3. ПРОГРАММИРОВАНИЕ В SHELL

3.1 Общие сведения

Синтаксис, возможности и сам набор команд в системах UNIX определяется интерпретатором команд (shell, оболочка). Существует несколько разновидностей интерпретаторов, входящих в любую поставку UNIX: Bourne shell (**sh**), Korn shell (**ksh**), C shell (**csh**), Bourne again shell (**bash**), Touch shell (**tcsh**) и др. Каждый из них имеет свои функциональные возможности. В процессе работы можно переключиться с одного интерпретатора на другой, запустив соответствующую программу. Оболочка по умолчанию задаётся пользователю при создании его учётной записи в системе. Командный интерпретатор, используемый по умолчанию в GNU/Linux – **bash**.

bash – совместимый с **sh** командный интерпретатор, выполняющий команды, поступающие на стандартный ввод или из файла. Файлы, содержащие команды для интерпретатора, называют *сценариями* (script). **bash** реализует многие полезные возможности интерпретаторов **ksh** и **csh**. **bash** соответствует спецификации POSIX 1003.2 для командных интерпретаторов и системных утилит.

Все встроенные команды **bash** описаны в разделе SHELL BUILTIN COMMANDS в конце справочного руководства **man bash**. Быстрый доступ к справочной информации по той или иной встроенной команде **bash** осуществляется при помощи встроенной команды **help**.

При входе пользователя в систему **bash** выполняет сначала общесистемный сценарий /etc/profile, а затем один из пользовательских сценариев настройки[6] ~/.bash_profile, ~/.bash_login или ~/.profile (какой найдёт первым). Экземпляр оболочки, запускаемый при входе пользователя в систему, называется *начальной оболочкой* (login shell). Когда начальная оболочка завершает работу (по [Ctrl]+[d] закончился стандартный ввод или пользователь выполнил встроенные команды **exit** или **logout**[7]), сеанс работы пользователя закрывается, при этом выполняется сценарий ~/.bash_logout. Если запускаются другие экземпляры оболочки, не являющиеся начальными (но интерактивными), выполняется сценарий ~/.bashrc. Когда оболочка запускается в неинтерактивном режиме (например, для выполнения пользовательского сценария), выполняется конфигурационный сценарий, имя которого хранится в переменной окружения BASH_ENV (если только эта переменная непустая).

В конфигурационном сценарии устанавливаются значения переменных окружения. Большинство из них принимает определённые значения при старте системы, однако, пользователь может по своему желанию переопределять эти значения. Кроме того, пользователь может переопределять команды, присваивая каким-либо вариантам команд или их комбинациям псевдонимы (aliases), тем самым создавая свою систему команд по вкусу.

3.2 Переменные окружения

Переменной окружения (параметром) называется некоторый объект, имеющий значение. Параметры образуют *среду окружения* (environment), переменные которой доступны каждой запущенной из оболочки команде. Особенности функционирования многих команд зависят от тех или иных переменных окружения. Переменная имеет имя, значение и опционально атрибуты. Переменная становится доступной после присвоения ей значения. Атрибуты можно назначить переменной при объявлении её во встроенной команде **declare** (или **typeset**, **export**, **readonly**, **local**). Значение переменной может быть получено при помощи специального преобразования: $NAME (NAME – имя переменной):

```
$ echo $SHELL - $BASH_VERSION
/bin/bash - 2.05b.0(1)-release
$ A=123
$ echo $A
123
$ A=A+1
$ echo $A
A+1
$ declare -i A=123
$ echo $A
123
$ A=A+1
$ echo $A
124
```

[6] Знаком ~ (тильда) обозначается домашний каталог пользователя (обычно /home/*user_login_name*).

[7] Команда **logout** не работает, если оболочка не является начальной.

Просмотреть значения всех определённых переменных можно с помощью встроенных команд **set** или **declare** без параметров. Удалить переменную из среды окружения можно при помощи встроенной команды **unset**.

