LA SHELL (*bash*, *powershell*, *cmd*)

La **shell** è **un programma che gestisce la comunicazione tra l'utente e il sistema operativo** (è un'interfaccia!).

- Vero sopratutto in Unix/Linux/DOS prima dell'avvento delle interfacce grafiche.

- **È un interprete dei comandi:** un vero e proprio ambiente di programmazione con cui realizzare script per automatizzare operazioni complesse e/o ripetitive.

- Si anche il prompt dei comandi (`cmd.exe`) è una shell grazie al **batch scripting** e da qualche anno al **VBscript**.

Tipi di shell in Unix/Linux e Mac OS (X)

- **Bourne Shell (sh)**: shell originale di Unix. Scritta da Steve Bourne, presente su tutti i sistemi Unix.
- **C Shell (csh):** scritta all'università di Berkley California. Usa un linguaggio di scripting simile al C.
- **TC Shell (tcsh)**: evoluzione di csh.
- **Korn shell (ksh)**: scritta da David Korn è un miscuglio fra csh, tcsh e sh.
- **Bourne Again Shell (bash)**: Scritta da FSF come shell free software. **Shell di default di Linux**, usa un linguaggio compatibile con sh.
- Dash: deriva da Ash shell del sistema NetBSD.
- Rbash: restricted bash shell, utilizzata nei sistemi di recovery.
- Per vedere quali shell sono installate in un sistema si guarda il file `/etc/shells`.

Tipi di shell in Windows

- **Prompt dei comandi (cmd.exe)**: storico porting del vecchio DOS. Permette l'esecuzione di comandi di sistema e la creazione di script in *batch* (.bat). Da qualche anno permette l'esecuzione di *Vbscript*.

- **PowerShell**: noto inizialmente come **Microsoft Shell** o **MSH** (o col nome in codice **Monad**) e poi come **Windows Shell**, è una shell caratterizzata dall'interfaccia a riga di comando (CLI) e da un linguaggio di scripting, sviluppata da Microsoft. *È basata sulla programmazione a oggetti e sul framework Microsoft .NET*. Permette l'esecuzione di script Vbs.

BASH

THE BOURNE-AGAIN SHELL

BASH - Login

- Cosa succede quando aprite un terminale o vi loggate in modalità testuale?
- Il sistema lancia la vostra shell predefinita, molto probabilmente una **bash**, che:
  1. Cerca il file "`/etc/profile`" e se esiste, lo elabora.
  2. Cerca il file "`$HOME/.bash_profile`" e se esiste, lo elabora, altrimenti
     - cerca il file "`$HOME/.bash_login`" e se esiste, lo elabora, altrimenti
     - prova con "`$HOME/.profile`", se esiste, lo elabora.
  3. Cerca il file "`$HOME/.bashrc`" e se esiste, lo elabora.

BASH – file di configurazione

- `.profile` è un file di configurazione generale che contiene la definizione delle variabili d'ambiente.
- `.bashrc` contiene comandi, alias e dichiarazioni di ulteriori variabili.
- I file `/etc/profile`, `$HOME/.bash_profile`, `$HOME/.bash_profile`,`$HOME/.profile` contengono le configurazioni per una shell di login (1).
- `$HOME/.bashrc` contiene informazioni per quelle successive al login (2).
- In pratica i file (1) vengono letti ed eseguiti solo per la prima shell che si apre (al login), e non per le successive le quali eseguono ogni volta i file (2).

BASH - history

- La shell è in grado di ricordare i comandi immessi dall'utente. Normalmente sono salvati nel file `~/.bash_history`
- Per vederne il contentuto dovete usare il comando `history`.
- Tramite le variabili di ambiente `HISTSIZE`, `HISTFILE`, `HISTFILESIZE`, si può dire al sistema come gestire la propria history.
- Esempio :

```
# export HISTFILE=/dev/null
```

  - Fa puntare a **`null`** il file `history`... in questo caso non viene salvato alcun elenco dei comandi per la shell in uso.

