

Special Shell Variables & Parameter Substitution and Expansion

\$ es utilizado para expansión de parámetros y sustitución de comandos

File descriptors

Special Shell Variables

variable	Significado
\$0	Nombre del script que se está ejecutando
\$1	Parámetro 1
\$2 - \$9	Parámetros del 2 al 9
\${10}	Parámetro 10
\$#	Número de parámetros
«\$*»	Todos los parámetros en 1 cadena, ha de llevar «
«\$@»	Todos los parámetros
\${#*}	Número de parámetros pasados (¿como \$#?)
\${#@}	Número de parámetros pasados (¿como \$#?)
\$?	Valor de retorno
\$\$	ID de proceso del script (PID)
\$-	Flags pasados al script
\$_	Último argumento del comando anterior
#!	PID del último trabajo ejecutado en background

Mirar también: <http://wiki.bash-hackers.org/scripting/posparams>

arrays y tablas

- permite almacenar 1024 variables (de la 0 a la 1023)
- formato:
 - setear: `variable[n]='valor'`
 - setear: `variable=(valor valor valor valor)`
 - recuperar: `VALOR=${variable[n]}`
 - recuperar: `echo ${variable[n]}`
 - recuperar todos los valores: `echo ${variable[*]}`
- /via: <http://ovtoaster.com/shell-scripts-en-linux-variables-y-parametros/>

Parameter substitution and Expansion

expresión	Significado
\${variable}	valor de variable, lo mismo que \$variable
\${parametro-valorPorDefecto}}	si el parámetro no está establecido, se le asigna el de por defecto
\${parametro:-valorPorDefecto}}	si el parámetro no está establecido o es vacío, se le asigna el de por defecto (sobre variables solo?)
\${parametro=valorPorDefecto}}	si el parámetro no está establecido, se le asigna el de por defecto (no funciona sobre parametros posicionales -\$1,\$2...-)

expresión	Significado
<code>\${parametro:=valorPorDefecto}</code>	si el parámetro no está establecido o es vacío, se le asigna el de por defecto (no funciona sobre parametros posicionales <code>-\$1,\$2...-</code>)
<code>\${variable?MensajeError}</code>	comprueba si variable no está establecida, mostrando el mensaje de error y para la ejecución del script
<code>\${variable:?MensajeError}</code>	comprueba si variable no está establecida o si es vacía, mostrando el mensaje de error y para la ejecución del script
<code>\${variable+OTHER}</code>	si el parámetro está establecido, se le asigna OTHER, en otro caso, NULL
<code>\${variable:+OTHER}</code>	si el parámetro está establecido, se le asigna OTHER, en otro caso, NULL
<code>\${!prefijoVariable*}</code>	devuelve todas las variables declaradas que empiecen por prefijoVariable
<code>\${!prefijoVariable@}</code>	devuelve todas las variables declaradas que empiecen por prefijoVariable

para manipular, mirar **shell expansion pattern matching** : <https://www.cyberciti.biz/tips/bash-shell-parameter-substitution-2.html>

resumen expansion

/via: [Enllaç extern](#)

Looking for a specific syntax you saw, without knowing the name?

- [Simple usage](#)
 - `$PARAMETER`
 - `${PARAMETER}`
- [Indirection](#)
 - `${!PARAMETER}`
- [Case modification](#)
 - `${PARAMETER^}`
 - `${PARAMETER^^}`
 - `${PARAMETER,}`
 - `${PARAMETER,,}`
 - `${PARAMETER~}`
 - `${PARAMETER~~}`
- [Variable name expansion](#)
 - `${!PREFIX*}`
 - `${!PREFIX@}`
- [Substring removal](#) (also for **filename manipulation!**)
 - `${PARAMETER#PATTERN}`
 - `${PARAMETER##PATTERN}`
 - `${PARAMETER%PATTERN}`
 - `${PARAMETER%%PATTERN}`
- [Search and replace](#)
 - `${PARAMETER/PATTERN/STRING}`
 - `${PARAMETER//PATTERN/STRING}`
 - `${PARAMETER/PATTERN}`
 - `${PARAMETER//PATTERN}`
- [String length](#)
 - `${#PARAMETER}`
- [Substring expansion](#)
 - `${PARAMETER:OFFSET}`
 - `${PARAMETER:OFFSET:LENGTH}`
- [Use a default value](#)
 - `${PARAMETER:-WORD}`
 - `${PARAMETER-WORD}`
- [Assign a default value](#)

- `${PARAMETER:=WORD}`
- `${PARAMETER=WORD}`
- Use an alternate value
 - `${PARAMETER:+WORD}`
 - `${PARAMETER+WORD}`
- Display error if null or unset
 - `${PARAMETER:?WORD}`
 - `${PARAMETER?WORD}`

Operaciones sobre cadenas

/vía: <http://www.marqueta.org/cadenas-en-bash/>

expresión	Significado	Ejemplo
<code>\${#parametro}</code>	longitud de parámetro	<code>echo \${#string}</code>
<code>\${cadena:posición}</code>	extrae subcadena desde posición	<code>echo \${string:4}</code>
<code>\${cadena:posición:longitud}</code>	extrae subcadena desde posición la longitud solicitada	<code>echo \${string:4:7} ; echo \${string::-1}</code>
<code>\${cadena#subcadena}</code>	eliminar subcadena del principio de la cadena	<code>echo \${string#substring}</code>
<code>\${cadena%subcadena}</code>	eliminar subcadena del final de la cadena	<code>echo \${string%substring}</code>
<code>\${cadena/s1/s2}</code>	reemplazar primera aparición s1 por s2	
<code>\${cadena//s1/s2}</code>	reemplazar todas apariciones s1 por s2	
<code>\${cadena/#s1/s2}</code>	reemplazar si hay coincidencia al principio de cadena	
<code>\${cadena/%s1/s2}</code>	reemplazar si hay coincidencia al final de cadena	
<code>\${cadena##*separador}</code>	extrae el último elemento de la lista, usando separador	<code>data=foo,bar,baz; echo \${data##*,}</code>