Другой способ задать значение какой-либо переменной – использовать встроенную команду **read**. В этом случае значение переменной считывается со стандартного ввода. Если команде **read** не указать никакого параметра, считанное значение присваивается переменной REPLY.

Специальный тип переменных окружения – *позиционные параметры* (*аргументы командной строки*). Позиционные параметры обозначаются цифрами (1, 2, 3 и т. д.) Значения позиционным параметрам присваиваются при запуске оболочки либо при помощи команды **set**. Непосредственным присваиванием изменить значение позиционных параметров нельзя. При получении значения позиционного параметра, обозначенного двумя и более цифрами, его обозначение следует заключать в фигурные скобки.

Некоторые параметры имеют специальное назначение в оболочке. На них можно только ссылаться, но непосредственно их значение изменить нельзя. Такие специальные параметры обозначаются различными знаками, перечисленными в таблице 3.1.

```
$ set a b c d e f g h i j k l m n
$ echo $10 ${10}
a0 j
$ set "$@"
$ echo $#
14
$ set "$*"
$ echo $#
1
```

Таблица 3.1. Обозначение специальных параметров **bash**

#	Соответствует количеству позиционных параметров
* @	Оба параметра соответствуют всему множеству аргументов командной строки, т. е. при подстановке значения этих специальных параметров подставляются все заданные аргументы командной строки. Различие наблюдается при использовании этих параметров в кавычках (см. пример выше). В этом случае значение параметра * подставляется как одно слово (т. е. как "$1 $2 $3 …"), а при использовании параметра @ позиционные параметры подставляются как отдельные слова (т. е. как "$1" "$2" "$3"…)
?	Содержит статус выхода последней выполненной команды. Нулевой статус выхода соответствует успешному завершению программы
$	Содержит PID выполняющегося командного интерпретатора
0	Содержит имя выполняющейся оболочки или сценария

Интерпретатор **bash** версии 2.0 и новее поддерживает *параметры-массивы*. Массивы в **bash** являются одномерными. Для массивов нет явного ограничения на размер, также не является обязательным требование непрерывности нумерации элементов. Нет необходимости специально объявлять массив (хотя это можно сделать при помощи **declare**) – массив создаётся при задании значения любому его элементу. Элементы массивов нумеруются с нуля. Элементу массива можно присвоить значение при помощи команды:

*имя***[***индекс***]=***значение*.

Чтобы задать значения нескольким элементам массива, можно использовать следующий синтаксис:

*имя***=(***значение1значение2…***)**,

где вместо *значения-i* может быть использовано выражение:

[*индекс***]=***значение*.

A=(a b c)	эквивалентно	A[0]=a; A[1]=b; A[2]=c;
B=([3]=z [5]=q [13]=w)	эквивалентно	B[3]=z; B[5]=q; B[13]=w;

Для получения значения элемента массива используется следующая запись: **${***имя***[***индекс***]}**. Если в качестве индекса использован символ * или @, вместо этого выражения подставляются все элементы массива. Различие в использовании * или @ наблюдается только тогда, когда это выражение заключено в кавычки. В таком случае вместо **"${***имя***[*]}"** подставляется **"${***имя***[0]} ${***имя***[1]}** …**"**, а вместо **"${***имя***[@]}"** – **"${***имя***[0]}" "${***имя***[1]}"** … Вместо выражения **${#***имя***[***индекс***]}** подставляется длина элемента с указанным *индексом*. Если в качестве *индекса* указан символ * или @, подставляется количество элементов в массиве.

```
$ echo $BASH_VERSINFO
2
$ for i in "${BASH_VERSINFO[@]}"; do echo = $i =; done
```

```
= 2 =
= 05b =
= 0 =
= 1 =
= release =
= i586-mandrake-linux-gnu =
$ for i in "${BASH_VERSINFO[*]}"; do echo = $i =; done
= 2 05b 0 1 release i586-mandrake-linux-gnu =
```

Обратите внимание, что тело первого цикла выполняется 6 раз, а тело второго цикла – только 1 раз.

Использование имени массива без указания индекса подразумевает обращение к элементу с индексом 0.

