УДК 681.3

А. В. Дробнич, Ю. А. Василенко (Ужгородский гос. ин-т информатики, экономики и права)

ИСПОЛЬЗОВАНИЕ АССОЦИАТИВНЫХ МАССИВОВ В ОБОЛОЧКЕ BASH UNIX-ПОДОБНЫХ ОПЕРАЦИОННЫХ СИСТЕМ

The creation and usage of associative arrays in the bash command shell realized on basis of Unix file system objects are discussed. The source texts of bash scripts for creation and removing of such arrays are shown.

Обсуждается создание и использование ассоциативных массивов в командной оболочке bash, реализованных в виде объектов файловой системы Unix. Представлены исходные тексты bash-скриптов, которые создают и удаляют такие массивы.

В большинстве учебников для системных администраторов и системных программистов по ОС Unix подчеркивается особая роль командных оболочек или шеллов. Они предоставляют собой более или менее законченные языки программирования для интеграции существующих программ в сложные макропрограммы и создания новых программ. Помимо расширения области действия существующих программ с помощью перенаправления информационных потоков между обьектами файловых систем и создания конвееров эти языки предоставляют средства традиционных языков программирования — переменные, операторы и выражения, средства ввода-вывода, команды ветвления алгоритма.

Существует большое число шеллов, разработанных разными программистами для разных целей. Это объясняется тем, что разработка шелл не является сложным делом [1]. Самая первая и наиболее распространенная оболочка носит название Bourne Shell (по имени знаменитого участника Unix-проекта). Далее по распространенности следует оболочка Bourne Again Shell, которая является развитием предыдущей оболочки и полностью совместима с ней. Другие популярные оболочки — Korn Shell, C Shell, Remote Shell, Windowing Korn Shell и т.д.

Существует устойчивое мнение [2–4], что для получения полных возможностей, заложенных в оболочках ОС Unix, необходимо изучить как минимум две оболочки — например Bourne Again Shell из-за ее распространенности на разных системах и Korn Shell или C Shell как наиболее мощные по заложенным возможностям. Среди возможностей, "нехватающих" Bourne Shell и Bourne Again Shell главным считается отсутствие массивов. Данная статья призвана показать с одной стороны, что на языке указанных шелл достаточно просто реализовываются массивы, с другой, что направленность шелл на работу со строками и обьектами файловой системы позволяет реализовать наиболее сложные с точки зрения традиционных языков программирования контейнеры — ассоциативные массивы.

В листинге 1 предлагается программа на языке Bourne Shell, которая в зависимости от контекста исполнения создает массив с заданным количеством пустых элементов, записывает значения в элемент массива или извлекает значение из конкретного элемента массива.

Принцип работы данной программы — создание массива как каталога во временной области файловой системы Unix — в директории /tmp. В таком случае элементами массива являются обычные файлы. Легко видеть, что индекс элемента массива, который предлагает последняя программа совершенно необязательно задавать в виде числа — вполне может подойти и произвольная строка. Таким образом мы видим,

что предложенный контейнер является ассоциативным массивом или хэш-таблицей — мы можем записывать и считывать произвольные данные в элемент, обозначаемый, например, числом 123, а также в элемент того же массива, обозначенный строкой "элемент, который хранит нужное мне завтра утром значение". Для назначения имен ключам таких массивов следует принимать лишь ограничения на имена файлов в Unix, которые более чем либеральны.

Листинг 1

```
# massiv
if test $# -ge 2 -a $# -le 4
  then
   if test $2 = "="
   then
       if test -d /tmp/massiv‘echo $1‘
      then
          echo error: u attemp to distoy existing massiv
      else
        mkdir /tmp/massiv‘echo $1‘
        i=1
       while test $i -le $3
         do
           touch /tmp/massiv‘echo $1‘/‘echo $i‘
           i=‘expr $i + 1‘
         done
      fi
  else
    if test $# -eq 2
    then
        cat /tmp/massiv‘echo $1‘/‘echo $2‘
    else
        if test $# -eq 4 -a $3 = "="
    then
       echo $4 > /tmp/massiv‘echo $1‘/‘echo $2‘
    fi
    fi
  fi
else
   echo Array Management Command
   echo usage:   massiv A = N "       " Nth elements array creation
   echo "       "massiv A M "       "  return the Mth element
   echo "       "massiv A L = K "  " write K to the Lth element
fi
```

Интересно также отметить такое свойство массива, построенного на файловой системе как работа с произвольными данными. Действительно, в элемент 1 массива мы можем записать число 3.1416 :

```
massiv test 1 = 3.1416
```

а в элемент 2 содержимое файла file1 :

```
massiv test 2 = ‘cat file1‘
```

Для чтения записанных данных можно предложить строки :

```
echo ‘massiv test 1‘
massiv test 2 | cat
```

Для удаления массивов, созданных приведенной выше программой, можно воспользоваться программой, приведенной на листинге 2.

