



Introduzione al linguaggio di Scripting del KVIrc

Start:

Questo vuol essere una sorta di Start alla programmazione, un sommario delle funzioni, dei comandi e dei costrutti più importanti.

Tradotto e adattato dal manuale originale (quindi tnx to Pragma =D).

Indice:

+Variabili:

- Globali
- Locali
- Array
- Dizionario

+Operatori

- Assegnamento
- Binding
- Operatori aritmetici con se stessi
- Incremento e decremento
- Concatenazione delle stringhe

+Cicli

- while() {}
- for(;;) {}
- foreach(,) {}

+Condizioni di controllo

- if/else
- switch/case()

Variabili

Variabili Globali:

Il nome di una variabile globale è definito tramite il simbolo di percentuale (%), seguito da una lettera maiuscola dalla A alla Z(*), e seguito da una serie di lettere (minuscole o maiuscole) o di numeri

o simboli (dalla 'a' alla 'z', dallo '0' a '9', '.', '_').

Ad esempio:

```
"%INDICE", "%Mio_nick", "%Indice", "%Articolo" and "%numero_articolo",  
una variabile globale esiste per l'intera applicazione
```

Provate a fare:

```
/%Hello = "Hello world!"
```

adesso da qualsiasi finestra, o in qualsiasi vostro script, la variabile %Hello esisterà e avrà come contenuto: "Hello world!".

Provate in qualsiasi finestra:

```
/echo %Hello
```

così ve ne renderete conto.

(*) con l'avvento del nuovo parser (disponibile entro la fine dell'anno nella prossima release stabile) le variabili globali dovranno essere dichiarate con il comando `global: global %var %var2 %var...`. Il vecchio metodo rimarrà per compatibilità ma sarà deprecato come uso.

Variabili Locali:

Il nome di una variabile locale è definito tramite il simbolo di percentuale (%), seguito da una lettera minuscola dalla a alla z (*), e seguito da una serie di lettere di numeri o simboli ('.', '_').

Ad esempio:

```
"%index", "%my_nickname", "%foo", "%bAR1" and "%foo.BAR"
```

Una variabile locale è visibile all'interno del blocco di istruzioni in cui viene creata, ad esempio, in un alias, o all'interno del codice presente in un evento.

Provate a fare:

```
/%hello = "Hello world!"
```

e adesso provate in qualsiasi finestra:

```
echo %hello
```

il risultato sarà... niente, perché la "vita" della variabile sarà terminata con l'esecuzione del comando.

Le variabili si creano assegnando loro un valore e le si distrugge assegnandogli valore nullo, ad esempio:

Creazione:

```
%nick="Grifisx"
```

Distruzione:

```
%nick=""
```

(*) vale il discorso relativo alle variabili globali: nel nuovo parser saranno locali tutte le variabili non dichiarate nel comando `global`.

Array:

Un array è una collezione di dati variabili indicizzati per numero, il primo indice dell'array è 0 mentre l'ultimo è uguale alla grandezza dell'array meno uno (poiché si parte da zero).

Per ottenere il numero di elementi che è contenuto in un array possiamo usare l'espressione `%ArrayEsempio[]#`.

Non è necessario dichiarare la grandezza dell'array come in altri linguaggi di programmazione, a mano a mano che si aggiungerà un numero, la grandezza del nostro array varierà automaticamente e se il primo elemento che assegneremo, lo assegneremo ad un indice maggiore di 0, tutte le posizioni precedenti saranno vuote.

