

man ksh(1)

ИМЯ

ksh, **rksh** - KornShell, стандартный/ограниченный командный интерпретатор и язык программирования

СИНОПСИС

```
/usr/bin/ksh [ $\pm$ abCefhikmnoprstuvx] [ $\pm$ о опция] ... [аргумент ...]  
/usr/bin/ksh -c [ $\pm$ abCefhikmnoprstuvx] [ $\pm$ о опция] ...  
    строка_команды [имя_команды [аргумент ...]]  
/usr/xpg4/bin/sh [ $\pm$ abCefhikmnoprstuvx] [ $\pm$ о опция] ... [аргумент ...]  
/usr/xpg4/bin/sh -c [ $\pm$ abCefhikmnoprstuvx] [ $\pm$ о опция] ...  
    строка_команды [имя_команды [аргумент ...]]  
/usr/bin/rksh [ $\pm$ abCefhikmnoprstuvx] [ $\pm$ о опция] ... [аргумент ...]  
/usr/bin/rksh -c [ $\pm$ abCefhikmnoprstuvx] [ $\pm$ о опция] ...  
    строка_команды [имя_команды [аргумент ...]]
```

ОПИСАНИЕ

Команда **/usr/xpg4/bin/sh** идентична **/usr/bin/ksh**, команде и языку программирования, выполняющему команды, прочитанные с терминала или из файла. **rksh** - ограниченная версия командного интерпретатора **ksh**; он используется для настройки регистрационных имен и сред выполнения, возможности которых контролируются больше, чем при использовании стандартного командного интерпретатора. Описание аргументов командного интерпретатора см. в разделе "**Вызов**" ниже.

Определения

Метасимвол - это один из следующих символов:

; & () < > НОВАЯ_СТРОКА ПРОБЕЛ ТАБУЛЯЦИЯ

Пробельный символ (blank) - это символ табуляции или пробел. *Идентификатор* - последовательность букв, цифр или символов подчеркивания, начиная с буквы или символа подчеркивания. Идентификаторы используются как имена для функций и переменных. *Слово* - это последовательность символов, разделенных одним или несколькими незамаскированными метасимволами.

Команда - это последовательность символов, соответствующая синтаксису языка командного интерпретатора. Командный интерпретатор читает каждую команду и выполняет желаемое действие либо непосредственно, либо вызывая отдельные утилиты. *Специальная команда* - это команда, выполняемая командным интерпретатором без создания отдельного процесса.

За исключением описанных в документации побочных эффектов, большинство специальных команд могут быть реализованы как отдельные утилиты.

Команды

Простая команда - это последовательность разделенных пробельными символами слов, перед которой может список присвоений переменным. (См. раздел "**Среда**" ниже.) Первое слово задает имя команды, которую надо выполнить. Кроме указанных ниже случаев, остальные слова передаются как аргументы вызванной команде. Имя команды передается как аргумент 0 (см. **exec(2)**). *Значение* простой команды - это ее статус выхода, если она завершилась нормально. Если же она завершается ненормально вследствие получения сигнала, значением будет номер сигнала плюс 128. Список значений сигналов см. на странице справочного руководства **signal(3HEAD)**. Очевидно, нормальные значения статуса выхода от 129 до 255 нельзя отличить от ненормального выхода, вызванного получением сигнала с номером от 1 до 127.

Конвейер - это последовательность одной или нескольких команд, разделенных символом |. Стандартный выходной поток каждой команды, кроме последней, соединяется с помощью **pipe(2)** со стандартным входным потоком следующей команды. Каждая команда запускается как отдельный процесс; командный интерпретатор ждет завершения последней команды. *Статус выхода конвейера* - это статус выхода его последней команды.

Список - это последовательность одного или нескольких конвейеров, разделенных символами ;, &, && или ||, необязательно завершенная символами ;, & или |&. Из этих пяти символов, ;, & и |& имеют равный

приоритет, который ниже, чем приоритет символов **&&** и **||**. Символы **&&** и **||** также имеют равный приоритет. Точка с запятой (**;**) вызывает последовательное выполнение предшествующего конвейера; амперсанд (**&**) вызывает асинхронное выполнение предшествующего конвейера (т.е., командный интерпретатор не ждет, пока этот конвейер завершит работу). Символ **|&** вызывает асинхронное выполнение предыдущей команды или конвейера так, что с родительским командным интерпретатором устанавливается двусторонний канал обмена данными (**pipe**).

Стандартный входной и выходной поток запущенной команды может быть прочитан и записан из родительского командного интерпретатора с помощью опции **-p** специальных команд **read** и **print**, описанных в разделе "**Специальные команды**". Символ **&&** (**||**) вызывает выполнение следующего за ним списка только если предшествующий конвейер возвращает не 0 (ненулевое значение). Произвольное количество переводов строк может идти в списке вместо точки с запятой в качестве разделителя команд. Командой является простая команда или одна из следующих конструкций. Если не сказано иначе, значение, возвращаемое командой, является значением последней выполненной простой команды.

for *идентификатор* [**in** *слово ...*] ; **do** *список* ; **done**

При каждом выполнении команды **for**, *идентификатор* устанавливается равным следующему *слову* из списка слов. Если конструкция **in слово ...** не указана, команда **for** выполняет *список do* один раз для каждого установленного позиционного параметра (см. раздел "**Подстановка параметров**" ниже). Выполнение завершается, когда в списке больше не остается слов.

select *идентификатор* [**in** *слово ...*] ; **do** *список* ; **done**

Команда **select** выдает в стандартный поток ошибок (дескриптор файла 2) набор слов с числом перед каждым из них. Если конструкция **in слово ...** не указана, используются позиционные параметры (см. раздел "**Подстановка параметров**" ниже). Выдается приглашение **PS3** и строка читается из стандартного входного потока. Если эта строка состоит из номера одного из перечисленных *слов*, то значение переменной *идентификатор* устанавливается равным слову, которое соответствует этому номеру. Если эта строка пуста, список выбора выдается еще раз. В противном случае, значение переменной *идентификатор* устанавливается равным **NULL**. (По поводу **NULL** см. раздел "**Интерпретация пробельных символов**"). Содержимое строки, прочитанной из стандартного входного потока, сохраняется в переменной **REPLY**. *Список* выполняется для каждого выбора, пока не произойдет принудительный останов (**break**) или не будет обнаружен символ конца файла (**EOF**). Если переменная **REPLY** устанавливается в **NULL** при выполнении *списка*, то список выбора выдается перед приглашением **PS3** для следующего выбора.