- `$ a=foo; b=bar; [«$a» = «$b»] && echo «iguales» || echo «diferentes»` ← espacios en condición importantes
- `$ a=foo; b=bar; [«$a» == «$b»] && echo «iguales» || echo «diferentes»` ← idem anterior
- `$ a=foo; b=bar; [«$a» != «$b»] && echo «diferentes» || echo «iguales»`
- `$ a=foto.jpg ; "$a" != *.gif && echo «no gif» || echo «gif»` ← doble corchetes hacen pattern matching
- `$ a=foo; b=bar; [«$a» = «foo» -a «$b» = «bar»] && echo «todo OK» || echo «algo falla»` ← -a equivale AND
- `$ a=foo; b=bar; [«$a» = «foo» -o «$b» = «bar»] && echo «al menos 1 OK» || echo «ninguno OK»` ← -o equivale OR
- `${@:param:num_param}` → coge desde <param> el número de parámetros indicado por <num_param>
 - permite acceder a parámetros más allá del 10
 - `${@:7:2}` → devuelve \$7 y \$8
 - @ indica todos → (se podrá indicar otra cosa?)
- recorrer una cadena palabra a palabra:

```
cadena="Esto es una cadena"
arr=( $cadena )
for i in ${arr[@]}; do echo $i; done
```

```
cadena="Esto es una cadena"  
for i in ${cadena[@]}; do echo $i; done
```

Operaciones sobre nombres de ficheros:

partiendo de esta cadena: `foo=/tmp/mi.directorio/imagen.png`

- `path = ${foo%/*} → »/tmp/mi.directorio«`
- `file = ${foo##*/} → «imagen.png»`
- `base = ${file%*.} → «imagen»`
 - hint: si la cadena tiene más de un ».«, usar `base=${file%.*}`
- `ext = ${file##*.} ← en este caso, devolvería: «directorio/imagen.png», cogiendo desde el primer punto que encuentra`
 - hint: si la cadena tiene más de un ».«, usar `ext=${file##*.} → «png»`

Operaciones con cadenas (varios)

- extraer información de procesos: `ps -L u n | tr -s » « | cut -d » « -f 2,3,14-`
 - <http://stackoverflow.com/questions/15643834/using-bash-ps-and-cut-together>

Operaciones con cadenas desde bash

- <http://www.marqueta.org/cadenas-en-bash/>
- <http://stackoverflow.com/questions/428109/extract-substring-in-bash>
- <http://rm-rf.es/unix-uso-del-comando-cut/>

indirection

usar el valor de una variable para acceder al contenido de otra

```
# declaramos 2 variables  
export xzyzy=plugh ; export plugh=cave  
  
echo ${xzyzy} # normal, xzyzy to plugh -> plugh  
  
echo ${!xzyzy} # indirection, xzyzy to plugh to cave -> cave
```

existen 2 casos en el que este comportamiento se ve modificado:

- `${!N*}` : muestra las variables que empiecen por N
- `${!name[@]}`

```
export myVar="hi"  
echo ${!my*} # myVar  
  
export ${!my*}="bye"  
echo $myVar # bye
```

/vía: <https://stackoverflow.com/questions/8515411/what-is-indirect-expansion-what-does-var-mean>

bash shell expansion

comando	descripción
<code>\${Var}</code>	variable
<code>\${!Var}</code>	Indirect expansion
<code>\${!Var@}</code>	Prefix expansion
<code>\${!Var[@]}</code>	Array keys expansion
<code>\${Var[@]}</code>	Plain array expansion

title=indirection

```
varname=var_one
var_one=a-value

echo "${varname}" # var_one
echo "${!varname} and ${var_one}" # a-value and a-value
```

title=prefix

```
head_one=foo
head_two=bar

printf '<%s> ' "${!head@}" # <head_one> <head_two>

printf '<%s> ' "${!head*}" # <head_one head_two>
```

nota: las variables están enganchadas por un espacio, que es el valor por defecto de IFS (espacio, tabulador, nueva_linea)

title=plain array

```
Array[1]=This
Array[2]=is
Array[3]=a
Array[4]=simple
Array[5]=test.

printf '<%s> ' "${Array[@]}" # <This> <is> <a> <simple> <test.>

printf '<%s> ' "${Array[*]}" # <This is a simple test.>
```

title= array associative list

```
unset Array # erase any notion of variable
array.
declare -A Array # make it associative

Array=( [foo]=one [bar]=two [baz]=three ) # give it values.

printf '<%s> ' "${Array[@]}" # <two> <three> <one> # List of values.

$ printf '<%s> ' "${!Array[@]}" # <bar> <baz> <foo> # List of keys
```

```
$ printf '<%s> ' "${Array[*]}" # <two three one> # One string of list of values.  
$ printf '<%s> ' "${!Array[*]}" # <bar baz foo> # One string of list of keys.
```

/vía: <https://unix.stackexchange.com/questions/247589/usage-of-in-parameter-expansion>

From:

<https://miguelangel.torresegea.es/wiki/> - **miguel angel torres egea**

Permanent link:

<https://miguelangel.torresegea.es/wiki/linux:bash:shellvariables?rev=1539336329>

Last update: **12/10/2018 02:25**