Для удаления массива или его отдельных элементов можно использовать встроенную команду **unset**.

3.3 Псевдонимы

Псевдоним – это строка символов, подставляемая оболочкой вместо первого слова командной строки. Псевдонимы определяются при помощи встроенной команды **alias** и удаляются при помощи **unalias**. Чтобы просмотреть список определённых псевдонимов, запустите **alias** без параметров. Синтаксис команды:

alias name=value

Например:

```
$ alias DIR='ls –la'
$ DIR
итого 65188
drwxr-xr-x 3 pupkin pupkin     4096 Сен 6 21:36 ./
drwx------ 7 pupkin pupkin     4096 Сен 8 12:59 ../
drwxr-xr-x 2 pupkin pupkin     4096 Май 9 14:16 bmk/
-rw-r--r-- 1 pupkin pupkin  1033427 Фев 22 2003 etc.tgz
lrwxrwxrwx 1 pupkin pupkin       13 Сен 6 21:36 home -> /home/pupkin/
-rw-r--r-- 1 pupkin pupkin 65626430 Фев 22 2003 home.tgz
```

Установки переменных окружения и псевдонимы определены только для текущего экземпляра оболочки. Порождённые процессы и дочерние оболочки получают в свою среду окружения параметры, для которых задано свойство экспортирования (при помощи ключа *-x* в команде **declare** или команды **export**). Однако дочерние процессы не могут изменить среду окружения родительской оболочки, и все изменения переменных и определения псевдонимов теряются при выходе из дочернего процесса.

3.4 Шаблоны

В различных конструкциях **bash** могут использоваться *шаблоны*. В шаблоне любой символ, кроме перечисленных в таблице 3.2 специальных символов, сопоставляется с самим собой. Если требуется сопоставить себе один из специальных символов, следует применять экранирование.

Таблица 3.2. Специальные символы в шаблонах

*	Сопоставляется с любой подстрокой, в т. ч. с пустой
?	Сопоставляется с любым ровно одним символом
[…]	Сопоставляется с одним из символов, перечисленных в квадратных скобках. Можно использовать минус (–) для определения диапазона символов. Если первым символом внутри скобок является ! или ^, то шаблон сопоставляется с любым из символов, НЕ перечисленных в скобках. Внутри скобок могут использоваться идентификаторы классов символов: [:alnum:] [:alpha:] [:ascii:] [:blank:] [:cntrl:] [:digit:] [:graph:] [:lower:] [:print:] [:punct:] [:space:] [:upper:] [:word:] [:xdigit:]

3.5 Простая команда, конвейер, список, составные команды

Простая команда (simple command) может содержать следующие элементы (в порядке следования, разделённые пробелами):

- определения переменных окружения

  [Var1=Val2] [Var2=Val2] …
- имя команды (или путь к исполняемому файлу)
- аргументы команды (позиционные параметры)
- операции перенаправления

  [<file1] [>file2] …

Конец простой команды определяется одним из ограничителей:

|| | & && ; ;; ( ) конец строки

Примеры:

```
ls –l >test.ls
MANWIDTH=40 man bash
GREP_COLOR=\001 GREP_OPTIONS="--color=always" grep bash \
</etc/passwd
```

Переменные, значения которым заданы в составе простой команды, попадают только в её среду окружения и никак не влияют на среду окружения оболочки.

Синтаксис конвейера (pipeline):

ПростаяКоманда1 **|** *ПростаяКоманда2* [**|** …]

При этом стандартный вывод *ПростойКоманды1* подаётся на стандартный ввод *ПростойКоманды2* и т.д. Пример:

```
cat /etc/passwd | grep bash | tr : " "
```

Список (list) – это последовательность одного или нескольких конвейеров (или простых команд), разделённых знаками:

; & && ||

Эта последовательность может быть завершена точкой с запятой (;), амперсандом (&) или ограничителем «конец строки». Ограничители «точка с запятой» и «конец строки» предназначены для простого разделения команд в списке, такие команды выполняются последовательно. Там где стоит точка с запятой, может быть конец строки и наоборот. Если команда завершается амперсандом, она выполняется в фоновом режиме. Оболочка не ожидает её завершения и приступает к выполнению следующей команды в списке.