case *слово* **in** [*шаблон* [| *шаблон*]) *список* ;;] ... **esac**

Команда **case** выполняет *список*, связанный с первым *шаблоном*, соответствующим *слову*. Форма шаблонов - такая же, как используется при генерации имен файлов (см. раздел "**Генерация имен файлов**" ниже).

if *список* ; **then** *список* ; [**elif** *список* ; **then** *список* ; ...] [**else** *список* ;] **fi**

Список, следующий за **if** выполняется и, если он возвращает статус выхода 0, выполняется *список*, следующий за первым **then**. В противном случае, выполняется *список*, следующий после **elif** и, если его значение - 0, выполняется список после следующего **next**. Если ни для одного из **elif** это не оказалось верным, выполняется *список* после **else**. Если конструкция **else** не указана или ни один *список* после **then** не был выполнен, команда **if** возвращает статус выхода 0.

while *список* ; **do** *список* ; **done**

until *список* ; **do** *список* ; **done**

Команда **while** повторно выполняет *список* после **while** и, если статус выхода последней команды в этом *списке* - 0, выполняет список после **do**; в противном случае, цикл прекращается. Если ни одна команда в списке **do** не выполнена, то команда **while** возвращает статус выхода 0; **until** можно использовать вместо **while** для применения отрицания к условию завершения цикла.

(*список*)

Выполнить *список* в отдельной среде. Учтите, что если для обеспечения вложенности необходимо указать две открывающие скобки подряд, между ними надо вставить пробел, чтобы избежать арифметического вычисления, как описано ниже.

{*список*}

Список просто выполняется. Учтите, что в отличие от метасимволов (**(и)**), **{ и }** являются зарезервированными словами и, чтобы быть распознанными, должны идти в начале строки или после **;**.

[[*выражение*]]

Вычисляет выражение и возвращает статус выхода 0, когда выражение истинно. Описание выражения см. в разделе "**Условные выражения**" ниже.

```
function идентификатор { список ; }  
идентификатор() { список ; }
```

Определяет функцию, на которую можно сослаться по **идентификатору**. Тело функции - это **список** команд между { и }. (См. раздел "**Функции**" ниже).

time *конвейер*

Конвейер выполняется и время выполнения, а также процессорное время в режиме пользователя и в режиме ядра (system time) выдается в стандартный поток ошибок.

Следующие зарезервированные слова распознаются только как первое слово команды, если они не замаскированы:

!		if	then	else	elif	fi	case
esac		for	while	until	do	done	{ }
function	select	time	[[]]				

Комментарии

Слово, начинающееся с #, вызывает игнорирование этого и всех следующих символов до новой строки.

Псевдонимы

Первое слово каждой команды заменяется текстом *псевдонима*, если соответствующий псевдоним был определен. Имя псевдонима состоит из любого количества символов, исключая метасимволы, символы маскировки (quoting characters), символы подстановки имен файлов, символы подстановки параметров и команд, а также символ =. Строка замены может содержать любой допустимый сценарий командного интерпретатора, включая перечисленные выше метасимволы. Первое слово каждой команды в замененном тексте, отличающееся от заменяемого, будет проверяться на соответствие псевдонимам. Если последний символ значения псевдонима - пробельный, то следующее за псевдонимом слово также будет проверяться на предмет подстановки псевдонима. Псевдонимы можно использовать для переопределения специальных встроенных команд, но не перечисленных выше зарезервированных слов. Псевдонимы можно создавать, просматривать и экспортировать с помощью команды **alias**, а удалять - с помощью команды **unalias**. Экспортированные псевдонимы действуют для сценариев, вызываемых по имени, но должны быть переинициализированы для отдельных вызовов командного интерпретатора (см. раздел "**Вызов**" ниже). Чтобы предотвратить бесконечные циклы рекурсивной подстановки псевдонимов, если командный интерпретатор сейчас не обрабатывает псевдоним с тем же именем, слово будет заменено значением псевдонима; в противном случае, оно заменено не будет.

Подстановка псевдонимов выполняется при чтении сценариев, а не при их выполнении. Поэтому, чтобы псевдоним был учтен, команда определения псевдонима должна быть выполнена до чтения команды, в которой используется псевдоним.

Псевдонимы часто используются для сокращения полных имен. Опция команды задания псевдонима позволяет автоматически устанавливать в качестве значения псевдонима полное имя соответствующей команды. Эти псевдонимы называются *отслеживаемыми* (tracked aliases). Значение отслеживаемого псевдонима определяется при первом поиске соответствующей команды и становится неопределенным при каждом изменении значения переменной **RATH**. Эти псевдонимы остаются отслеживаемыми, так что последующая ссылка переопределит значение. Несколько отслеживаемых псевдонимов скомпилированы ("защиты") в командный интерпретатор. Опция **-h** команды **set** преобразует имя каждой использованной команды в отслеживаемый псевдоним.

Следующие экспортированные псевдонимы скомпилированы (и встроены) в командный интерпретатор, но могут быть отменены или переопределены:

```
autoload='typeset -fu'  
false='let 0'  
functions='typeset -f'  
hash='alias -t'  
history='fc -l'  
integer='typeset -i'  
nohup='nohup '  
r='fc -e -'  
true=':'  
type='whence -v'
```

Далее следует пример, связанный с завершающими пробельными символами и зарезервированными словами. Если пользователь набрал:

```
$ alias foo="/bin/ls "  
$ alias while="/"
```

эффектом выполнения:

```
$ while true
> do
> echo "Hello, World"
> done
```

будет выдача на экран бесконечной последовательности строк **Hello, World**. Но, если пользователь наберет:

```
$ foo while
```

результатом будет листинг каталога /, выданный командой **ls**. Поскольку подставленное вместо псевдонима **foo** значение завершается пробелом, подстановка псевдонима выполняется и в следующем слове.

Следующее слово, **while**, также является псевдонимом, поэтому вместо него тоже подставляется значение.

Поскольку в этой позиции слова команда не предполагается, оно не распознается как зарезервированное слово.

Если пользователь наберет:

```
$ foo; while
```

while интерпретируется как обычно, как зарезервированное слово.