BASH – caratteri speciali

Come ogni interprete/compilatore, anche la bash ha la sua sintassi, a partire da dei caratteri speciali:

- `;` separatore comandi.
- `&` esecuzione in background.
- `()` raggruppa i comandi.
- `{}` blocco di comandi.
- `|` pipe.
- `>< &` reindirizzamento.
- `*?[]~!` Caratteri jolly.
- `#` indica un commento! Attenti a non confonderlo con il prompt (#, >, ~) ...

BASH - Esempi

- `comando&` # lancia un comando in background.
- `comando1; comando2` # lancia comando1 e poi comando2
- `(comando1; comando2)` # come sopra ma raggruppa i risultati in un unico blocco.
- `comando1 | comando2` # esegue comando2 usando come input il risultato di comando1.
- ~~`comando1 ‘comando2’` # esegue comando1 passando come parametro il risultato di comando2~~.
- `comando1 $(comando2)` # esegue comando1 passando come parametro il risultato di comando2 .
- `comando1 || comando2` # esegue comando1 OR comando2.
- `comando1 && comando2 # esegue comando1 AND comando2.`

Che è sta roba?

```
(sleep 1, /.soluzione)& ./cypto -i /tmp/PipeIn -o /tmp/PipeOut
```

BASH – Esempi

- `(date; who; pwd) > logfile`
- `date; who; pwd > logfile`
  - I due comandi sopra hanno esiti diversi…
- `sort <miofile>.c | pr –n -h “Mio programma” | lp`
  - Ordina il file prova .c, lo suddivide in pagine aggiungendo numeri di riga e header, e lo manda in stampa.
- `nano $(grep –l ifdef *.c)`
  - Apre com emacs il primo file .c con una riga ifdef.
- `grep –E “(yes|no)” list.txt`
  - Cerca le occorrenze di yes o no nel file list.txt.
- `grep aaa file.txt && lp file.txt`
  - Cerca la stringa aaa in `file.txt`. Se la trova stampa `file.txt`.
- `grep bbb file.txt || echo “non esiste”`
  - Cerca la stringa bbb in `file.txt`. Se non la trova stampa “non esiste”.

BASH - Reindirizzamento

- `0` stdin tastiera.
- `1` stdout terminale.
- `2` stderr terminale.
- `< file` legge input da file.
- `> file` scrive output su file creando/sovrascrivendo.
- `>> file` scrive output su file creando il file o accodando il contenuto se esiste.
- `<< text` legge `stdin` fino a trovare una riga uguale a text.
- `>&n` Duplica lo standard output nel file descriptor n.
- `<&n` Duplica lo standard input dal file descriptor n.
- `&>file` Dirige lo standard output e lo standard error in un file.

BASH - Comandi

- Oltre ai comandi di sistema (`ls, df, cd, rm`, ecc...) potete usare dei comandi propri della bash nei vostri `script`.
- Uno `script` è un file di testo scritto in codice shell con permessi di esecuzione:

```
# touch mioscript.sh
# chmod +x mioscript.sh
# nano mioscript.sh
# ./mioscript.sh
```

- `#!shell` indica la shell usata nel vs script. **Deve essere la prima riga!!** Ad esempio:

```
#!/bin/bash [-e: stop al primo errore, -x:debug]
```

- Per includere un file da eseguire potete usare le direttive

```
. <nomefile>
source <nomefile>
```

BASH - Variabili

prompt

- Il valore di una variabile si assegna tramite l'operatore `=`

```
# MYVAR="PIPPO" # occhio agli spazi!!!
```

- Il carattere `$` come prefisso per indicarne il valore.

```
# echo $MYVAR
  PIPPO
```

- Generalmente le variabili sono visibili solo all'interno della shell stessa. Per passare variabili ad altri programmi chiamati all'interno della shell si deve utilizzare il comando `export`.
- Se racchiuse da `[ ]` sono considerate come array:

```
A[0]=1
A[1]=2
A[n]=100
```

oppure `A=(0 1 2 … 100)`

per estrarre il valore: `# echo ${A[1]}`

2

BASH – variabili speciali

- `$0` indica il nome del programma.
- `${n}` indica i singoli argomenti della riga di comando parametri di input `n=1..9`.
- `$#` indica il numero argomenti riga di comando.
- `$*` indica tutti gli argomenti della riga di comando.
- `$$` indica il `pid` del processo che esegue lo script.
- `$?` restituisce lo `status code` dell'ultimo comando eseguito.
- `$!` contiene il `pid` dell'ultimo processo mandato in background.