Листинг 2

```
# killarray
if test -d /tmp/massiv‘echo $1‘
    then
         rm -r -f /tmp/massiv‘echo $argv[1]‘
    else
         echo The array doesn’t exist
fi
```

Ниже приведен пример программы на bash, которая вводит элементы массива, а затем выводит минимальный элемент:

Листинг 3

```
die(){
killarray temp;
}

echo "how many elements..."
read N
massiv temp = $N
i=1
s=0

while test $i -le $N
do
  echo "input $i-th element"
  read buf
  massiv temp $i = $buf
  s=‘expr $s + $buf‘
  i=‘expr $i + 1‘
done
echo $s
die
```

Учитывая такое свойство файловой системы Unix как многосвязность, т. е. возможность помещать ссылки на данный файл из разных каталогов, мы можем одному элементу — файлу нашего массива присвоить несколько имен, адресующих один и тот-же обьект. Это означает,что мы можем создавать сколько угодно ключей, адресующих данный элемент внутри данного массива. Это значительно расширяет традиционные возможности хэш-таблицы. Ниже приведена команда, создающая жесткую ссылку "aircraft"на элемент "Boeng737":

```
ln Boeng737 aircraft
```

тот же эффект можно получить и с помощью мягкой ссылки :

```
ln -s Boeng737 airbus
```

Существует широкий круг задач, где необходимо произвести сложный анализ строк либо текстовых файлов. В качестве примеров можно привести рассчет статистики работы пользователей на сервере, CGI-обработчики HTML-форм, и т. д. Для таких задач доминирующим кроссплатформным инструментом программирования считается язык Perl. Рассмотрим задачу подсчета работы пользователей на Unix - сервере. Частным случаем этой задачи есть подсчет статистики по выходу пользователей в Интернет [5]. Unix — команда last выдает характеристики отдельных сессий пользователей:

```
larry ttyp1 muadib.oit.unc.e Sun Jan 16 15:11 - 15:14 (00:03)
larry ttyp1 muadib.oit.unc.e Sun Jan 16 18:10 - 18:35 (00:25)
linus ttypf kruuna.helsinki. Sun Jan 16 18:20 - 19.10 (00.50)
```

Из этой информации мы должны извлечь пары данных "имя — общее время работы" и вывести в алфавитном порядке. Ниже приведена соответствующая программа на Perl:

Листинг 4

```
#!/usr/bin/perl
while(<STDIN>){
  if(/^(\S*)\s*.*\((.*):(.*)\)$/)  {
      $hours{$1} += $2;
      $minutes{$1} += $3;
      $logins{$1} ++;
      }
}
foreach $user (sort(keys %hours)){
  $hours{$user} += int($minutes{$user}/60)
  $minutes{$user} %= 60;
  print "User $user, total login time ";
  printf "%02d:02d, ",$hours{$user}, $minutes{$user};
  print "total logins $logins{$user}.\n";
}
```

Запуск этой программы

```
chmod +x logintime
last | logintime
```

приводит к выдаче информации:

```
User larry, total login time 00:28, total logins 2.
User linus, total login time 00:50, total logins 1.
```

Проанализируем данную программу. Во-первых в глаза бросается ее краткость. Действительно, вместо тяжеловесных инструкций с обычными индексными массивами, многократно сканирующими входящую информацию с целью анализа строк, создания списка пользователей, наполнения массивов временами сессий, сортировки и т.д., основная задача решается 4-мя строками: регулярное выражение сканирует каждую новую строку и выдает имя, часы и минуты в переменные $1, $2 и $3 соответственно. Далее программа работает с ассоциативными массивами hours, minutes и logins, индексируемыми именем пользователя $1 как ключем. Здесь иллюстрируется основное преимущество ассоциативных массивов — способность "на лету"связывать разнородные структуры данных. Для написания такой программы

в bash можно воспользоваться предложенными массивами над файловой системой. Для этого вместо регулярных выражений Perl нам понадобятся команды awk, которые умеют разбирать строки по достаточно сложным правилам и выполнять над полученными данными операции:

```
awk '{  print($1);  \
    print("   ");  \
    split ($0, M, \( );  \
    print(substr (M[2], 1, 2));  \
            print("   ");   \
    print(substr (M[2], 4, 2)  \
     }'
```

Скрипт-языки ОС Unix имеют широчайшие возможности и способны взаимозаменяться и взаимодополняться. Этот принцип, заложенный архитекторами Unix, подтверждается работой с ассоциативными массивами. Предложенный подход к созданию массивов в файловой системе требует значительно больше затрат времени, чем массивы в памяти традиционных языков программирования. Для создания массивов в области памяти можно предложить простой выход — написать программу-сервер на языке С. С другой стороны, предложенный подход имеет ряд привлекательных возможностей: например, создавая подкаталоги в директории массива, можно моделировать подмассивы языка Perl, реализуемым видится построение многомерных массивов и баз знаний.

1. *Теренс Ч.* Системное программирование на C++ для Unix. – К.: Издательская группа BHV, 1999.
2. *Дунаев С.* UNIX SYSTEM V. Releas 4.2. Общее руководство. – М.: Диалог-МИФИ, 1995.
3. *Bourne S. R.* The UNIX System. – MA, Addison-Wesley, Reading, 1983.
4. *Kernighan B. W., Pike R.* The UNIX Programming Environment. – Englewood Cliffs, NJ, Prentice-Hall, 1984.
5. *Matt W., Matthias K. D., Kaufman L.* Running Linux, Third Edition. – Sebastopol, CA, O'Reilly, 1999.

Получено 24.09.2003