Подстановка тильды

После того, как выполнена подстановка псевдонимов, каждое слово проверяется на то, не начинается ли оно с незамаскированного символа ~. Если да, то слово, вплоть до символа /, проверяется на совпадение с именем пользователя. Если такой пользователь найден, символ ~ и соответствующее регистрационное имя заменяется начальным каталогом этого пользователя. Это называется *подстановкой тильды*. Если соответствие не найдено, исходный текст отстает без изменений. Отдельный символ ~ или перед / заменяется значением **\$HOME**. Символ ~, за которым идет + или -, заменяется значением **\$PWD** и **\$OLDPWD**, соответственно.

Кроме того, попытка подстановки тильды выполняется, когда значение, присваиваемое переменной, начинается с ~.

Замена тильды

Тильда-префикс состоит из незамаскированного символа тильды в начале слова, после которого идут все символы до незамаскированной косой черты в слове или все символы в слове, если косой в нем нет. В операторе присвоения можно использовать несколько тильда-префиксов: в начале слова (т.е., после символа равно в операторе присвоения), после незамаскированного двоеточия, или и там, и там. Тильда-префикс в операторе присвоения завершается первым незамаскированным двоеточием или косой. Если ни один из символов в тильда-префиксе не замаскирован, символы тильда-префикса после тильды рассматриваются как одно из возможных регистрационных имен их базы данных пользователей. Переносимое регистрационное имя не может содержать символы, не входящие в набор, представленный в описании переменной среды **LOGNAME**. Если регистрационное имя - пустое (т.е. тильда-префикс содержит только тильду), тильда-префикс будет заменен значением переменной **HOME**. Если переменная **HOME** не определена, результаты не регламентируются. В противном случае, тильда-префикс заменяется полным именем начального каталога соответствующего пользователя, регистрационное имя которого получается с помощью функции **getpwnam**. Если система не распознает регистрационное имя, результаты не определены. Замена тильды обычно происходит только в начале слов, но есть также исключение, основанное на исторической практике. Здесь:

```
PATH=/posix/bin:~dkg/bin
```

замена тильды выполняется, поскольку тильда идет после двоеточия и ни один из имеющих значение символов не замаскирован. Было высказано соображение в пользу запрета этого поведения, поскольку в любом из следующих примеров подстановка тоже вполне обоснована:

```
PATH=$(printf %s ~karels/bin : ~bostic/bin)
```

```
for Dir in ~maat/bin ~srb/bin .
```

```
do
```

```
PATH=${PATH:+$PATH;}$Dir
```

```
done
```

В первой команде явные двоеточия используются для каждого каталога. Во всех случаях, командный интерпретатор выполняет замену тильды для каждого каталога, поскольку все они являются отдельными словами для командного интерпретатора.

Учтите, что выражения в операндах, такие как:

```
make -k mumble LIBDIR=~chet/lib
```

не рассматриваются в качестве присвоений переменным командного интерпретатора и замена тильды не выполняется (если только команда сама этого не делает; **make** - не делает).

Специальная последовательность **\$~** была выделена для будущих реализаций, чтобы можно было выполнить замену тильды в любом слове.

Из-за требования, что слово не должно быть замаскировано, следующие конструкции не эквивалентны; только в последней будет выполнена замена тильды:

```
\~hlj/ ~\hj/ ~"hlj"/ ~hlj\ ~hlj/
```

Результаты указания тильды с неизвестным регистрационным именем не определены, поскольку конструкции KornShell `~+` и `~` используются, но, в общем случае, передача некорректного регистрационного имени с тильдой - это ошибка. Последствия не установленной переменной **HOME** не определяются, поскольку некоторые ранее использовавшиеся командные интерпретаторы считают это ошибкой.

Подстановка результатов выполнения команды

Стандартный выходной поток команды, взятой в круглые скобки, перед которыми идет символ доллара (т.е. **\$**(команда)), или окруженной парой символов обратного апострофа (```), может быть использован как слово или часть слова; завершающие символы новой строки при этом удаляются. Во второй (устаревшей) форме, строка между апострофами обрабатывается в поисках специальных маскирующих символов перед выполнением команды. (См. раздел "**Маскировка**" ниже.) Подстановку результатов выполнения команды **\$**(cat файл) можно заменить эквивалентной, но быстрее выполняющейся конструкцией **\$**(<файл).

Подстановка результатов выполнения большинства специальных команд, не выполняющих перенаправления ввода-вывода, производится без создания отдельного процесса.

Подстановка результатов выполнения команды позволяет результат команды подставить вместо ее имени. Она выполняется, когда команда указана следующим образом:

\$(команда)

или (версия в обратных апострофах):

``команда``

Командный интерпретатор будет выполнять подстановку результатов выполнения команды, выполняя ее в среде порожденного интерпретатора и заменяя обращение к команде (текст команды плюс окружающие ее **\$** () или обратные апострофы) стандартным выходным потоком команды, с удалением последовательностей из одного или нескольких символов новой строки в конце. Встроенные символы новой строки, не следующие непосредственно в конце, не будут удаляться; однако, они могут рассматриваться как разделители полей и удаляться при делении строки на поля, в зависимости от значения **IFS** и действующей маскировки.

При использовании подстановки результатов выполнения команды с помощью обратных апострофов, обратная косая будет использоваться буквально, если только она не используется в следующем контексте:

\$` (символ доллара, обратный апостроф, обратная косая). Поиск соответствующего обратного апострофа завершается успешно при нахождении первого же символа обратного апострофа, перед которым не идет обратная косая; в ходе этого поиска, если незамаскированный обратный апостроф обнаруживается в комментарии, в конструкции "документ здесь", во встроенной подстановке результатов выполнения команды в виде **\$**(команда) или в строке в кавычках, результаты непредсказуемы. Строка в одиночных или двойных кавычках, начинающаяся, но не заканчивающаяся последовательностью ``...``, приводит к непредсказуемым результатам.

В форме **\$**(команда) все символы после открывающей круглой скобки и до соответствующей закрывающей образуют команду. В качестве команды можно использовать любой сценарий командного интерпретатора, кроме следующих:

- Сценарий, состоящий исключительно из перенаправлений, дает неопределенные результаты.
- См. ограничение на одиночные порожденные командные интерпретаторы, описанное ниже.

Результаты подстановки команды не будут подвергаться делению на поля и подстановке полных путей для дальнейшей замены тильды, замены параметров, подстановки результатов выполнения команд или замены арифметических выражений. Если подстановка команды происходит в двойных кавычках, над ее результатом подстановка команд выполняться не будет.