BASH - if

```
if <condizione> then <comandi>
else <comandi>
fi
```

La clausola `else if` diventa **`elif`.**

La condizione posta sempre tra [.] è un test che da risultato binario (true|false) e corrisponde al comando `test`. Provate da terminale a scrivere:

```
# test 1 -eq 1 && echo "sono uguali" :)
```

```
if [ $# -eq 0 ]; then
 echo “Usa $0 nomefile”;
 exit 1;
fi
```

```
if [ -f /etc/hosts ]; then
      echo “Esiste!”;
      cat /etc/hosts;
fi
```

BASH – if operatori di confronto

- Questi sono solo alcuni degli operatori di confronto che possono essere utilizzati nei costrutti del linguaggio bash come `if`, `while` e `until`. Altri li potete ottenere digitando `man test`.

test operators

comparison operator	description
=, !=, \<, \>	compares two **string** variables
-z, -n	tests if a string is empty (zero-length) or not empty (nonzero-length)
-lt, -le, -eq, -gt, -ge, -ne	compares **numbers**; equivalent to Java's <, <=, ==, >, >=, !=
-e, -f, -d	tests whether a given file or directory exists
-r, -w, -x	tests whether a file exists and is readable/writable/executable

```
if [ $USER = "husky14" ]; then
    echo 'Woof! Go Huskies!'
fi

LOGINS=`w -h | wc -l`
if [ $LOGINS -gt 10 ]; then
    echo 'attu is very busy right now!'
fi
```

*Note: `man test` will show other operators.

More if testing

compound comparison operators	description
`if [ expr1 -a expr2 ]; then ...` `if [ test1 ] && [ test2 ]; then ...`	and
`if [ expr1 -o expr2 ]; then ...` `if [ test1 ] || [ test2 ]; then ...`	or
`if [ ! expr ]; then ...`	not

```
# alert user if running >= 10 processes when
# attu is busy (>= 5 users logged in)
LOGINS=`w | wc -l`
PROCESSES=`ps -u $USER | wc -l`
if [ $LOGINS -gt 5 -a $PROCESSES -gt 10 ]; then
    echo "Quit hogging the server!"
fi
```

BASH – flussi: for

```
for i in <elenco>; do
  comandi
done
```

- Assegna ogni parola dell'elenco a `i` e per ogni parola assegnata esegue i comandi. Se l'argomento elenco viene omesso, allora vengono utilizzati i parametri posizionali `$@` (lista dei parametri in input `$0, ... , $9`).
- Esempio

```
for i in $(ls -1);
do
   du -hs $i 2> log.txt
done
```

BASH – flussi: while, until

```
while condizione;
do
   comandi;
done
```

- Esempio:

```
a=$1
while [ $a -gt 0 ]
do
   echo $a
   a=$a-1;
done
```

Finchè `$a > 0`

```
until condizione;
do
   comandi;
done
```

- Esempio:

```
a=$1
until [ $a -gt 0 ]
do
   echo $a;
   a=$a-1;
done
```

Finchè `$a > 0`

BASH – flussi: case

```
case $var in
valore1)
   comandi;;
valore2)
   comandi;;
. . . . . .
*)
   comandi;;
esac
```

Esempio

```
x=5
case $x in
1)
   echo "x=1"
   ;;
5)
   echo "x=5"
   ;;
. . . . . .
*)
   echo "x=$x"
   ;;
esac
```

Bash - select

`select` `v [ in items; ]` **`do`** `comandi;` **`done`**

- Permette all'utente di selezionare una delle parole elencate in `items` stampandole sullo standard output, associando ognuna ad un numero (**sostanzialmente crea un elenco numerato**). Se la lista `items` non viene fornita viene utilizzata la lista dei parametri in input `$0, ... , $9`. Dopo la stampa viene visualizzato sullo schermo un prompt e viene acquista la risposta dell'utente.