Proviamo ad esempio:

```
%Array[0]=Grifisx
```

```
%Array[1]=Noldor
```

```
%Array[2]=Pragma
```

```
#Stampo il contenuto di tutto l'array
echo %Array[]
#Stampo la grandezza dell'array
echo %Array[]#
#Stampo solo il primo elemento
echo %Array[0]
```

Mettiamolo nello script tester ed eseguiamo.
Adesso proviamo questo codice:

```
%Array[0]=Grifisx
%Array[1]=Non mostrare questo
%Array[2]=Noldor
%Array[5]=Segreto shhhh..
%Array[8]=Pragma
for(%i=0;%i < %Array[]#;%i+=2)echo Entry %i: \"%Array[%i]\";
```

Come vedete è abbastanza semplice crearsi delle collezioni indicizzate per numero, così come lo è anche muoversi all'interno di esse, qui si è voluto usare un ciclo *for* ma ovviamente avremmo anche potuto usare un *foreach(%item,%Array[])echo %item*, oppure un ciclo *while*.

Un array potremmo anche inizializzarlo in questo modo **`%Array[]=$array(Grifisx,Noldor,Pragma,Madero)`**; ovvero utilizzando la funzione `$array(<el1>,<el2>,<el3>,<el4>,..)`. Adesso è il momento di esaminare il fratello maggiore dell'array, ovvero sia il dizionario.

Dizionario

I dizionari non sono altro che array associativi di stringhe (detto in termini "poveri", array con indice non numerico), per capire bene, riprendiamo l'esempio del manuale ufficiale:

```
%Songs{Jimi Hendrix} = Voodoo child
%Songs{Shawn Lane} = Gray piano's flying
%Songs{Mina} = Brava
%Songs{Greg Howe} = "Full Throttle"
# Mostra tutto in una stringa
echo %Songs{}
# Mostra tutti gli elementi del dizionario
foreach(%var,%Songs{ })echo %var
```

Ovviamente anche qui, come negli array, `%Songs{}#` restituirà il numero degli elementi del dizionario. Mentre `%Songs{}@` restituirà una lista degli elementi separata da virgole.

Operatori e assegnamento

"=" (Assegnamento)

L'operatore di assegnamento è "=" e funziona come in tutti i linguaggi, alcuni esempi sono i seguenti:

```
# Assegno alla variabile locale %idx il valore di 0
%idx=0;
# Assegno alla variabile globale %My_Nick il mio nick
%My_Nick = "Grifisx";
# Assegno alla variabile "%nome" il valore ritornato da una funzione
%nome = $function();
```

```
# Memorizzo nell'elemento 0 dell'array "%Rubrica" la stringa "start"
%Rubrica[0]="start";
```

"=~" (Binding) [scripting avanzato](#)

Questo operatore è sicuramente per una cerchia più matura di scripters.

Serve per fare delle ricerche e sostituzioni all'interno di una stringa, utilizzando anche le regular expression.

Sintassi di base:

<stringa_base> =~ <operazione>[parametri]

Dove <operazione> può essere 't','s'.

<stringa_base> è la stringa su cui operare <operazione>.

- 't' serve per la sostituzione di lettere.

La sintassi completa è:

<stringa_base>=~t/<CaratteriDaSostituire>/<CaratteriDiSostituzione>/

Questa operazione può essere anche effettuata con 'y' or 'tr' (per preservare la compatibilità con altri linguaggi).

Esempio:

```
%A=This is a test string
echo %A
%A=~ tr/abcdefghi/ABCDEFGH/
echo %A
```

- 's' è per sostituire delle associazioni di lettere.

La sintassi completa è:

<stringa_base>=~s/<pattern_da_cercare>/<pattern_da_sostituire>/[flag s]

Esempio con regular expression:

```
%A=This is a test string
echo %A
%A=~ s/([a-z])i([a-z])/\1I\2/
echo %A
%A=~ s/([a-z])i([a-z])/\1@\2/gi
echo %A
```

[flags] possono essere una combinazione delle lettere 'g','i' e 'w'.
'g' ->ricerca globalmente non fermandosi alla prima occorrenza del <pattern_da_ricerca>.

'i' ->ricerca non case sensitive.

'w' ->ricerca tramite semplice wildcards.

"X=" Operazioni aritmetiche con se stessi

Sintassi Generale:

<left_operand> <operation> <right_operand>

Dove <left_operand> e <right_operand> devono essere numeri.