Подстановка результатов выполнения команды может быть вложенной. Чтобы указать вложение в форме с обратными апострофами, приложение должно предварять внутренние обратные апострофы обратными косыми; например:

``\`команда`\``

Форма **\$()** подстановки результатов выполнения команды решает проблему несогласованного поведения при использовании обратных апострофов. Например:

Команда	Результат
<code>echo "\$x"</code>	<code>\\$x</code>
<code>echo `echo "\$x"`</code>	<code>\$x</code>
<code>echo \$(echo "\$x")</code>	<code>\\$x</code>

Кроме того, синтаксис с обратными апострофами имеет исторические ограничения на содержание встроенной команды. Тогда как новая форма `$()` может обрабатывать допустимый вложенный сценарий любого вида, форма с обратными апострофами не может обрабатывать некоторые вполне допустимые сценарии, включающие обратные апострофы. Например, эти во всех отношениях допустимые вложенные сценарии не работают в левом столбце, но работают в правом:

```
echo `                | echo $(
cat <<eof              | cat <<eof
"документ здесь" с `   | "документ здесь" с )
eof                    | eof
`                      | )

echo `                | echo $(
echo abc # комментарий с ` | echo abc # комментарий с )
`                      | )

echo `                | echo $(
echo "'              | echo ')
`                      | )
```

Из-за этого несогласованного поведения, вариант подстановки команды в обратных апострофах не рекомендуется для новых приложений с вложенными подстановками команд или попытками встраивания сложных сценариев.

Если подстановка результатов выполнения команды состоит из единственного вложенного вызова командного интерпретатора, например:

```
$( команда )
```

переносимое приложение должно разделять `$()` и `(` на две лексемы (т.е. разделять их пробельным символом). Это требуется во избежание любой путаницы с заменой арифметических выражений.

Замена арифметических выражений

Арифметическое выражение, взятое в двойные круглые скобки, перед которыми идет знак доллара (`$ ((арифметическое_выражение))`) заменяется значением указанного арифметического выражения. Замена арифметических выражений обеспечивает механизм вычисления и подстановки значений арифметических выражений. При этом используется следующий формат:

```
$(выражение)
```

Выражение обрабатывается так, как если бы оно было в двойных кавычках, за исключением того, что двойная кавычка в выражении не обрабатывается специальным образом. Командный интерпретатор будет обрабатывать все лексемы в выражении, выполняя в них замену параметров, подстановку результатов выполнения команд и удаление кавычек.

Затем командный интерпретатор вычисляет арифметическое выражение и подставляет его значение. Арифметическое выражение обрабатывается в соответствии с правилами ISO C, со следующими исключениями:

- Поддерживается только целочисленная арифметика.
- Оператор `sizeof()`, а также префиксные и постфиксные операторы `++` и `--` не поддерживаются.
- Операторы выбора, итерации и перехода не поддерживаются.

В качестве расширения, командный интерпретатор может распознавать и другие арифметические выражения. Если выражение недопустимо, при вычислении значения произойдет сбой, и командный интерпретатор выдаст сообщение о причине сбоя в стандартный выходной поток.

Простой пример использования замены арифметического выражения:

```
# повторить команду 100 раз
x=100
while [ $x -gt 0 ]
do
    команда
    x=$((x-1))
done
```

Подстановка процессов

Эта возможность доступна в SunOS и в версиях ОС UNIX, поддерживающих каталог `/dev/fd` для открытых файлов. Каждый аргумент команды вида `<(список)` или `>(список)` будет запускать процесс *список*, асинхронно подключенный к некоторому файлу в каталоге `/dev/fd`. Имя этого файла станет аргументом команды. Если выбрана форма с `>`, то запись в этот файл будет формировать входные данные для *списка*. Если же использована форма `<`, то файл, переданный в качестве аргумента, будет содержать результат процесса *список*. Например,

```
paste <(cut -f1 file1) <(cut -f3 file2) | tee >(process1) >(process2)
```

вырезает поля 1 и 3 из файлов **file1** и **file2**, соответственно, соединяет результаты и посылает их процессам **process1** и **process2**, а также выдает в стандартный выходной поток. Учтите, что файл, передаваемый как аргумент команды, - это UNIX **pipe(2)**, так что, программы, предполагающие выполнение функции **lseek(2)** над файлом, не будут работать.

Подстановка параметра

Параметр - это идентификатор, одна или несколько цифр или любой из символов `*`, `@`, `#`, `?`, `-`, `$` и `!`. Переменная (параметр, представленный идентификатором) имеет значение и ноль или более атрибутов. Переменным можно присваивать значения и задавать атрибуты с помощью специальной команды **typeset**. Атрибуты, поддерживаемые командным интерпретатором, описаны позже, при рассмотрении специальной команды **typeset**. При экспортировании переменных значения и атрибуты передаются в среду. Командный интерпретатор поддерживает одномерные массивы. К элементам переменной-массива обращаются по индексу. Индекс задается как `[`, после которой идет арифметическое выражение (см. раздел "**Вычисление арифметических выражений**" ниже), а затем `]`. Для присвоения значений массиву используется **set -A имя значение ...**. Значения всех индексов должны быть в диапазоне от 0 до 4095. Массивы не обязательно объявлять. Любая ссылка на переменную с допустимым индексом - законно, и массив будет создан при необходимости. Обращение к массиву без индекса эквивалентно обращению к элементу 0. Если использован идентификатор массива с индексом `*` или `@`, то подставляется значение для каждого из элементов (через символ-разделитель полей).

Значение переменной можно присвоить, написав:

```
имя=значение [ имя=значение ] ...
```

Если для имени установлен атрибут целочисленности, `-i`, значение подлежит арифметическому вычислению, как описано ниже.

Позиционным параметрам, обозначаемым числом, можно присвоить значения с помощью специальной команды **set**. Параметр **\$0** устанавливается равным нулевому аргументу при вызове командного интерпретатора. Если параметр - одна или несколько цифр, то это - позиционный параметр. Позиционный параметр из нескольких цифр необходимо брать в квадратные скобки.

Замена параметра

Для замены параметра его значением используется следующий формат:

```
${выражение}
```

где **выражение** состоит из всех символов до соответствующей `}`. Любой символ `}`, замаскированный обратной косой чертой или входящий в строку в кавычках, а также символы во встроенных арифметических выражениях, подстановках команд и заменах переменных, не учитываются при поиске соответствующей закрывающей скобки, `}`.

Простейшая форма для замены параметра:

```
${параметр}
```

Вместо него будет подставлено значение *параметра*, если он определен.