`seq` `[OPTION]... FIRST INCREMENT LAST`

- Stampa i numeri da `FIRST` a `LAST` incrementando di `INCREMENT`.
- Esempio: `$ seq 1 4`

```
1
2
3
4
```

Esempio di script in BASH

```
#!/bin/bash
# Sintassi: rdesklab.sh lab fila,..,fila user password
RDESKCMD="rdesktop-vrdp"
MAX_FILA=8
DOMINIO=""
PREFIX="lab"
RES="1024x768"

LAB=$1
utente=$3
password=$4

#### vi risparmio i controlli sull'input #######
if [ $1 -lt 10 ]; then
  LAB="0$1"
fi

FILE="$(echo "$2" | cut -d , -s -f 1-10 --output-delimiter ' ')"

if [ -z $FILE ]; then
 FILE=$2
fi

for j in $FILE; do
  if [ $j -lt 10 ]; then
    j="0$j"
  fi
  for i in $(seq 1 $MAX_FILA); do
    if [ $i -lt 10 ]; then
      i="0$i"
    fi
   (($RDESKCMD -u $utente -p $password $PREFIX$LAB$j$i -g$RES)&) ;
 done;
done
```

Dal manuale di `cut`:

`-d`, `--delimiter=DELIM`
use DELIM instead of TAB for field delimiter

`-s`, `--only-delimited`
do not print lines not containing delimiters

`-f`, `--fields=LIST`
select only these fields; also print any line that contains no delimiter character, unless the -s option is specified

`--output-delimiter=STRING`
use STRING as the output delimiter the default is to use the input delimiter

BASH variabile di sistema IFS

- **IFS**, ovvero l'**Internal Field Separator**, *rappresenta la sequenza di caratteri che separa i diversi parametri fra di loro*. Il valore di default è **SPACE TAB NEWLINE (`' \t\n'`).**
- Può essere modificata ed in tal caso è facile mantenerne una copia.
- Quando si manipola la variabile **IFS** è importante considerare che questa potrebbe contenere spazi, newline e altri caratteri "incontrollabili". È perciò una buona pratica quella di usare le virgolette nella loro gestione, cioè usare `OLDIFS="$IFS"` invece che `OLDIFS=$IFS`.
- Se manipolate **IFS** dentro un vostro script, ricordatevi sempre di riassegnarle il valore di default prima di terminare il programma!!! Altri script eseguiti successivamente potrebbero fa riferimento al valore precedente di **IFS**!.

BASH variabile di sistema IFS

```
#!/bin/bash
OLDIFS=”$IFS”;
IFS=$'\t\n'; # default IFS=$' \t\n'
for i in $(ls -1); do
  du -hs $i;
done;
$IFS=”$OLDIFS”
```

Altri linguaggi di scripting - Perl

- **Perl** è un linguaggio di programmazione ad alto livello, dinamico, procedurale e interpretato, creato nel 1987 da Larry Wall. Possiede un singolare insieme di funzionalità ereditate da `C`, `sh`, `awk`, `sed` e in diversa misura da molti altri linguaggi di programmazione, compresi alcuni linguaggi funzionali.
- Molto noto come linguaggio per lo sviluppo di **CGI** (agli arbori del web dinamico), è stato creato inizialmente come ausilio ai sistemisti, per la manipolazione di testo e file.
- Grazie alla struttura modulare è stato esteso negli anni con librerie per l'accesso ai database, per la manipolazione di immagini ecc…
- Da anni (circa 14) si attende la versione 6, anche se praticamente abbandonato resta nel cuore di molti sistemisti. Molti script utilizzati qui al DAIS sono scritti in Perl (ad esempio quelli per la creazione delle login).