Tutte queste operazioni effettuano <operation> (che può essere +,-,*,/,%,|,&) tra l'operatore di destra e quello di sinistra, e quindi il risultato viene conservato nell'operatore di sinistra (che deve, ovviamente, essere una variabile oppure un elemento di un array oppure di un dizionario).

<operation> può essere:

+= : somma <right_operand> a <left_operand>

-= : sottrae <right_operand> da <left_operand>

*= : moltiplica <left_operand> per <right_operand>

%= : calcola il modulo <right_operand> di <left_operand>

|= : calcola l'OR logico tra <left_operand> e <right_operand>
&= : calcola l'AND logico tra <left_operand> e <right_operand>
/= : divide <left_operand> per <right_operand>
Es:

```
%A=8  
%A+=3  
echo %A
```

"++,--" Operatori di incremento e decremento

Questi due operatori funzionano solo con variabili di tipo numerico.

Sintassi Generale:

<left_operand> <operator>

++ incrementa <left_operand> di una unità

-- decrementa <left_operand> di una unità

sarebbe come fare += 1 oppure -= 1.

ES:

```
%A=3  
%A++  
echo %A
```

Il risultato sarà, ovviamente, **4**.

Questi operatori sono utilizzati spessissimo nei cicli come ad esempio:

```
%idx=0;  
while(%idx==8)  
{  
    echo Valore attuale %idx;  
    %idx++;  
}
```

"<<,<+,<" Operatori di concatenazione delle stringhe

Operatore <+ :appende l'operando di destra a quello di sinistra in modo continuo

Operatore << : appende l'operando di destra a quello di sinistra separandolo da uno spazio.

Operatore < , : simile a '<<' appende separando gli operatori tramite una ','.

ES:

```
%frase= Ciao  
%frase < , come stai  
%frase << ?  
echo %frase
```

Mettiamo sempre nel nostro script tester ed eseguiamo per vedere i risultati.

Cicli

while

Sintassi:

```
while (<condizione>) {<comandi>;
```

Il comando `while` esegue in modo ciclico un comando, o un blocco di <comandi> fino a quando si verifica (o non si verifica) la <condizione> imposta.

ES:

```
%i = 0;
while(%i != 10)
{
    echo %i;
    %i++;
}
echo Fine
```

Ecco un esempio con un unico comando:

```
%i = 0;
while(%i != 10) %i++;
echo %i;
```

Si può interrompere il ciclo con il comando `break`

I metodi di confronto che posso essere utilizzati sono `==` (è uguale a), `!=` (è diverso da), `<=` (è minore o uguale a), `>=` (è maggiore o uguale a), `<` (è minore di), `>` (è maggiore di), `!<variabile>` (non esiste es: `if(!%nick){<comandi>` significa: se non c'è la variabile `%nick` e quindi non ha un valore), `<variabile>` (esiste Es: `if(%nick){}` significa: se c'è la variabile `%nick` è ha un valore).

Inoltre si possono usare condizioni multiple concatenandole tramite `&&` (che corrisponde ad un AND logico, ovvero i comandi vengono eseguiti solo se entrambe le condizioni poste sono vere) oppure tramite `||` (che corrisponde ad un OR logico, ovvero i comandi vengono eseguiti solo se almeno una delle condizioni è vera).

for

Sintassi:

```
for([inizializzazione];<condizione>;[operazione]) {<comandi>;}
```

Il ciclo `for(;;){}` permette l'inizializzazione della variabile che utilizziamo come indice per ciclare, l'imposizione della condizione valida per l'esecuzione del ciclo e l'operazione che ci permette di agire sull'indice, tutto tramite un unico comando complesso.

Si può interrompere il ciclo con il comando `break`.