Имя или символ параметра может быть взято в фигурные скобки, которые не обязательны кроме как для позиционных параметров из нескольких цифр или когда после параметра идет символ, который можно интерпретировать как часть имени. Соответствующая закрывающая фигурная скобка будет определяться путем подсчета уровней вложенности скобок, пропуская входящие в строки в кавычках и подстановки результатов выполнения команд.

Если имя или символ параметра не взяты в фигурные скобки, при замене будет использовано самое длинное допустимое имя, независимо от того, определен ли параметр с этим именем. Когда командный интерпретатор просматривает входные данные для определения границ имени, он не ограничивается уже известными ему определенными именами. Например, если **F** - определенная переменная командного интерпретатора, команда:

```
echo $Fred
```

не выдаст значение переменной **\$F** и **red** после него; интерпретатор выбирает самое длинное допустимое имя, **Fred**, которое в данном случае может быть неопределенным.

Если замена параметра происходит в двойных кавычках:

- Замена имен файлов не будет выполняться в результатах этой замены.
- Выделение полей не будет выполняться над результатами этой замены, за исключением @.

Кроме того, замену параметра можно изменить с помощью одного из следующих форматов. В каждом из случаев, когда необходимо значение *слова* (в зависимости от состояния *параметра*, как описано ниже), *слово* будет подвергнуто замене тильды, замене параметров, подстановке результатов выполнения команд и значений арифметических выражений. Если *слово* не нужно, оно не будет обрабатываться. Символ }, ограничивающий представленные далее изменения замены параметра, ищется так же, как описано в этом разделе ранее. (Например, `${foo-bar}xyz` приведет к подстановке значения **foo**, за которым идет строка **xyz**, если параметр **foo** установлен, или к подстановке строки **barxyz** в противном случае).

`${параметр:-слово}`

Использовать стандартные значения. Если *параметр* не установлен или пуст, будет подставлено *слово* после обработки; в противном случае, подставляется значение параметра.

`${параметр:=слово}`

Присвоить стандартные значения. Если *параметр* не установлен или пуст, значение *слова* после обработки будет присвоено *параметру*. В любом случае, окончательное значение параметра будет подставлено. Только переменные, но не позиционные или специальные параметры, могут получить значения таким образом.

`${параметр:?[слово]}`

Сообщить об ошибке, если значение не задано или пустое. Если *параметр* не установлен или пуст, результат обработки *слова* (или сообщение о том, что параметр не установлен, если *слово* не указано) будет выдан в стандартный поток ошибок и командный интерпретатор завершит работу с ненулевым статусом выхода. Иначе подставляется значение *параметра*. Интерактивному командному интерпретатору завершать работу не нужно.

`${параметр:+[слово]}`

Использовать альтернативное значение. Если параметр не установлен или пуст, будет подставлено пустое значение, иначе, будет подставлено значение слова после обработки.

В представленных выше конструкциях замены параметров использование двоеточия приводит к проверке параметра на неустановленность или пустоту; если же двоеточие не указано, проверяется только, установлен ли параметр. Влияние двоеточия подытожено в следующей таблице:

	параметр установлен и не пуст	параметр установлен, но пуст	параметр не установлен
<code>\${параметр:-слово}</code>	подставить <i>параметр</i>	подставить <i>слово</i>	подставить <i>слово</i>
<code>\${параметр-слово}</code>	подставить <i>параметр</i>	подставить пустое значение	подставить <i>слово</i>
<code>\${параметр:=слово}</code>	подставить <i>параметр</i>	присвоить <i>слово</i>	присвоить <i>слово</i>
<code>\${параметр=слово}</code>	подставить <i>параметр</i>	подставить <i>параметр</i>	присвоить пустое значение
<code>\${параметр:?слово}</code>	подставить <i>параметр</i>	ошибка, выход	ошибка, выход
<code>\${параметр?слово}</code>	подставить <i>параметр</i>	подставить пустое значение	ошибка, выход
<code>\${параметр:+слово}</code>	подставить <i>слово</i>	подставить пустое значение	подставить пустое значение
<code>\${параметр+слово}</code>	подставить <i>слово</i>	подставить <i>слово</i>	подставить пустое значение

Во всех случаях, когда сказано "подставить", выражение заменяется указанным значением. Во всех случаях, когда сказано "присвоить", параметр получает это значение, которое также заменяет выражение.

`$#параметр`

Длина строки. Длина значения *параметра* в символах. Если *параметр* - * или @, то подставляются все позиционные параметры, начиная с **\$1** (разделенные символом-разделителем полей).

Следующие четыре варианта замены параметров позволяют обрабатывать подстроки. В любом случае, при проверке шаблонов будет использоваться нотация сопоставления с образцом, а не регулярных выражений. Если *параметр* - * или @, то подставляются все позиционные параметры, начиная с **\$1** (разделенные

символом-разделителем полей). Взятие всей строки замены параметра в двойные кавычки не вызывает маскировки следующих четырех вариантов символов-шаблонов, тогда как маскировка символов в фигурных скобках учитывается.

`${параметр%слово}`

Удалить самый короткий суффикс, соответствующий шаблону. *Слово* будет обработано для получения шаблона. Замена параметра приведет к подстановке значения *параметра*, из которого удален самый короткий суффикс, соответствующий шаблону.

`${параметр%%слово}`

Удалить самый длинный суффикс, соответствующий шаблону. *Слово* будет обработано для получения шаблона. Замена параметра приведет к подстановке значения *параметра*, из которого удален самый длинный суффикс, соответствующий шаблону.

`${параметр#слово}`

Удалить самый короткий префикс, соответствующий шаблону. *Слово* будет обработано для получения шаблона. Замена параметра приведет к подстановке значения *параметра*, из которого удален самый короткий префикс, соответствующий шаблону.

`${параметр##слово}`

Удалить самый длинный префикс, соответствующий шаблону. *Слово* будет обработано для получения шаблона. Замена параметра приведет к подстановке значения *параметра*, из которого удален самый длинный суффикс, соответствующий шаблону.

Примеры:

`${параметр:-слово}`

В этом примере, команда **ls** выполняется, только если параметр **x** имеет пустое значение или не установлен. (Конструкция **\$(ls)** для подстановки результатов выполнения команды рассмотрена выше в разделе "Подстановка результатов выполнения команд".)

