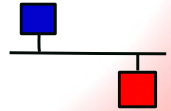


Esempi di espressioni regolari

- Per esempio, se cerchiamo tutte le immagini png in una directory possiamo scrivere
  `*.png`
- Ma anche, se cerchiamo tutti i file che contengano Stefano nel nome possiamo scrivere

  `*Stefano*`
- Possiamo anche mettere l'* per le directory. Possiamo ad esempio cercare

  `*/README.*`
- Lascio a voi indicare che cosa viene trovato

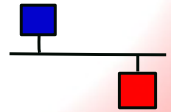


Espansione dei nomi

- La shell, quando trova un’espressione regolare cerca tutti i file che le corrispondono e crea una lista
- Se ad esempio noi scriviamo:

  `ls *.png`

- Non è il comando **`ls`** che cerca i file da elencare, ma direttamente la shell
- Troveremo questa caratteristica molto utile in seguito

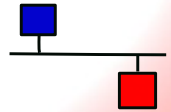


Il ciclo for

- La shell consente di usare costrutti di programmazione come il for, l'if ed il while sia da riga di comando sia negli script (i comandi refistrati nei file)
- Partiamo dal for:

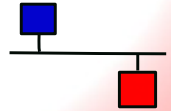
```
for <variabile> in <lista>
do
   <elenco azioni che usano i valori della variabile>
done
```

- Dove:
  - **<*variabile*>** deve essere il **nome** di una variabile, quindi **senza il $**
  - **<*lista*>** è una lista di valori che verranno assegnati a turno alla **<*variabile*>**



Il ciclo for

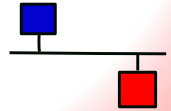
- Il più delle volte **`<lista>`** è o un'***espressione regolare*** che produce una lista di file o un **programma** che produce valori, uno per riga
- In ogni caso i dati forniti vengono 'tagliati' ad ogni carattere di spazio, per questo motivo è **meglio evitare di usare nomi di file contenti spazi**
- Un metodo alternativo è di **racchiudere tra doppi apici** la **variabile**, in questo modo ogni nome di file sarà racchiuso tra apici e gli spazi non daranno più fastidio



Facciamo un esempio

- Voglio raccogliere tutti i file ‘README’ della home directory e delle sottodirectory di primo livello in un unico file, con un’intestazione prima di ogni file costituita da una riga di asterischi, il nome del file ed una seconda riga di asterischi.
- Lo posso ottenere con il seguente script (che posso sia scrivere ‘al volo’ oppure inserire in un file con l’intestazione #!/bin/sh come spiegato precedentemente):

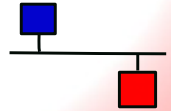
```
for r in README.* */README.*
do
    echo ****************************************** >> README.all
    echo $r >> README.all
    echo ****************************************** >> README.all
    cat “$r” >> README.all
done
```



Analizziamo lo script

- Cominciamo dalla prima riga:

  `for r in README.* */README.*`
- Per prima cosa, **`r`**, senza il **`$`**, è il nome della variabile dove verrà di volta in volta messo il percorso del file
- Poi, **`README.*`** e **`*/README.*`** sono due diverse espressioni regolari che verranno entrambe espanse dalla shell e costituiranno complessivamente la lista dei file.

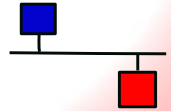


L’intestazione di ogni file

- Vediamo ora:

```
echo ********************************** >> README.all
echo $r >> README.all
echo ********************************** >> README.all
```

- Ci sono due cose da notare:
  - Tutti e tre gli **echo** sono rediretti con il **>>** che accoda la stringa prodotta al file di uscita, senza sovrascriverlo
  - Nella seconda riga viene usato **$r** per ottenere il percorso del file corrente, senza doppi apici perché tanto **echo** stampa tutte le parole che seguono.

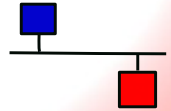


La copia del file

- Per finire:

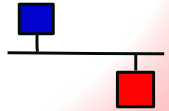
  `cat “$r” >> README.all`

- Il comando **`cat`** copia, uno dopo l’altro (concatena), i file passati come parametro sulla sua uscita
- In questo caso noi indichiamo un solo file, utilizzando sempre **`$r`**, ma questa volta lo mettiamo tra **doppi apici** per far capire che, anche se ci fossero spazi nel nome, è comunque un unico percorso.