Altri linguaggi di scripting - Perl

```
#!/usr/bin/perl

use DBI;

$database = "database";

$username = "username";

$password = "password";

$hostname = "hostname";



$db = DBI->connect("DBI:mysql:$database:$hostname", $username, $password);

$query = $db->prepare("select cognome,nome,matricola,codice,estratto,email,lingua,datetime from matricola
where estratto='0' order by datetime;");

$query->execute;

$conta=0;

while(@array = $query->fetchrow_array) {

        ($cognome,$nome,$matricola,$codice,$estratto,$email,$lingua,$timestamp) = @array;

        print("$cognome,$nome,$matricola,$codice,$lingua; $timestamp\n");

        $conta++

};

print("totale=".$conta."\n");

$query->finish;
```

Altri linguaggi di scripting - Python

- Ideato da Guido Van Rossum nel 2001, è un linguaggio di programmazione ad alto livello, orientato agli oggetti, utilizzato per sviluppare applicazioni distribuite, **scripting**, computazione numerica.
- Gira su Windows, Linux/Unix, Mac OS X, OS/2, Amiga, palmari Palm e cellulari Nokia..
- Viene usato spesso per gestire il Backend dei portali Web.
- **Linguaggio semplice da imparare, pieno di librerie e moduli in continua espansione**.

Powershell

- **Windows PowerShell® (nome in codice Monad)** è una shell basata sulla programmazione ad oggetti e sul framework MS .NET: sostanzialmente é un linguaggio di scripting progettato specificamente per l'amministrazione del sistema.
- PowerShell è la combinazione di compiti complessi e di una serie di componenti, le **cmdlets** (**command lets**, serie di comandi) che sono classi .NET.
- Linguaggio di scripting simile al C#.
- Completamento automatico dei comandi estendibile dall'utente.
- Permette di accedere e manipolare non solo il file system, ma anche altre strutture dati gerarchiche, come i registri di Windows.

- Per approfondire

`https://technet.microsoft.com/it-it/library/bb978526.aspx`

Powershell

- Sono presenti, naturalmente, tutte le caratteristiche che ci si aspetta di trovare in un linguaggio di scripting, quali **l'iterazione** (`for/foreach/while`), i **costrutti condizionali** (`if/switch`), i **campi di visibilità delle variabili** (`global/script/local/private`) e la possibilità di definire funzioni.

- **La differenza fondamentale tra l'approccio Unix e quello di PowerShell risiede nel fatto che piuttosto che creare una "pipeline" basata su input ed output testuali, PowerShell fa passare i dati da una cmdlet all'altra come oggetti (dati dotati di una struttura ben precisa)**.

- Per approfondire

  `https://technet.microsoft.com/it-it/library/bb978526.aspx`

Powershell - Esempi

Ecco alcuni esempi presi da:
https://docs.microsoft.com/it-it/powershell/scripting/samples/collecting-information-about-computers?view=powershell-7

- Elenco delle informazioni sul BIOS

```
Get-CimInstance -ClassName Win32_BIOS
```

- Elenco delle informazioni sul processore

```
Get-CimInstance -ClassName Win32_Processor
| Select-Object -ExcludeProperty "CIM*"
```

- Elenco degli hotfix installati

```
Get-CimInstance -ClassName
Win32_QuickFixEngineering
```

Powershell - Esempi

- Elenco di informazioni sulla versione del sistema operativo

```
Get-CimInstance -ClassName
Win32_OperatingSystem |
Select-Object -Property
BuildNumber,BuildType,OSType,ServicePackMaj
orVersion,ServicePackMinorVersion
```

- Elenco di utenti locali e proprietario

```
Get-CimInstance -ClassName
Win32_OperatingSystem | Select-Object -
Property *user*
```

- Recupero dello spazio su disco disponibile

```
Get-CimInstance -ClassName
Win32_LogicalDisk -Filter "DriveType=3"
```

Prompt dei comandi - Batch

- Il **prompt dei comandi** era la shell di default di Windows fino all'arrivo di Powershell.
- Il "linguaggio" principe per la realizzazione di script per il DOS e il prompt dei comandi era **Batch**. Ora si usa **Vbscript**.
- Gli **Script Batch** nascono con il debutto delle prime versioni di DOS/Windows.
- Di solito un file batch, ha estensione ".BAT" ed è apprezzato per i seguenti motivi:
  - Risparmia il lavoro di digitazione dei comandi DOS.
  - Consente di eseguire velocemente operazioni ridondanti.