`${x:-$(ls)}`

`${параметр:=слово}`

```
unset X
echo ${X:=abc}
abc
```

`${параметр:?слово}`

```
unset posix
echo ${posix:?}
sh: posix: parameter null or not set
```

`${параметр:+слово}`

```
set a b c
echo ${3:+posix}
posix
```

`${#параметр}`

```
HOME=/usr/posix
echo ${#HOME}
10
```

`${параметр%слово}`

```
x=file.c
echo ${x%.c}.o
file.o
```

`${параметр%%слово}`

```
x=posix/src/std
```

```
echo ${x%%/}*}  
posix
```

`${параметр#слово}`

```
x=$HOME/src/cmd  
echo ${x#$HOME}  
/src/cmd
```

`${параметр##слово}`

```
x=/one/two/three  
echo ${x##*/}  
three
```

Параметры, устанавливаемые командным интерпретатором

Следующие параметры автоматически устанавливаются командным интерпретатором:

#

Количество позиционных параметров в десятичном виде.

-

Флаги, переданные командному интерпретатору при вызове или с помощью команды **set**.

?

Десятичное значение, возвращенное последней выполненной командой.

\$

Номер процесса данного командного интерпретатора.

_

Изначально значением **_** будет полное имя файла командного интерпретатора или выполняемого сценария, переданное в среде. В дальнейшем, ему присваивается последний аргумент предыдущей команды. Этот параметр не устанавливается для асинхронно запускаемых команд. Он также используется для хранения имени соответствующего файла **MAIL** при проверке наличия новой почты.

!

Номер процесса последней вызванной в фоновом режиме команды.

ERRNO

Значение переменной **errno**, установленное последним системным вызовом, который завершился сбоем. Это значение зависит от системы и предназначено для отладки.

LINENO

Номер текущей строки в выполняемом сейчас сценарии или функции.

OLDPWD

Предыдущий рабочий каталог, установленный командой **cd**.

OPTARG

Значение последнего аргумента-опции, обработанного специальной командой **getopts**.

OPTIND

Индекс последнего аргумента-опции, обработанного специальной командой **getopts**.

PPID

Идентификатор родительского процесса для командного интерпретатора.

PWD

Текущий рабочий каталог, установленный командой **cd**.

RANDOM

При каждом обращении к этой переменной генерируется случайное целое число, с равномерным распределением в диапазоне от 0 до 32767. Последовательность случайных чисел можно инициализировать, присваивая числовое значение переменной **RANDOM**.

REPLY

Эта переменная устанавливается оператором **select** и специальной командой **read**, когда ей не переданы аргументы.

SECONDS

При каждом обращении к этой переменной возвращается количество секунд, прошедших с момента вызова командного интерпретатора. Если присвоить этой переменной значение, то при обращении к переменной будет возвращено присвоенное значение плюс количество секунд, прошедших после присвоения.

Переменные, используемые командным интерпретатором

Следующие переменные используются командным интерпретатором:

CDPATH

Путь поиска для команды **cd**.

COLUMNS

Если эта переменная установлена, ее значение используется для определения ширины окна редактирования для режимов редактирования командного интерпретатора и для печати списков выбора (оператора **select**).

EDITOR

Если значение этой переменной заканчивается на **emacs**, **gmacs** или **vi** и переменная **VISUAL** не установлена, то будет включена соответствующая опция (см. описание специальной команды **set** ниже).

ENV

В этой переменной при вызове командный интерпретатор производит замену параметров, а полученное значение используется как полное имя файла, содержащего команды интерпретатора для выполнения в текущей среде. Файл не обязательно должен быть выполняемым. Если полученное после обработки значение **ENV** не задает абсолютное (начинающееся с /) имя файла, результаты непредсказуемы. Значение **ENV** будет игнорироваться, если реальный и эффективный идентификаторы пользователя или реальный и эффективный идентификаторы группы - различны. Эту переменную можно использовать для установки псевдонимов и других локальных значений для данного вызова командного интерпретатора. Файл, на который указывает **ENV**, отличается от **\$HOME/.profile** тем, что **.profile** обычно выполняется в начале сеанса, тогда как **ENV** вызывается в начале каждого вызова командного интерпретатора. Значение **ENV** интерпретируется аналогично сценарию, *вызываемому через точку* (dot script), т.е. команды выполняются в текущей среде и файл должен быть доступен на чтение, но не обязательно на выполнение. Однако, в отличие от сценариев, вызываемых через точку, поиск по перечисленным в **PATH** каталогам не выполняется. Это сделано, чтобы защититься от "троянских программ".

FCEDIT

Имя стандартного редактора для команды **fc**.

FPATH

Путь поиска определений функций. По умолчанию, каталоги **FPATH** просматриваются после каталогов, перечисленных в переменной **PATH**. Если найден выполняемый файл, то он читается и выполняется в текущей среде. Каталоги **FPATH** просматриваются перед **PATH**, когда вызывается функция с атрибутом **-u**. Предопределенный псевдоним **autoload** вызывает создание функции с атрибутом **-u**.

IFS

Внутренние разделители полей, обычно - пробел, табуляция и новая строка, которые используются для разделения слов команды, которые получают в результате подстановки результатов выполнения команды или замены параметра, а также для разделения слов при использовании специальной команды **read**. Первый символ переменной **IFS** используется для разделения аргументов при подстановке значения **\$*** (см. раздел "**Маскировка**" ниже).

HISTFILE

Если эта переменная установлена при вызове командного интерпретатора, то значение интерпретируется как полное имя файла, который будет использоваться для хранения списка выполненных команд. (См. раздел "**Повторный ввод команд**" ниже.)

HISTSIZE

Если эта переменная установлена при вызове командного интерпретатора, то количество ранее введенных команд, которые будут доступны этому командному интерпретатору, будет больше или равно соответствующему числу. Стандартное значение 128.

HOME

Стандартный аргумент (начальный каталог) для команды **cd**.

LC_ALL

Эта переменная задает стандартное значение для переменных **LC_***.

LC_COLLATE

Эта переменная определяет поведение выражений, задающих диапазоны, классов эквивалентности и особенности сортировки многобайтовых символов при сопоставлении с образцом.

LC_CTYPE

Определяет то, как командный интерпретатор обрабатывает символы. Когда переменная **LC_CTYPE** имеет допустимое значение, командный интерпретатор может отображать и обрабатывать текст и имена файлов, содержащие допустимые для соответствующей *локали* символы. Если переменная **LC_CTYPE** (см. **environ(5)**) не установлена в среде, поведение командного интерпретатора определяется значением переменной среды **LANG**. Если установлена переменная **LC_ALL**, ее значение переопределяет как **LANG**, так и значения других переменных **LC_***.

