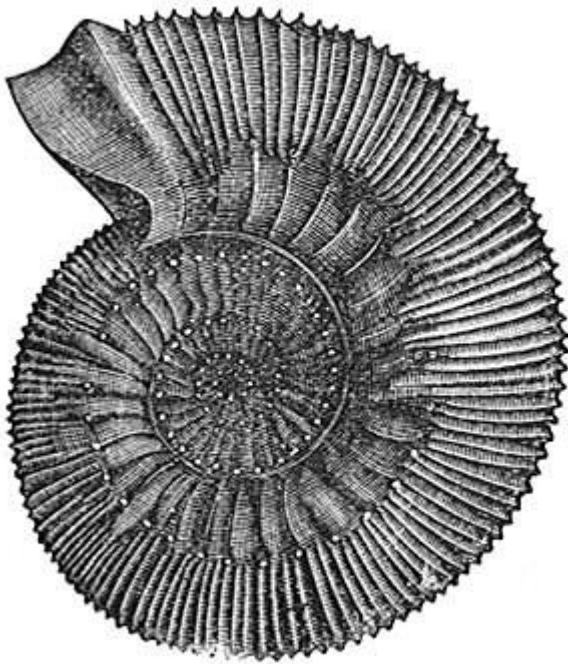


# Shell



La shell è quell'interfaccia con la quale è possibile interagire con il sistema operativo.

La shell è un programma che interpreta ed esegue i comandi letti dallo standard input o da un file.

# Tipi di shell

- sh - Bourne shell (su tutti i sistemi Unix)
- csh - C shell (utilizza costrutti derivati dal linguaggio C)
- bash - Bourne Again Shell ( è GPL, su tutti i sistemi Linux è un'estensione della sh)
- ksh - Korn shell (molto diffusa)
- tcsh - E' un'evoluzione della csh

# bash

## Bourne Again Shell (Linux, ma anche Unix e Windows)

Alcune sue caratteristiche:

- Possibilità di editare la command line
- Auto-completamento dei comandi
- Possibilità di definire alias
- Possibilità di avere la storia dei comandi eseguiti (History)
- Funzionalità di scripting, funzioni condizionali e di ciclo .
- Possibilità di definire funzioni
- Possibilità di gestire array indicizzati di dimensioni infinite
- Gestione e controllo dei job
- Espressioni aritmetiche
- Caratteri jolly (metacaratteri) nella gestione dei nomi di file

# Personalizzare la bash

File di configurazione (“init” o “rc” o “.”)

- `/etc/bashrc` (contiene gli alias e le funzioni valide per l'intero sistema)
- `/etc/profile` (file di inizializzazione generale per il sistema, eseguito per le shell di login)
- `~/.bashrc` (contiene gli alias e funzioni utente)
- `~/.bash_profile` (file di inizializzazione utente, eseguito per le shell di login)
- `~/.inputrc` (file di inizializzazione utente per definizione tasti)
- `~/.bash_logout` (file utente per le operazioni pre-logout)

# .bashrc

```
# .bashrc
# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
# User specific aliases and functions
#esempio alias
alias a='alias'
#creare un file solo per gli alias
if [ -f ${HOME}/.alias ]; then
    . ${HOME}/.alias
fi
```

# Comandi

- TAB: tasto importante nella bash, permette l'autocompletamento di comandi o di nomi di file e directory.
- FRECCIA SU - FRECCIA GIU: permettono di spostarsi nella history dei comandi digitati precedentemente nella shell.
- FRECCIA DESTRA - FRECCIA SINISTRA: permettono di muoversi all'interno della riga di comando corrente per modificarne il testo.
- CTRL-w: permette di cancellare all'indietro fino al primo carattere "spazio".
- CTRL-a o CTRL-e: permettono di andare rispettivamente ad inizio e fine riga.
- SHIFT+FRECCIA SU - SHIFT+FRECCIA GIU: permettono di visualizzare l'output della shell che ormai non è più visibile nella schermata corrente, hanno cioè funzione di scrolling riga per riga.
- SHIFT+PAGINA SU - SHIFT+PAGINA GIU: come le precedenti ma lo scrolling è a pagine.