Управляющие операции списка “&&” и “| |” используются следующим образом:

команда1 **&&** *команда2*
команда1 **||** *команда2*

В первом случае *команда2* выполняется, только если *команда1* выполнилась успешно (вернула нулевой статус выхода). Во втором случае *команда2* выполняется, только если *команда1* выполнилась неуспешно (вернула ненулевой статус выхода). В примерах ниже использованы команды **`true`** и **`false`**, которые ничего не делают, а всегда возвращают определённый статус выхода: **`true`** возвращает статус 0 (успешный), а **`false`** возвращает статус 1 (неуспешный).

```
$ true && uname
Linux
$ false && uname
$ true || uname
$ false || uname
Linux
```

Составные команды (compound commands):

for *NAME* **in** *WORDS* **;** **do** *LIST* **;** **done**

В цикле выполняются команды из списка *LIST*. В каждой итерации цикла параметру *NAME* присваивается очередное слово из списка *WORDS* (параметры, разделённые пробелами). Цикл завершается, когда все слова из списка *WORDS* исчерпаны.

```
$ for A in 1 2 3 ; do echo $A ; done
1
2
3
```

for ((*EXPR1* **;** *EXPR2* **;** *EXPR3* **))** **;** **do** *LIST* **;** **done**

Перед выполнением цикла по правилам арифметических выражений вычисляется *EXPR1*. Перед выполнением очередной итерации цикла проверяется значение арифметического выражения *EXPR2*. Если его значение отлично от нуля, выполняется список *LIST* и вычисляется выражение *EXPR3*. Когда значение выражения *EXPR2* становится равным 0, цикл завершается.

```
$ for ((A=100; A>=96; A--)); do echo $A; done
100
99
98
97
96
```

if *LIST* **;** **then** *LIST* **;** [**elif** *LIST* **;** **then** *LIST* **;**] …
[**else** *LIST* **;**] **fi**

Выполняется список, стоящий после **if**. Если список выполняется успешно (статус выхода 0), то выполняется список, стоящий после **then**. В противном случае выполняется список, стоящий после **elif** или после **else**.

select *NAME* **in** *WORDS* **; do** *LIST* **; done**

На стандартный вывод ошибок выдаётся пронумерованный список слов *WORDS*. Со стандартного ввода в параметр REPLY читается строка, а затем выполняется список *LIST*. При этом если введённая строка содержит номер одного из указанных слов, то параметру *NAME* присваивается это слово, в противном случае *NAME* остаётся пустой. Эта последовательность действий выполняется до тех пор, пока не будет выполнена встроенная команда **break** или не закончится стандартный ввод.

```
$ select R in a b c ; do echo Reply:[$REPLY] R:[$R]
> if [ "$R" == a ]; then break; fi
> done
1) a
2) b
3) c
#? q
Reply:[q] R:[]
#? 2
Reply:[2] R:[b]
#? 1
Reply:[1] R:[a]
```

case *WORD* **in** *PATTERN* [**|** *PATTERN*] **)** *LIST* **;;** … **esac**

Слово *WORD* последовательно сравнивается со всеми шаблонами *PATTERN*. При нахождении соответствия, выполняется список *LIST*, стоящий после подходящего шаблона. Дальнейшие сопоставления не производятся.

```
$ case $OSTYPE in
> linux* | freebsd* ) uname;;
> *) echo Unknown system;;
> esac
Linux
```

while *LIST1* **; do** *LIST2* **; done**
until *LIST1* **; do** *LIST2* **; done**

Список *LIST2* выполняется до тех пор, пока статус выхода *LIST1* не станет отличен от нуля (для **while**) или не станет равным нулю (для **until**).

Внутри циклических команд могут быть использованы встроенные команды **break** и **continue**. Команда **break** немедленно завершает цикл, а команда **continue** вызывает немедленный переход к очередной итерации.