Batch

- Un file batch **non ha una sintassi vera e propria, e usa i comandi del DOS (prompt dei comandi)**. Per questo motivo non può essere definito un vero e proprio linguaggio di programmazione .

- Come ottenere informazioni riguardo ai comandi?

  Basta andare sul prompt (`Start → Esegui → cmd`) e digitare questa sintassi:

  `<comando>/?`

  Ad esempio: `IF /?`

Batch - istruzioni

- **`echo`**: ha due principali funzioni: quella di mostrare o nascondere il percorso in cui si sta lavorando, l'altra è quella di mostrare semplici messaggi a video.
- **`@echo on`**: Questa istruzione ci permette di lasciare acceso (ON) l'echo.
- **`@echo off`**: Questa istruzione è il contrario di quella precedente, l'echo è spento (OFF).
- **`echo <Messaggio>`**: Dove messaggio è il testo da stampare. Visualizza il testo sullo schermo.

Batch - istruzioni

- **`pause`**: Mette in pausa il programma visualizzando "Premere un tasto per continuare".
- **`type <file>`**:stampa il contenuto di file a video, con “`| more` “ ottenete uno scrolling primitivo

```
type divinacommedia.txt | more
```

- **Leggere dalla riga di comando**: gli argomenti sono idenficati da `%1, %2, %N`:

```
echo Hai inserito '%1' dalla riga di comando.
```

Batch - istruzioni

- L'istruzione `IF` fa eseguire operazioni condizionali al programma batch. Ogni `IF` restituisce un `Errorlevel`, che ha risultati numerici a seconda se l'operazione è andata a buon fine o no.

- L'uguaglianza tra due stringhe (quali gli argomenti) si esprime con "`stringa1==stringa2`".
  - `IF %1==ciao echo Salve Padrone`
  - `IF %1==grazie echo Prego`

Batch – istruzioni

- Impostare una variabile

```
set nomevariabile=valorevariabile
```

- Estrarre il valore

```
%nomevariabile%
```

- I commenti si fanno con `REM`
- Esempio:

```
rem prova variabili
@echo off
set myvar=fabrizio
echo %myvar%
```

Batch - istruzioni

- `EXIST <path/file>` ritorna 1 se esiste un file

```
@echo off
if EXIST %1 echo %1 esiste
if NOT EXIST %1 echo %1 non esiste
```

- `FOR`: permette di gestire in modo elementare dei cicli

```
for %%variabile in (files) do istruzione
```

Esempio

```
 @echo off
if not exist %1 exit
for %%f in (%1) do goto dsfiles
:dsfiles
echo File %%f :
echo.
type %%f
echo.
```

Batch – comandi

- Avete notato l'uso del **`goto`**? Il salto verso una etichetta? Batch **non è strutturato** come non lo è il Basic.
- Attenzione a non scrivere script incompresibili…. (di solito sono già ostici).

- Altri comandi: `sort, choice, find.`

- Ovviamente potete eseguire comandi esterni.

Batch - esempio

```
rem copia un file nella home di tutti gli utenti in %listautenti%
@echo off
set CurrentDirectory=%cd%
set ProfileDir="d:\utenti\home\"
set dacopiare=%1
set listautenti=%2
FOR /F %%i IN (%CurrentDirectory%\%listautenti%) DO call :RunScript %%i
goto end
:RunScript
@echo off
if not exist %ProfileDir%%1\. goto end
copy %dacopiare% %ProfileDir%%1\.

@echo off
goto End
:End
REM ==
```

Microsoft

VBScript

VB Script

- **VBScript** (abbreviazione di **Microsoft's Visual Basic Scripting Edition**) è un sottoinsieme di **Visual Basic** utilizzato in *Windows Script Host* come linguaggio di scripting.

- È anche usato come sostituto, integrazione o appoggio per i file batch di MS-DOS o per meglio dire, della interfaccia da linea di comando di Windows.