LC_MESSAGES

Эта переменная определяет язык, на котором должны выдаваться сообщения.

LANG

Задаёт стандартное значение для неустановленных или пустых переменных интернационализации. Если какая-то из переменных интернационализации содержит недопустимое значение, утилита будет вести себя так, как если бы ни одна из этих переменных не была определена.

LINENO

Эта переменная устанавливается командным интерпретатором равной десятичному числу, представляющему текущий порядковый номер строки (нумерация начинается с 1) в сценарии или функции перед выполнением каждой команды. Если пользователь делает неопределенной или сбрасывает **LINENO**, эта переменная может потерять свое специальное значение до завершения работы соответствующего командного интерпретатора. Если командный интерпретатор сейчас не выполняет сценарий или функцию, значение **LINENO** не определено.

LINES

Если эта переменная установлена, значение используется для определения длины столбца для выдачи списков выбора (**select**). Списки выбора выдаются вертикально, пока не будет заполнено примерно две трети значения **LINES**.

MAIL

Если эта переменная установлена равной имени файла почтового ящика и переменная **MAILPATH** не установлена, командный интерпретатор информирует пользователя о получении почты в указанный файл.

MAILCHECK

Эта переменная определяет, как часто (в секундах) командный интерпретатор будет проверять, не изменилось ли время модификации любого из файлов, указанных переменными **MAILPATH** или **MAIL**. Стандартное значение - 600 секунд. По истечении указанного времени командный интерпретатор выполнит проверку перед выдачей очередного приглашения.

MAILPATH

Список имен файлов через двоеточие (:). Если эта переменная установлена, командный интерпретатор информирует пользователя о любых изменениях в указанных файлах, которые

произошли за последние **MAILCHECK** секунд. После каждого имени файла можно указать ? и сообщение, которое будет выдано. В сообщении будет выполнена подстановка значений параметров, причем, переменная **\$_** заменяется значением изменившегося файла. Стандартное сообщение - **you have mail in \$_**.

NLSPATH

Определяет местонахождение каталогов сообщений для обработки значений **LC_MESSAGES**.

PATH

Путь поиска для команд (см. раздел "**Выполнение**" ниже). Пользователь не может изменять значение **PATH** при работе в **rksh** (за исключением файла **.profile**).

PPID

Эта переменная устанавливается командным интерпретатором равной десятичному идентификатору процесса, вызвавшего командный интерпретатор. В порожденном командном интерпретаторе (subshell) **PPID** будет установлен в то же значение, что и у родительского командного интерпретатора. Например, команды **echo \$PPID** и (**echo \$PPID**) выдадут одно значение.

PS1

В значении этой переменной выполняется подстановка значений параметров, и оно определяет строку основного приглашения, по умолчанию - "\$ ". Символ ! в основном приглашении заменяется номером команды (см. раздел "**Повторный ввод команды**" ниже). Два подряд вхождения ! при выдаче строки приглашения породят один символ !.

PS2

Строка вторичного приглашения, по умолчанию - "> ".

PS3

Приглашение для выбора, используемое в цикле **select**, по умолчанию - "#? ".

PS4

В значении этой переменной выполняется подстановка значений параметров, и оно выдается перед каждой строкой в трассировке выполнения. Если не указано явно, приглашение трассировки - "+".

SHELL

Полное имя командного интерпретатора поддерживается в среде. При вызове, если базовое имя в значении этой переменной - **rsh**, **rksh** или **krsh**, то командный интерпретатор становится *ограниченным*.

TMOU

Если переменной установлено значение больше нуля, командный интерпретатор будет прекращать работу, если команда не введена в течение соответствующего количества секунд после выдачи приглашения **PS1**. (Учтите, что командный интерпретатор может быть скомпилирован с максимальным значением для этой переменной, которое нельзя превзойти.)

VISUAL

Если значение этой переменной заканчивается строкой **emacs**, **gmacs** или **vi**, будет установлена соответствующая опция (см. описание специальной команды **set** ниже).

Командный интерпретатор задает стандартные значения переменным **PATH**, **PS1**, **PS2**, **PS3**, **PS4**, **MAILCHECK**, **FCEDIT**, **TMOU** и **IFS**, тогда как переменные **HOME**, **SHELL**, **ENV** и **MAIL** им вообще не устанавливаются (хотя переменная **HOME** устанавливается командой [login\(1\)](#)). В некоторых системах переменные **MAIL** и **SHELL** также устанавливаются при регистрации пользователя.

Интерпретация пробелов

После подстановки параметров и результатов выполнения команд, результаты подстановок просматриваются в поисках символов-разделителей полей (указанных в качестве значений **IFS**) и разбиваются на отдельные аргументы по найденным последовательностям таких символов. Явные пустые аргументы (" " или " ") сохраняются. Неявные пустые аргументы (получающиеся в результате подстановки параметров, не имеющих значений) удаляются.

Генерация имен файлов

После подстановок каждое слово команды просматривается в поисках символов *, ? и [, если только не установлена опция -f. При выявлении одного из этих символов слово считается шаблоном. Такое слово заменяется лексикографически упорядоченным списком имен файлов, соответствующих шаблону. Если не найдено ни одного имени файла, соответствующего шаблону, слово остается без изменений. Когда шаблон используется для генерации имен файлов, символ точки (.) в начале имени файла или сразу после /, а также сам символ /, должны сопоставляться явно. Имя файла, начинающееся с точки, не будет соответствовать шаблону с точкой в круглых скобках; то есть, команда

ls @(r*)

выдаст файл с именем .restore, а команда ls @(.r*) - нет. В остальных случаях сопоставления с образцом символы / и . специальным образом не обрабатываются.

*

Соответствует любой строке, включая пустую.

?

Соответствует любому одному символу.

[...]

Соответствует любому из указанных символов. Пара символов, разделенных -, соответствует любому символу в диапазоне между этой парой, включая сами ограничивающие символы. Если первый символ после открывающей "[" - "!", то шаблону соответствуют любые символы, не входящие в набор. Символ - можно включить в любой набор, поместив его первым или последним.

Список_шаблонов - это последовательность из одного или более шаблонов, разделенных символами |.

Составные шаблоны могут быть сформированы из одной или нескольких следующих конструкций:

?(список_шаблонов)

Соответствует любому из указанных шаблонов.