3.6 Преобразования

При обработке командной строки **bash**, встретив неэкранированные специальные символы, выполняет определённые преобразования.

При *преобразовании скобок* (brace expansion) выражение вида «a**{**d,c,b**}**e» заменяется на «ade ace abe». Это преобразование может быть вложенным. Сортировка по алфавиту не выполняется – сохраняется порядок слева направо.

Примеры:

```
$ echo test{1,2,3,4}
test1 test2 test3 test4
$ ls /{bin,sbin}/*sh
/bin/bash /bin/rbash /bin/tcsh /bin/csh /bin/sh
```

Если слово начинается с тильды (~), выполняется *преобразование тильды* (tilde expansion): вместо одиночной тильды подставляется путь к домашнему каталогу текущего пользователя; вместо выражения ~*username* – путь к домашнему каталогу указанного пользователя; вместо ~+ подставляется значение переменной PWD (текущий каталог); вместо ~– подставляется значение переменной OLDPWD (каталог, который был текущим до выполнения последней команды **cd**).

Преобразование параметров (parameter expansion):

Таблица 3.3. Преобразование параметров

$*NAME* или **${***NAME***}**	Вместо выражения подставляется значение параметра *NAME*. Фигурные скобки обязательны, только если параметр является позиционным с номером больше 9 или после имени параметра следует символ, который может быть интерпретирован как часть имени, но ею не является
${!*NAME***}**	*Косвенная подстановка.* Подставляется значение параметра, имя которого хранится в *NAME*
${!***PREFIX****}** или **${!*PREFIX***@}**	Подставляются все имена переменных, начинающиеся с указанного префикса
${*NAME***:-***WORD***}**	Если параметр *NAME* не установлен либо пуст, вместо выражения подставляется *WORD*, в противном случае подставляется значение параметра
${***NAME***:***OFS***}** **${*NAME***:***OFS***:***LEN***}**	*Подстановка подстроки.* Вместо выражения подставляется не более *LEN* символов из значения параметра *NAME*, начиная с символа со смещением *OFS* (смещения от 0). Если смещение отрицательно, символы отсчитываются от конца строки. Вместо *NAME* может быть * или @, тогда подставляются позиционные параметры, начиная с параметра с индексом *OFS* в количестве не более *LEN* штук. То же самое касается массивов с индексом * или @
${#*NAME***}**	Подставляется размер значения параметра *NAME* в символах. Если в качестве *NAME* использован массив, см. комментарии в п. 3.2 Переменные окружения
${***NAME***#***PATTERN***}** **${*NAME***##***PATTERN***}**	Вместо выражения подставляется значение параметра *NAME*, при этом из начала значения параметра вырезается минимальная (для #) или максимальная (для ##) подстрока, соответствующая шаблону *PATTERN*
${***NAME***%***PATTERN***}** **${*NAME***%%***PATTERN***}**	Вместо выражения подставляется значение параметра *NAME*, при этом из конца значения параметра вырезается минимальная (для %) или максимальная (для %%) подстрока, соответствующая шаблону *PATTERN*
${***NAME***/***PTRN***/***STR***}** **${*NAME***//***PTRN***/***STR***}**	Вместо выражения подставляется значение параметра *NAME*, в котором первая подстрока (для /) или все подстроки (для //) максимальной длины, соответствующие шаблону *PTRN*, заменены на *STR*. Если *STR* отсутствует, подстроки удаляются

Следующие примеры позволяют наблюдать разницу между различными вариантами преобразований:

```
$ ZZZ=1qwerty2qwerty3qwerty
$ echo ${ZZZ:14}
3qwerty
$ echo ${ZZZ:7:7}
2qwerty
$ echo ${ZZZ:(-3):2}
rt
$ echo ${ZZZ#1q*rt}
y2qwerty3qwerty
$ echo ${ZZZ##1q*rt}
y
$ echo ${ZZZ%qw*ty}
1qwerty2qwerty3
$ echo ${ZZZ%%qw*ty}
1
$ echo ${ZZZ/q/\'}
1'werty2qwerty3qwerty
$ echo ${ZZZ//q/\'}
1'werty2'werty3'werty
```

Подстановка команды (command substitution) подразумевает, что вместо выражения будет подставлен текст, выданный на стандартный вывод командой, содержащейся в выражении. Есть две равнозначные формы синтаксиса подстановки команды[8]:

`` `команда` `` $(команда)

Специальный случай подстановки команды: вместо **$(cat** *файл***)** можно использовать **$(<***файл***)**, т. е. вместо выражения будет подставлено содержимое указанного *файла*.

