10.2. Array variables

10.2.1. Creating arrays

An array is a variable containing multiple values. Any variable may be used as an array. There is no maximum limit to the size of an array, nor any requirement that member variables be indexed or assigned contiguously. Arrays are zero-based: the first element is indexed with the number 0.

Indirect declaration is done using the following syntax to declare a variable:

`ARRAY[INDEXNR]=value`

The *INDEXNR* is treated as an arithmetic expression that must evaluate to a positive number.

Explicit declaration of an array is done using the **declare** built-in:

declare `-a ARRAYNAME`

A declaration with an index number will also be accepted, but the index number will be ignored. Attributes to the array may be specified using the **declare** and **readonly** built-ins. Attributes apply to all variables in the array; you can't have mixed arrays.

Array variables may also be created using compound assignments in this format:

`ARRAY`=(value1 value2 ... valueN)

Each value is then in the form of *[indexnumber=]string*. The index number is optional. If it is supplied, that index is assigned to it; otherwise the index of the element assigned is the number of the last index that was assigned, plus one. This format is accepted by **declare** as well. If no index numbers are supplied, indexing starts at zero.

Adding missing or extra members in an array is done using the syntax:

`ARRAYNAME[indexnumber]=value`

Remember that the **read** built-in provides the `-a` option, which allows for reading and assigning values for member variables of an array.

10.2.2. Dereferencing the variables in an array

In order to refer to the content of an item in an array, use curly braces. This is necessary, as you can see from the following example, to bypass the shell interpretation of expansion operators. If the index number is @ or *, all members of an array are referenced.

```
[bob in ~] ARRAY=(one two three)

[bob in ~] echo ${ARRAY[*]}
```

```
one two three

[bob in ~] echo $ARRAY[*]
one[*]

[bob in ~] echo ${ARRAY[2]}
three

[bob in ~] ARRAY[3]=four

[bob in ~] echo ${ARRAY[*]}
one two three four
```

Referring to the content of a member variable of an array without providing an index number is the same as referring to the content of the first element, the one referenced with index number zero.

10.2.3. Deleting array variables

The **unset** built-in is used to destroy arrays or member variables of an array:

```
[bob in ~] unset ARRAY[1]

[bob in ~] echo ${ARRAY[*]}
one three four

[bob in ~] unset ARRAY

[bob in ~] echo ${ARRAY[*]}
<--no output-->
```

10.2.4. Examples of arrays

Practical examples of the usage of arrays are hard to find. You will find plenty of scripts that don't really do anything on your system but that do use arrays to calculate mathematical series, for instance. And that would be one of the more interesting examples...most scripts just show what you can do with an array in an oversimplified and theoretical way.

The reason for this dullness is that arrays are rather complex structures. You will find that most practical examples for which arrays could be used are already implemented on your system using arrays, however on a lower level, in the C programming language in which most UNIX commands are written. A good example is the Bash **history** built-in command. Those readers who are interested might check the `built-ins` directory in the Bash source tree and take a look at `fc.def`, which is processed when compiling the built-ins.

Another reason good examples are hard to find is that not all shells support arrays, so they break compatibility.

After long days of searching, I finally found this example operating at an Internet provider. It distributes Apache web server configuration files onto hosts in a web farm:

```
#!/bin/bash

if [ $(whoami) != 'root' ]; then
        echo "Must be root to run $0"
        exit 1;
fi
if [ -z $1 ]; then
        echo "Usage: $0 </path/to/httpd.conf>"
        exit 1
```

```
fi

httpd_conf_new=$1
httpd_conf_path="/usr/local/apache/conf"
login=htuser

farm_hosts=(web03 web04 web05 web06 web07)

for i in ${farm_hosts[@]}; do
        su $login -c "scp $httpd_conf_new ${i}:${httpd_conf_path}"
        su $login -c "ssh $i sudo /usr/local/apache/bin/apachectl graceful"

done
exit 0
```

First two tests are performed to check whether the correct user is running the script with the correct arguments. The names of the hosts that need to be configured are listed in the array `farm_hosts`. Then all these hosts are provided with the Apache configuration file, after which the daemon is restarted. Note the use of commands from the Secure Shell suite, encrypting the connections to remote hosts.

Thanks, Eugene and colleague, for this contribution.

Dan Richter contributed the following example. This is the problem he was confronted with:

"...In my company, we have demos on our web site, and every week someone has to test all of them. So I have a cron job that fills an array with the possible candidates, uses **date +%W** to find the week of the year, and does a modulo operation to find the correct index. The lucky person gets notified by e-mail."

And this was his way of solving it:

```
#!/bin/bash
# This is get-tester-address.sh
#
# First, we test whether bash supports arrays.
# (Support for arrays was only added recently.)
#
whotest[0]='test' || (echo 'Failure: arrays not supported in this version of
bash.' && exit 2)

#
# Our list of candidates. (Feel free to add or
# remove candidates.)
#
wholist=(
     'Bob Smith <bob@example.com>'
     'Jane L. Williams <jane@example.com>'
     'Eric S. Raymond <esr@example.com>'
     'Larry Wall <wall@example.com>'
     'Linus Torvalds <linus@example.com>'
   )
#
# Count the number of possible testers.
# (Loop until we find an empty string.)
#
count=0
while [ "x${wholist[count]}" != "x" ]
do
   count=$(( $count + 1 ))
done

#
```

```
# Now we calculate whose turn it is.
#
week=`date '+%W'`       # The week of the year (0..53).
week=${week#0}          # Remove possible leading zero.

let "index = $week % $count"   # week modulo count = the lucky person

email=${wholist[index]}     # Get the lucky person's e-mail address.

echo $email             # Output the person's e-mail address.
```

This script is then used in other scripts, such as this one, which uses a *here* document:

```
email=`get-tester-address.sh`   # Find who to e-mail.
hostname=`hostname`             # This machine's name.

#
# Send e-mail to the right person.
#
mail $email -s '[Demo Testing]' <<EOF
The lucky tester this week is: $email

Reminder: the list of demos is here:
    http://web.example.com:8080/DemoSites

(This e-mail was generated by $0 on ${hostname}.)
EOF
```
