Сценарии

Введение в разработку сценариев

Командные оболочки, такие, как **bash** и **Korn shell** поддерживают программные конструкции, которые могут быть сохранены в форме **сценариев**. Эти **сценарии**, в свою очередь, впоследствии могут использоваться в качестве реализаций дополнительных команд **командной оболочки**. Многие команды Linux реализованы в форме **сценариев**. Например, **сценарии для обслуживания профиля пользователя** исполняются при входе пользователя в систему, а **сценарии инициализации** - при остановке и запуске **демона**.

Исходя из вышесказанного, системные администраторы также должны иметь базовое представление о **сценариях командной оболочки** для понимания того, как функционируют их серверы, запускаются, обновляются, исправляются, поддерживаются, настраиваются и удаляются их приложения, а также для понимания устройства программного окружения.

Целью данной главы является предоставление достаточной информации для того, чтобы вы могли читать и понимать сценарии. Для этого вам совершенно не нужно становиться разработчиком сложных сценариев.

Предварительное чтение

Перед тем, как приступить к чтению данной главы, вам необходимо прочитать и понять главы из **Части III. "Раскрытие команд командной оболочкой"** и **Части IV. "Программные каналы и команды"**.

Hello world

По аналогии с практически любым курсом по программированию, мы начнем работу с разработки сценария **hello_world**. Следующий сценарий будет выводить строку **`Hello World`**.

```
echo Hello World
```

После создания этого простого сценария в редакторе **vi** или с помощью команды **`echo`** вам придется выполнить команду **`chmod +x hello_world`** для того, чтобы сделать файл сценария исполняемым. В том случае, если вы не будете добавлять путь к директории с вашими сценариями в список директорий из переменной окружения PATH,

вам придется вводить полный путь к сценарию для того, чтобы командная оболочка смогла найти его.

```
[paul@RHEL4a ~]$ echo echo Hello World > hello_world
[paul@RHEL4a ~]$ chmod +x hello_world
[paul@RHEL4a ~]$ ./hello_world
Hello World
[paul@RHEL4a ~]$
```

She-bang

Давайте немного доработаем наш пример, разместив строку **#!/bin/bash** в начале сценария. Последовательность символов **#!** называется **she-bang** (или иногда **sha-bang**), причем слово **she-bang** составлено из названий двух первых символов сценария.

```
#!/bin/bash
echo Hello World
```

Вы ни при каких обстоятельствах не можете быть уверены в том, какая командная оболочка используется в системе пользователя. Сценарий, превосходно работающий в командной оболочке **bash**, может не работать в командных оболочках **ksh, csh** или **dash**. Для того, чтобы проинструктировать командную оболочку о необходимости запуска вашего сценария в определенной командной оболочке, вы должны начинать ваш сценарий с последовательности символов **she-bang**, после которой должен располагаться путь к бинарному файлу командной оболочки, в которой сценарий должен исполняться. Приведенный ниже сценарий будет исполняться в командной оболочке **bash**.

```
#!/bin/bash
echo -n hello
echo Дочерняя командная оболочка bash `echo -n hello`
```

А этот сценарий будет исполняться в командной оболочке Korn shell (за исключением тех случаев, когда по пути **/bin/ksh** расположена жесткая ссылка на бинарный файл **/bin/bash**). Файл **/etc/shells** содержит список путей к командным оболочкам, установленным в вашей системе.

```
#!/bin/ksh
echo -n hello
```

```
echo Дочерняя командная оболочка Korn shell `echo -n
hello`
```

Комментарий

Давайте еще немного усовершенствуем наш пример, добавив строки комментариев.

```
#!/bin/bash
#
# Сценарий Hello World
#
echo Hello World
```

Переменные

Ниже приведен простой пример объявления переменной в сценарии.

```
#!/bin/bash
#
# простая переменная в сценарии
#
var1=4
echo var1 = $var1
```

Сценарии могут содержать переменные, но ввиду того, что сценарии исполняются в своих собственных экземплярах командных оболочек, переменные не смогут пережить момент завершения исполнения сценария.

```
[paul@RHEL4a ~]$ echo $var1

[paul@RHEL4a ~]$ ./vars
var1 = 4
[paul@RHEL4a ~]$ echo $var1

[paul@RHEL4a ~]$
```

Использование рабочей командной оболочки

К счастью, у вас имеется возможность исполнения сценария в той же рабочей командной оболочке; данная техника называется **использованием рабочей командной оболочки** (sourcing a script).

```
[paul@RHEL4a ~]$ source ./vars
var1 = 4
[paul@RHEL4a ~]$ echo $var1
4
[paul@RHEL4a ~]$
```

Представленная выше команда аналогична следующей команде.

```
[paul@RHEL4a ~]$ . ./vars
var1 = 4
[paul@RHEL4a ~]$ echo $var1
4
[paul@RHEL4a ~]$
```

Отладка сценария

Другой способ исполнения сценария в отдельной командной оболочке заключается во вводе команды **bash** перед именем сценария, которое в этом случае будет передаваться бинарному файлу командной оболочки в качестве параметра.

```
paul@debian6~/test$ bash runme
42
```

Дополнение данной команды до формы **bash -x** позволит вам ознакомиться со всеми командами, исполняемыми командной оболочкой (после раскрытия команд).

```
paul@debian6~/test$ bash -x runme
+ var4=42
+ echo 42
42
paul@debian6~/test$ cat runme
# сценарий runme
var4=42
echo $var4
paul@debian6~/test$
```

Обратите внимание на отсутствие строки комментария (первым символом которой является символ **#**), а также замену значения переменной перед исполнением команды для вывода данных **echo**.

Предотвращение подмены имен файлов сценариев с целью повышения привилегий в системе

Какой-либо пользователь может попытаться выполнить сценарий с установленным битом **setuid** с целью повышения привилегий в системе, подменив имя файла этого сценария путем создания ссылки на него (**root spoofing**). Это довольно редкая, но возможная атака. Для повышения безопасности функционирования вашего сценария, а также для предотвращения подмены имен файлов сценариев, вам необходимо добавить после строки **#!/bin/bash** параметр **--**, который позволит деактивировать механизм обработки параметров, благодаря чему командная оболочка не примет дополнительных параметров.

```
#!/bin/bash -
```

или

```
#!/bin/bash --
```

Любые аргументы, расположенные после последовательности символов **--** будут рассматриваться как имена файлов и аргументы. Аргумент **-** эквивалентен аргументу **--**.