Арифметическая подстановка (arithmetic expansion) предназначена для вычисления арифметических выражений и подстановки результата: **$((***выражение***))**. Например:

```
$ echo 5+6=$((5+6))
5+6=11
```

Преобразование пути (pathname expansion) выполняется, если после всех преобразований слово содержит астериск (*), знак вопроса (?) или квадратные скобки [] – специальные символы шаблонов (см. п. 3.4). В таком случае слово считается шаблоном имени файла и заменяется на все имена файлов, удовлетворяющие этому шаблону. Имена файлов подставляются в алфавитном порядке. Если не один файл этому шаблону не соответствует, то в зависимости от настроек **bash** слово может быть заменено на пустую строку или оставлено в неизменном виде. Примеры:

```
$ echo ~/.bash*
/home/pupkin/.bash_history /home/pupkin/.bash_logout /home/pupkin/.bash_profile
/home/pupkin/.bashrc
$ ls /etc/[knqwy]*
/etc/krb5.conf     /etc/nsswitch.conf  /etc/warnquota.conf  /etc/ytalkrc
/etc/ksysguarddrc  /etc/qt.fontguess   /etc/wgetrc
```

3.7 Арифметические выражения

Арифметические выражения используются в арифметических подстановках, а также во встроенной команде **let**. Вместо команды **let "***выражение***"** можно использовать следующий синтаксис:

((*выражение***))**

В любом случае такая команда вычисляет выражение по правилам для арифметических выражений и возвращает успешный статус выхода (0), если результат выражения отличен от нуля, и неуспешный статус выхода (1), если результат арифметического выражения – нуль. При вычислении арифметических выражений **bash** оперирует только целыми числами с разрядностью, соответствующей архитектуре машины (32 или 64 бита). Воспринимаются те же арифметические операции, что и в языке Си, с таким же порядком приоритета:
`++ -- ! ~ **` (возведение в степень) `* / % + – << >> <= >= < > == != & ^ | && || ?: =` (и др. виды присваивания).

В качестве операндов могут стоять переменные окружения без использования преобразования параметров. Константы, начинающиеся с 0, считаются восьмеричными. Константы, начинающиеся с 0x или 0X, считаются шестнадцатеричными. Константы, начинающиеся с *num#* (где num=2…64), считаются заданными в системе исчисления по основанию *num*. В противном случае константа считается десятичной.

Примеры:

```
$ echo $((5+6))
11
$ ((10>20))
$ echo $?
1
$ ((10<20))
$ echo $?
0
$ A=2#111
$ let A++
$ printf %o\\n $A
10
```

3.8 Условные выражения

Условные выражения указываются в составном операторе **[[**…**]]** и во встроенной команде **test** (или в её синониме **[…]**). Условные выражения используются для проверки атрибутов файлов, а также сравнения строк и чисел. Если условное выражение истинно, команда **[[…]]** или **test** даёт успешный статус выхода (0). Ложному условному выражению соответствует неуспешный статус выхода (1).

---

[8] Обратите внимание, в первом случае используются не прямые апострофы (одинарные кавычки), а обратные (на клавиатуре располагаются, как правило, рядом с [Tab]).

Полностью с возможностями условных выражений можно ознакомиться в **man bash**. Некоторые возможные операции перечислены в таблице 3.4.