# Ridirezione e pipe

- standard input (stdin o file descriptor 0)
- standard output (stdout o file descriptor 1)
- standard error (stderr o file descriptor 2)
- “|” o pipe prende l'output di un programma e lo passa in input ad un altro
- “>” o “>>” ridirigono lo standard output
- “<” ridirige lo standard input
- “2>” ridirige lo standard error

# Esempi

- `ls -laF > ls-out.txt`: l'output del comando “ls -laF” viene diretto sul file `ls-out.txt`. Notare che se il file esiste viene sovrascritto.
- `cat ls-out.txt | less`: il contenuto del file `ls-out.txt` viene stampato su stdout da “cat”, la pipe ridirige l'uscita al comando “less” che lo visualizza a “pagine”.
- `ps -edalf | grep $USER`: permette di visualizzare tutti i processi dell'utente che esegue il comando.
- `make > outerr.txt 2>&1`: ridirige sia lo stdout che lo stderr del comando “make” sul file “outerr.txt”



# Regular Expressions

- Il termine espressione regolare è stato coniato nei primi anni '50 dal matematico Stephen Kleene, nel corso dello sviluppo della teoria formale dei linguaggi (vengono definite come metodi per esprimere la concatenazione di elementi di un linguaggio).
- Nell'accezione pratica le espressioni regolari sono un potente linguaggio simbolico che permette di rappresentare le caratteristiche di una stringa o di un insieme di stringhe.
- L'uso delle R.E. facilita la scrittura di qualunque applicazione nella quale sia necessario analizzare stringhe.
- Qualunque utente di Dos o Unix ha già usato una forma di R.E.:  
> dir \*.exe
- Esistono strumenti, che sfruttano al meglio le potenzialità delle R.E.

# Metacaratteri

- . identifica un singolo carattere
- ^ identifica l'inizio riga
- \ neutralizza il significato del metacarattere seguente
- \$ identifica la fine della stringa
- [] identifica qualsiasi carattere indicato tra parentesi
- [^] identifica tutti i caratteri non specificati nell'insieme
- [0-9] identifica ogni carattere compreso nell'intervallo (numeri o lettere)

- ? zero o una volta
- + una o più volte
- \* zero o più volte
- | uno o l'altro elemento prima e dopo il segno |

# Esempi

- `egrep -i '(string1|string2)' file` : Ricerca e visualizza in un file string1 e string2
- `grep '^1' list.txt` : Ricerca in list.txt le righe che iniziano con 1
- `egrep '^2[234]' list.txt` : Ricerca le righe che iniziano con 22,23,24
- `grep '^Linux$' list.txt` : Ricerca le righe che contengono SOLO la parola Linux
- `grep -c '^$' list.txt` : Visualizza il numero di righe vuote in list.txt
- `grep '^[^0-9]' list.txt` : Visualizza le righe che NON iniziano con un numero
- `grep '\<[Ll]inux\>' list.txt` : Visualizza le righe che contengono la parola singola Linux o linux, ma non visualizza quelle con, per esempio, LinuxOS

# Scripts

- Insieme di comandi che vengono interpretati (in questo caso dalla shell)
- Permettono di personalizzare l'ambiente (variabili e comandi nuovi)
- Sono alla base di qualunque macchina Linux

# Costrutti

<b>if</b>	<i>if</i> [espressione] <i>then</i> comando1 <i>else</i> comando2 <i>fi</i>
<b>for</b>	<i>for</i> variabile <i>in</i> lista <i>do</i> comando <i>done</i>
<b>while</b>	<i>while</i> [espressione] <i>do</i> comando <i>done</i>
<b>case</b>	<i>case</i> stringa <i>in</i> pattern1) comando1 ;; pattern2) comando2 ;; <i>esac</i>

# test.sh

```
#!/bin/sh
#test.sh
#stampa i parametri passati alla riga di comando

echo "Numero parametri = $#
echo "Nome del Programma = "$0
for param in $*
do
    echo "Parametro ${param}"
done
```

# test.bash

```
#!/bin/bash
# test.bash stesso programma ma per bash
echo "Numero parametri = "$#
echo "Nome del Programma = "$0
n_param="$#"
for((param=1; param<=$n_param; param++)); do
    echo "Parametro ${param}/${n_param} $1"
    shift
    echo "Nuovo numero di parametri dopo lo shift = "$#
done
```



# test1.bash

```
#!/bin/bash
#stesso esempio ma con uso di funzione ricorsiva
stampa_parametro()
{
    if [ "$#" -eq "0" ]
    then
        return
    else
        echo "Parametro $1"
        shift
        stampa_parametro $*
    fi
}
echo "Numero parametri = "$#
echo "Nome del Programma = "$0
stampa_parametro $*
```