

arrays bash

linux, bash, pendiente, wiki

/via: <http://www.thegeekstuff.com/2010/06/bash-array-tutorial/>

más información de interés (arrays indirection)

<http://wiki.bash-hackers.org/syntax/arrays?rev=1534920675>

declaración y asignación

```
array=(valor0 'valor uno' 'valor dos')
```

```
array[0]='valor0'
array[1]='valor uno'
array[2]='valor dos'
```

inicialización durante declaración

```
declare -a array=(valor0 'valor uno' 'valor dos')
```

+ asignaciones

```
array[n]=valor
array[cadena]=valor
array=valor # se asigna al índice 0 (en asociativos, a "0")
array=(valor1 valor2 valor3)
array=( [0]=valor1 1=valor2 2=valor3)
array=( [cadena]=valor1 [cadena2]=valor2)
array+=(valor5 valor6) # añade
array=("${otro_array[@]}")
```

recuperación

Syntax	Description
<code>\${ARRAY[N]}</code>	Expands to the value of the index N in the indexed array ARRAY. If N is a negative number, it's treated as the offset from the maximum assigned index (can't be used for assignment) - 1
<code>\${ARRAY[S]}</code>	Expands to the value of the index S in the associative array ARRAY.
<code><<\${ARRAY[@]}<< \${ARRAY[@]}<< <<\${ARRAY[*]}<< \${ARRAY[*]}</code>	Similar to mass-expanding positional parameters , this expands to all elements. If unquoted, both subscripts * and @ expand to the same result, if quoted, @ expands to all elements individually quoted, * expands to all elements quoted as a whole.

Syntax	Description
<pre>«\${ARRAY[@]:N:M}« \${ARRAY[@]:N:M} «\${ARRAY[*]:N:M}« \${ARRAY[*]:N:M}</pre>	Similar to what this syntax does for the characters of a single string when doing substring expansion , this expands to M elements starting with element N. This way you can mass-expand individual indexes. The rules for quoting and the subscripts * and @ are the same as above for the other mass-expansions.
Sintaxis	Descripción
<pre>\${ARRAY[N]}</pre>	recupera el elemento N
<pre>\${ARRAY[S]}</pre>	recupera el elemento S en arrays asociativos
<pre>«\${ARRAY[@]}« \${ARRAY[@]} «\${ARRAY[*]}« \${ARRAY[*]}</pre>	Recupera todos los elementos Si está entrecomillado, @ devuelve cada elemento, * devuelve un único resultado con todos los elementos. Si no lo está, las dos devuelven lo mismo
<pre>«\${ARRAY[@]:N:M}« \${ARRAY[@]:N:M} «\${ARRAY[*]:N:M}« \${ARRAY[*]:N:M}</pre>	De funcionamiento similar a la regla anterior, añade que devuelve desde el elemento N, M elementos

metadata

Syntax	Descripción
<pre> \$#ARRAY[N]</pre>	Longitud del elemento N
<pre> \$#ARRAY[STRING]</pre>	idem en asociativo
<pre> \${#ARRAY[@]} \${#ARRAY[*]}</pre>	Número de elementos
<pre> \${!ARRAY[@]} \${!ARRAY[*]}</pre>	Extrae los índices del array, @ elemento a elemento, * en un bloque (desde BASH 3.0)
Syntax	Description
<pre> \$#ARRAY[N]</pre>	Expands to the length of an individual array member at index N (stringlength)
<pre> \$#ARRAY[STRING]</pre>	Expands to the length of an individual associative array member at index STRING (stringlength)
<pre> \${#ARRAY[@]} \${#ARRAY[*]}</pre>	Expands to the number of elements in ARRAY
<pre> \${!ARRAY[@]} \${!ARRAY[*]}</pre>	Expands to the indexes in ARRAY since BASH 3.0

destrucción

The [unset](#) builtin command is used to destroy (unset) arrays or individual elements of arrays.

Syntax	Description
<pre>unset -v ARRAY unset -v ARRAY[@] unset -v ARRAY[*]</pre>	Destroys a complete array
<pre>unset -v ARRAY[N]</pre>	Destroys the array element at index N
<pre>unset -v ARRAY[STRING]</pre>	Destroys the array element of the associative array at index STRING

usar siempre comillas para encerrar la variable para evitar problemas con las variables glob

mostar array/elementos/subcadenas

- todo el array: `echo ${array[@]}`
- un elemento: `echo ${array[n]}`

- del elemento n al m: `echo ${array[@]:n:m}`
- del elemento n, subcadena a-b: `echo ${array[n]:a:b}`

longitud

- del array: `echo ${#array[@]}`
- del primer elemento: `echo ${#array}`
- del tercero: `echo ${#array[2]}`

adición / eliminación / sustitución elementos

- añadir elemento(s) a un array existente: `array=(«${array[@]}» «valor 3» «valor 4»)`
- eliminar un elemento n: `unset array[n]`
 - existe forma de eliminar el índice o eliminar por patrón
- eliminar un array: `unset array`
- cambiar string1 por string2: `${array[@]/string1/string2}`
- copiar un array: `new_array=(«${array[@]}»)`
- concatenar arrays: `new_new_array=(«${array[@]}» «${new_array[@]}»)`

cargar contenido de un fichero

- `array=(` cat file `)`

funky constructs

- todos los items del array: `${array[*]}`
- todos los índices del array: `${!array[*]}`
- número de items: `${#array[*]}`
- vía: <http://www.linuxjournal.com/content/bash-arrays>

ejemplos

quoted "*", quoted "@", unquoted

```
#!/bin/bash

array=("first item" "second item" "third" "item")

echo "Number of items in original array: ${#array[*]}"
for ix in ${!array[*]}
do
    printf "    %s\n" "${array[$ix]}"
done
echo

arr=(${array[*]})
```

```
echo "After unquoted expansion: ${#arr[*]}"
for ix in ${!arr[*]}
do
    printf "    %s\n" "${arr[$ix]}"
done
echo

arr=("${array[*]}")
echo "After * quoted expansion: ${#arr[*]}"
for ix in ${!arr[*]}
do
    printf "    %s\n" "${arr[$ix]}"
done
echo

arr=("${array[@]}")
echo "After @ quoted expansion: ${#arr[@]}"
for ix in ${!arr[*]}
do
    printf "    %s\n" "${arr[$ix]}"
done
```

resultado:

```
Number of items in original array: 4
    first item
    second item
    third
    item

After unquoted expansion: 6
    first
    item
    second
    item
    third
    item

After * quoted expansion: 1
    first item second item third item

After @ quoted expansion: 4
    first item
    second item
    third
    item
```

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

Permanent link:
<https://miguelangel.torresegea.es/wiki/linux:bash:arrays?rev=1539081258>

Last update: **09/10/2018 03:34**