VB Script - variabili

- Dichiarare una variabile con il costrutto "`Dim`":

```
Option Explicit 'obbliga la dichiatazione delle variabili
Dim a,b
Dim c(3) 'array
a="ciao"
b=1
c(0)=1
c(1)=2
c(3)=3
wscript.echo a
wscript.echo b
wscript.echo c(2)
> ciao
> 1
> 2
```

- Non è necessario dichiarare le variabili prima di usarle ma... Se scrivete un programma vbs lungo usate l'opzione explicit in cima al programma, vi obbliga a dichiarare tutte le variabili!!!

VB Script - operatori

- **Operatori Matematici:**
  - **+** (Addizione).
  - **-** (Sottrazione).
  - ***** (Moltiplicazione).
  - **/** (Divisione).
  - **^** (Elevamento a potenza) .
- **Operatori di Confronto:**
  - **=** (Uguaglianza).
  - **>** (Maggiore).
  - **<** (Minore).
  - **>=** (Maggiore o uguale).
  - **<=** (Minore o uguale).
  - **<>** (Diverso).

VB script - operatori

- **Operatori Logici (o Booleani):**
  - **`AND`** (congiunzione).
  - **`OR`** (disgiunzione).
  - **`NOT`** (negazione).
- **Operatori su Stringhe:**
  - **`&`** (Unione o Concatenamento)
  - **`+`** (Unione o Concatenamento)

VB Script - `if`

SINTASSI:

```
if <condizione> then 'se si avvera la condizione, allora
<istruzioni>
else                 'altrimenti
<istruzioni>
end if
```

VB Script – select case

SINTASSI:

```
select case (<variabile>)
case <valore> 'Se la variabile assume il valore x
<istruzioni>
case <valore> 'Se la variabile assume il valore x
<istruzioni>
….
case <valore>,…,<valore>
<istruzioni>
case else
<istruzioni>
end select
```

VB Script - `for`

SINTASSI:

```
for <variabile> = <valore1> to <valore2> step
<valore3>

'La variabile specificata che vale valore1 viene
'incrementata di valore3 fino a valore2
'<istruzioni>

next
```

VB Script – `for each`

SINTASSI:

```
for each <variabile> in <vettore>

'Variabile assumerà il valore di ciascun
'elemento di vettore ed eseguirà delle
'istruzioni per ciascuno di essi <istruzioni>

Next
```

VB Script – do until, loop until

SINTASSI:

```
do until <condizione>
 <istruzioni>
loop
'Esegue fino a quando si avvera la condizione
```

SINTASSI:

```
do
<istruzioni>
loop until <condizione>
'Esegue prima le istruzioni e poi verifica la condizione.
```

VB Script - Funzioni di Input Output

- `wscript.echo`: stampa una stringa a video nel prompt dei comandi.
- `msgbox`: apre una finestra grafica con un messaggio definito dal programmatore.
- `inputbox`: legge un dato in input tramite una interfaccia grafica.

VBScript - Oggetti

SINTASSI:

```
set <variabile> = CreateObject(<oggetto>)
```

Esempio:

```
Option Explicit
dim WshShell 'Variabile oggetto
set WshShell = CreateObject("wscript.Shell") 'Oggetto shell
wshShell.Exec("calc") 'Chiama la calcolatrice di Windows
```

VBScript – Esempio

- Cambia il percorso di profilo e home per un elenco di utenti preso da file

```
On Error Resume Next


Dim objUser

Set fso = CreateObject("Scripting.FileSystemObject")

Set ts = fso.OpenTextFile("elencoutenti.txt",1)

strComputer = "."


do while not ts.atEndOfStream

   Utente = ts.ReadLine

   Set objUser = GetObject("LDAP://cn=" & Utente & ", cn=Users, dc=win, dc=dsi, dc=unive, dc=it")

   objUser.Put "profilePath", "\\broot.dsi.unive.it\Profili\" & Utente

   objUser.Put "scriptPath", "logon.bat"

   objUser.Put "homeDirectory", "\\broot.dsi.unive.it\Home\" & Utente

   objUser.Put "homeDrive", "Z:"

   objUser.SetInfo

Loop


ts.Close
```