Таблица 3.4. Опции и операнды команды **test**

-e *файл*	Истинно, если *файл* существует
-f *файл*	Истинно, если *файл* существует и является обычным файлом
-d *файл*	Истинно, если *файл* существует и является директорией
-r *файл*	Истинно, если *файл* существует и доступен для чтения
-w *файл*	Истинно, если *файл* существует и доступен для записи
-x *файл*	Истинно, если *файл* существует и является исполняемым
-s *файл*	Истинно, если *файл* существует и имеет ненулевой размер
-z *строка*	Истинно, если длина *строки* равна нулю (строка пустая)
-n *строка*	Истинно, если длина *строки* не равна нулю (строка непустая)
строка1 == *строка2* *строка1* != *строка2*	Сравнение строк на равенство или неравенство
арг1 ОР арг2	Сравнение численных аргументов

ОР может быть: **-eq** (равно), **-ne** (не равно), **-lt** (меньше), **-le** (меньше или равно), **-gt** (больше), **-ge** (больше или равно).

Обратите внимание, что в примерах вокруг квадратных скобок стоят пробелы. Они необходимы, т. к. символы “[” и “]” не являются *метасимволами* (символами, разделяющими слова).

```
$ [[ 10 –gt 20 ]]
$ echo $?
1
$ [[ 10 –lt 20 ]]
$ echo $?
0
$ if test $RANDOM –gt 15000
> then echo big
> else echo small ; fi
ok
$ if test $RANDOM –gt 15000
> then echo big
> else echo small ; fi
try again
$ test –r /etc/passwd && echo file is readable
file is readable
$ [ -r /etc/shadow ] && echo file is readable
$ [ -r /etc/shadow ] || echo file is NOT readable
file is NOT readable
```

3.9 Сценарии командного интерпретатора

Командный интерпретатор может обрабатывать команды не только со стандартного ввода (из командной строки), но и из текстового файла – *сценария*. Принято помещать в первую строку сценария путь к интерпретатору в виде:

```
#!путь опции
```

Это правило справедливо не только для **bash**, но и для других языков описания сценариев, например, Perl. Строки, начинающиеся с октоторпа (#), считаются комментариями и игнорируются (как и пустые строки). Запуск сценария на выполнение может осуществляться несколькими способами. Если файл сценария является исполняемым, то сценарий можно запустить, просто указав его имя в качестве команды. Обратите внимание, что система ищет команды только в каталогах, указанных в параметре PATH, в противном случае надо писать путь к исполняемому файлу. Для запуска сценария можно также использовать встроенные команды **source** или **.** (точка). В таком случае имя сценария передаётся как аргумент командной строки.

Пример сценария:

```
#!/bin/bash
num=${1:-0}
```

```
a='#'
for ((i=0; i<num; i++))
do
       echo $a
       a="#$a"
done
```

Если сценарий назвать myscript.sh, то в работе он будет выглядеть следующим образом:

```
$ source myscript.sh 3
#
##
###
```

Контрольные вопросы и задания

1. Выясните назначение следующих переменных: BASH, BASH_VERSINFO, BASH_VERSION, GROUPS, HOSTNAME, HOSTTYPE, MACHTYPE, OSTYPE, PPID, PWD, RANDOM, REPLY, SHLVL, UID, HOME, LANG, PATH, PS1, PS2, PS3. Запишите значение перечисленных переменных для того экземпляра оболочки, в котором вы работаете.
2. Просмотрите файлы системных настроек в вашем домашнем каталоге. Измените вид строки приглашения для вашей среды так, чтобы она содержала время.
3. Назначьте командам **cp**, **mv** и **rm** в качестве псевдонимов их аналоги из DOS.
4. В разделе «Выполнение команды» (COMMAND EXECUTION) справочного руководства по **bash** выясните приоритет, в котором выполняется поиск соответствия имени команды. Выясните назначение встроенных команд **builtin**, **command**, **enable** и **type**.
5. При помощи преобразования **${!***PREFIX****}** и цикла **for** выведите значения всех переменных, имена которых начинаются на BASH.
6. Что помещается в переменную A при присваивании `A=`ls`?` Что помещается в переменную A при присваивании `A=`<~/.bash_history`?`
7. Выясните назначение встроенной команды **ulimit**.