Parametri

- Se noi mettiamo lo script in un file possiamo richiamarlo passandogli dei parametri sulla riga di comando
- Questi parametri sono inseriti in variabili con dei nomi standard:
- **`$0`** è il primo parametro, che di fatto è il nome dello script
- **`$1`** è il primo parametro
- **`$2`** il secondo
- **`$...`**
- **`$*`** contiene la lista di tutti i parametri da $1 in avanti

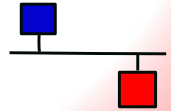


Verificare se c’è un parametro

- Per verificare se un parametro è presente occorre usare una struttura condizionale – un if
- La struttura è:

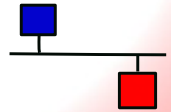
```
if <test>
then
        <comandi se vero>
else
        <comandi se falso>
fi
```

- **<test>** è un comando che ritorna 0 se il test ha successo diverso da 0 altrimenti
- **<comandi se vero>** Vengono eseguiti se il test ha avuto successo
- **<comandi se falso>** Vengono eseguiti se il test ha fallito
- **else** ed i relativi comandi possono anche non esserci



Cosa usare per il test

- Come test possiamo usare qualunque comando che ritorni un vero o falso. Ne vedremo
- Il comando più utile è **`[`**
- Dopo il simbolo di **`[`** si può indicare una o più **espressioni** commesse da **operatori logici** (che salto) e si termina con **`]`**
- I più comuni test sono:
  - **`-n STRINGA`** (di solito una variabile con il $) che è vera se la **`STRINGA` non è vuota** (ricordare di mettere **`STRINGA`** tra doppi apici altrimenti se è vuota si ottiene un errore)
  - **`-z STRINGA`** che è vera se **`STRINGA`** è **vuota** (ricordare di mettere **`STRINGA`** tra doppi apici altrimenti se è vuota si ottiene un errore)
  - **`STRING1 = STRING2`** che è vera se le due stringhe sono **uguali**
  - **`STRING1 != STRING2`** che è vera se le due stringhe sono **diverse**
  - **`INT1 (-eq, -ne, -gt, ge, -lt, -le) INT2`** vera se i due numeri interi sono (uguali, diversi) o se **`INT1`** è (maggiore, maggiore o uguale, minore, minore o uguale) di **`INT2`**
  - **`(-e , -f, -d, -h) FILE`** vera se il **`FILE`** (esiste, è un file normale, è una directory, è un link simbolico)

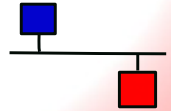


Veniamo al dunque

- Allora, come possiamo fare per verificare la presenza del parametro (il primo)?
- Possiamo scrivere il seguente script (che poi andrà riempito opportunamente con dei comandi)

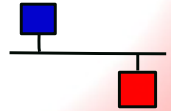
```
#!/bin/sh
if [ -z "$1" ]
then
        echo ERRORE: Nessun parametro
        exit 1
else
        echo Hai indicato il parametro $1
fi
```

- Nel caso il primo parametro sia vuoto (non ci sia), stamperemo un messaggio di errore e quindi termineremo con errore
- Il comando **`exit`** termina lo script ed il valore che indichiamo sarà il valore restituito, in questo caso 1 che indica errore (magari l'errore 1 di molti)



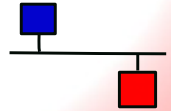
Mettiamo tutto insieme

- Voglio creare uno script che possa clonare gli schermi oppure disporli con il monitor esterno a destra o a sinistra.
  - Se lo chiamerò **senza parametri clonerà** gli schermi
  - Se passerò il **parametro `-r`** metterà il monitor esterno a **destra**
  - Se passerò il **parametro `-l`** metterà il monitor esterno a **sinistra**



Cominciamo

- Per cominciare, rivediamo i comandi da dare.
- So che lo schermo del portatile si chiama **`eDP-1`**, il monitor esterno **`HDMI-1-0`** e che hanno la stessa risoluzione
- I comandi saranno:
  - Per mettere lo schermo a destra:
    `xrandr --output HDMI-1-0 --right-of eDP-1`
  - Per mettere lo schermo a sinistra:
    `xrandr --output HDMI-1-0 --left-of eDP-1`
  - Per clonare gli schermi:
    `xrandr --output HDMI-1-0 --same-as eDP-1`
- Sono praticamente identici. Cambia solo il modo.



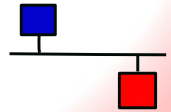
Impostiamo lo script

- Comiciamo a scrivere lo script usando delle variabili ed un po’ di logica per gestire i parametri:

```
POSITION='same-as'
if [ -n “$1” ]
then
        if [ $1 = '-l' ]
        then
                POSITION='left-of'
        fi
        if [ $1 = '-r' ]
        then
                POSITION='right-of'
        fi
fi

PRIMARY=eDP-1
SECONDARY=HDMI-1-0

xrandr --output $SECONDARY --$POSITION $PRIMARY
```