ES:

```
# Con un unico comando
for(%i = 0;%i < 100;%i++)echo %i
for(%i = 100;%i;%i -= 10)echo %i
%i = 0
# Con un blocco di comandi
for(;%i;)
{
    echo %i
    %i++
    if(%i > 10)
        break
}
```

foreach

Sintassi:

```
foreach(<variabile>,[<item>[,<item2>[,<item3>[.....]]]) {<comando>;
```

Esegue <comando> (che può essere un blocco di comandi racchiuso tra le {} come per il while e il for), fino a quando può assegnare a <variabile> un <itemX>.

<item> può essere una variabile qualsiasi, un array, un dizionario, o una funzione che restituisce un valore, o una lista di valori.

ES:

```
foreach(%i,1,2,3,4,5,6,7,8,9)echo %i
```

oppure provate semplicemente a fare in una finestra di un canale
/foreach(%nick,\$chan.users) echo Utente: %nick

In quest'ultimo caso il risultato è giustificato dal fatto che la funzione `$chan.users` restituisce la lista dei nick presenti nel canale separati da virgola.

Per rendervi conto provate uno `/echo $chan.users`.

Condizioni di controllo

if/else

Sintassi:

```
if(<condizione>){<comando1>}[else {<comando2>}]
```

Esegue il <comando1> (o il blocco di comandi) se la <condizione> si verifica, se è utilizzato anche `else` eseguirà <comando2> (o il secondo blocco di comandi) se la <condizione> dell'`if` è falsa.

ES:

```
%idx=0
while(%idx<=10)
{
    if(%idx==3 && %idx!=6) echo TRE
        else echo %idx
    if(%idx==6)
    {
        echo S
        echo E
        echo I
    }
    else
    {
        echo .....
    }
}
%idx++
}
```

Anche qui come nel ciclo while i metodi di confronto che posso essere utilizzati sono `==` (è uguale a), `!=` (è diverso da), `<=` (è minore o uguale a), `>=` (è maggiore o uguale a), `<` (è minore di), `>` (è maggiore di), `!<variabile>` (non esiste es: `if(!%nick){<comandi>}` significa: se non c'è la variabile `%nick` e quindi non ha un valore), `<variabile>` (esiste Es: `if(%nick){}` significa: se c'è la variabile `%nick` è ha un valore).

Inoltre si possono usare condizioni multiple concatenandole tramite **&&** (che corrisponde ad un AND logico, ovvero i comandi vengono eseguiti solo se entrambe le condizioni poste sono vere) oppure

tramite `||` (che corrisponde ad un OR logico, ovvero i comandi vengono eseguiti solo se almeno una delle condizioni è vera).

switch\case

Sintassi:

```
switch [-s] (<expression>
{
    case(<value>) [:]<command>
    [break]
    match(<wildcard_expression>) [:]<command>
    [break]
    regexpr(<regular_expression>) [:]<command>
    [break]
    default[:]<command>
    [break]
}
```

Il costrutto switch è stato arricchito (rispetto al normale costrutto che si trova nel linguaggio C) da 2 "nuove" etichette, ovvero `match()` che ci permette di fare un confronto tramite le normali wildcard
ES:

```
%nick=Grifisx
switch(%nick)
{
    match(*r?fisx)
    {
        echo Hello Grifisx
        break
    }
    match(*W?fisx)
    {
        echo who are you?
        break
    }
}
```

Poi troviamo il normale case
ES:

```
%nick=Grifisx
switch(%nick)
{
    case(Grifisx)
    {
        echo Hello Grifisx
        break
    }
    case(WHO)
    {
        echo who are you?
        break
    }
}
```

e infine l'etichetta `regexpr` che, agli utenti più avanzati di

utilizzare di fare confronti anche utilizzando le regular expression. Infine lo switch (-s) può essere usato per far si che i confronti non diventino case sensitive (cioè distinguano maiuscole e minuscole).

/ECHO STOP.

" Tu vedi cose e ne spieghi il perché, io invece immagino cose che non sono mai esistite e mi chiedo perché no." (George Bernad Shaw)

Grifisx