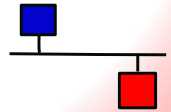


Rendiamo più generale lo script

- In questo script ho inserito i nomi del mio schermo interno e del mio monitor.
- Se lancio lo script su di un diverso PC o con un diverso schermo non funzionerà
- Cerchiamo allora di individuare i nomi degli schermi.
- Il comando **`xrandr`** stampa le informazioni che mi servono, ma anche moltissime di più.
- Per il monitor primario mi dice, oltre alle risoluzioni disponibili ed ai relativi modi:

  `eDP-1 connected primary 1920x1200+0+0 (normal left inverted right x axis y axis) 344mm x 215mm`
- Mentre per il secondario:

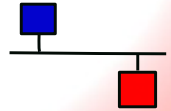
  `HDMI-1-0 connected 1920x1080+1920+0 (normal left inverted right x axis y axis) 480mm x 270mm`
- Inoltre indica una serie di monitor disconnessi.
- A noi però interessano solo quelli connessi, quindi dovremo cercare le righe con la parola *connected*



Estrarre righe

- Per fare questo potremo usare il comando **`grep`**
- A questo comando indicheremo una parola (o meglio un'espressione regolare) e lui estrarrà dal suo ingresso o dai file che gli indichiamo dopo l'espressione di ricerca tutte le righe che corrispondono a quell'espressione
- Ne nostro caso applicheremo l'espressione *connected* all'uscita del comando xrandr **--**query
- Per la precisione, useremo " connected" per evitare che selezioni anche i ***dis****connected*
- Quindi scriveremo:

  `xrandr | grep " connected"`
- Che ci restituirà l'elenco dei monitor connessi



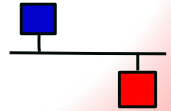
Primario e secondario

- A noi servono i nomi del monitor **primario** (quello interno) e del **secondario** (quello esterno, l'altro)
- Ovviamente il primario ha la parola primary nel nome, l'altro no.
- Per estrarre il primario manderemo l'uscita della ricerca di prima in un'altro grep che cercherà *primary*:

  ```
  xrandr | grep " connected" | grep primary
  ```

- Per trovare l'altro cercheremo la riga che non ha la parola primary (abbiamo solo due monitor, quindi se non è primario è l' unico esterno)
- Useremo il parametro -v per invertire la ricerca:

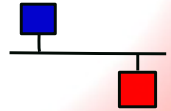
  ```
  xrandr | grep " connected" | grep -v primary
  ```



Estrarre il solo nome

- Ora abbiamo due righe che iniziano con il nome di un monitor
- Dobbiamo estrarre solo questo nome, da usare nei comandi successivi
- Ci sono (almeno) tre comandi utili a questo scopo:
  - **`cut`**
  - **`awk`**
  - **`sed`**
- L'ultimo (string editor) consente di applicare delle operazioni ad un file come se fosse un editor testuale tipo ed o vi e quindi è il più flessibile, quindi uso quello
- Il metodo usato è di sostituire con una stringa vuota dal primo spazio alla fine della riga, quindi userò l'espressione **`"s/ .*$//"`** - i doppi apici sono obbligatori perché l'espressione contiene degli spazi
- I comandi che uso sono quindi i seguenti:

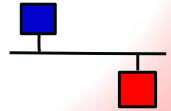
```
xrandr | grep " connected" | grep primary | sed -e "s/ .*$//"
xrandr | grep " connected" | grep -v primary | sed -e "s/ .*$//"
```



Assegniamo i nomi alle variabili

- Esistono due modi per usare l'output di un programma come stringa, per esempio per assegnarla ad una variabile:
- Possiamo racchiudere il comando tra aprici rovescati (` o **alt-'**)
- Possiamo usare **$(<comando>)** dove **<comando>** è ovviamente il nostro comando
- Scelgo la prima e quindi sostituisco l'assegnazione di **PRIMARY** e **SECONDARY** con le seguenti:

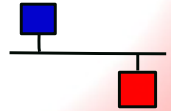
```
PRIMARY=`xrandr | grep " connected" | grep primary | sed -e "s/ .*$//"`
SECONDARY=`xrandr | grep " connected" | grep -v primary | sed -e "s/ .*$//"`
```



Lo script completo

```
#!/bin/bash
POSITION='same-as'
if [ x$1 != 'x' ]
then
        if [ $1 = '-l' ]
        then
                POSITION='left-of'
        fi
        if [ $1 = '-r' ]
        then
                POSITION='right-of'
        fi
fi
PRIMARY=`xrandr | grep " connected" | grep primary | sed -e "s/ .*$//"`
SECONDARY=`xrandr | grep " connected" | grep -v primary | sed -e "s/ .*$//"`

xrandr --output $SECONDARY --$POSITION $PRIMARY
```



Comandi citati

- Xrandr
- Echo
- Cat
- [
- Exit
- Sed (awk,cut)