*(список_шаблонов)

Соответствуют нулю или более вхождений указанных шаблонов.

+(список_шаблонов)

Соответствует одному или более вхождений указанных шаблонов.

@(список_шаблонов)

Соответствует ровно одному из указанных шаблонов.

!(список_шаблонов)

Соответствует любому имени, кроме соответствующего одному из указанных шаблонов.

Маскировка

Каждый из приведенных выше метасимволов (см. "Определения") имеет специальное значение для командного интерпретатора и вызывает завершение слова, если только не замаскирован. Символ может быть замаскирован (другими словами, можно лишить его специального смысла), если указать перед ним \. Пара \НОВАЯ_СТРОКА удаляется. Все символы между парой апострофов (') замаскированы. Апостроф нельзя указывать в апострофах. В двойных кавычках (") выполняется подстановка значений параметров и результатов выполнения команд, а символ \ маскирует \, ', " и \$. Значения \$* и @\$ идентичны, когда не взяты в кавычки или когда используются как значения при присвоении параметра или как имя файла. Однако при использовании в качестве аргумента команды, \$* эквивалентно "\$1d\$2d...", где d - первый символ переменной IFS, тогда как @\$ эквивалентно "\$1 \$2 ...". В обратных апострофах (`), символ \ маскирует символы \, ' и \$. Если обратные апострофы выявляются в двойных кавычках, то \ также маскирует символ ".

Зарезервированные слова или псевдонимы можно лишить специального значения, замаскировав любой символ зарезервированного слова. Распознавание имен функций или имен специальных команд, перечисленных ниже, нельзя изменить, маскируя их.

Арифметическое вычисление

Возможность выполнять целочисленную арифметику обеспечивается специальной командой let. Вычисления выполняются с помощью "длинной" арифметики. Константы имеют вид [base#]n, где base - десятичное число от 2 до 36, представляющее основание системы счисления, а n - число в этой системе счисления. Если значение base не указано, предполагается значение 10 (десятичная система счисления). В арифметических выражениях используется тот же синтаксис, приоритеты и ассоциативность операторов, что и в языке C. Поддерживаются все целочисленные операторы, кроме ++, --, ?: и . На переменные в

арифметическом выражении можно сослаться по имени, не используя синтаксис подстановки значений параметров. При указании переменной, ее значение вычисляется как арифметическое выражение. Внутреннее целочисленное представление переменной можно указать с помощью опции **-i** специальной команды **typeset**. Арифметическое вычисление выполняется над значением при каждом присвоении переменной с атрибутом **-i**. Если система счисления не указана, она определяется при первом присвоении значения переменной. Эта система счисления используется, когда происходит подстановка параметра. Поскольку многие из арифметических операторов требуют маскировки, предлагается альтернативная форма команды **let**. Для любой команды, начинающейся с **((**, все символы до соответствующей пары **)** обрабатываются как замаскированное выражение. Точнее, **((...))** эквивалентно **let "..."**.

Приглашения

При интерактивном использовании командный интерпретатор выдает приглашение, получающееся в результате подстановки значения параметра **PS1** перед чтением команды. Если в любой момент вводится новая строка и для завершения команды надо еще вводить символы, выдается вторичное приглашение (т.е., значение **PS2**).

Условные выражения

Условное выражение используется с составной командой **[[** для проверки атрибутов файлов и для сравнения строк. Деление на слова и генерация имен файлов не выполняется для слов между **[[** и **]]**. Каждое выражение может быть построено из одного или нескольких следующих унарных или бинарных выражений:

-a файл

Истинно, если *файл* существует.

-b файл

Истинно, если *файл* существует и является специальным блочным устройством.

-c файл

Истинно, если *файл* существует и является специальным символьным устройством.

-d файл

Истинно, если *файл* существует и является каталогом.

-e файл

Истинно, если *файл* существует.

-f файл

Истинно, если *файл* существует и является обычным файлом.

-g файл

Истинно, если *файл* существует и у него установлен бит **setgid**.

-k файл

Истинно, если *файл* существует и у него установлен бит **sticky**.

-n строка

Истинно, если длина *строки* не равна нулю.

-o опция

Истинно, если указанная *опция* включена.

-p файл

Истинно, если *файл* существует и является специальным файлом **fifo** или каналом (pipe).

-r файл

Истинно, если *файл* существует и доступен на чтение текущему процессу.

-s файл

Истинно, если *файл* существует и его размер больше нуля.

-t дескриптор_файла

Истинно, если файл с указанным числовым *дескриптором_файла* открыт и связан с терминальным устройством.

-u файл

Истинно, если *файл* существует и у него установлен бит **setuid**.

-w файл

Истинно, если *файл* существует и доступен текущему процессу на запись.

-x файл

Истинно, если *файл* существует и доступен текущему процессу на выполнение. Если *файл* существует и является каталогом, проверяется, имеет ли текущий процесс право искать в каталоге.

-z строка

Истинно, если длина *строки* равна нулю.

-L файл

Истинно, если *файл* существует и является символьной связью.

-O файл

Истинно, если *файл* существует и принадлежит пользователю, идентификатор которого совпадает с эффективным идентификатором пользователя этого процесса.

-G файл

Истинно, если *файл* существует и принадлежит группе, соответствующей эффективному идентификатору группы этого процесса.

-S файл

Истинно, если *файл* существует и является сокетом.

файл1 -nt файл2

Истинно, если *файл1* существует и новее, чем *файл2*.

файл1 -ot файл2

Истинно, если *файл1* существует и старше, чем *файл2*.

файл1 -ef файл2

Истинно, если *файл1* и *файл2* существуют и ссылаются на один и тот же файл.

строка

Истинно, если указанная *строка* не является пустой.

строка = шаблон

Истинно, если *строка* соответствует *шаблону*.

строка != шаблон

Истинно, если *строка* не соответствует *шаблону*.

строка1 = строка2

Истинно, если строки *строка1* и *строка2* идентичны.

строка1 != строка2

Истинно, если строки *строка1* и *строка2* не идентичны.

строка1 < строка2

Истинно, если *строка1* идет до *строки2* при сравнении с учетом локали, установленной для переменной **LC_COLLATE**.

строка1 > строка2

Истинно, если *строка1* идет после *строки2* при сравнении с учетом локали, установленной для переменной **LC_COLLATE**.

выражение1 -eq выражение2

Истинно, если *выражение1* равно *выражению2*.

выражение1 -ne *выражение2*

Истинно, если *выражение1* не равно *выражению2*.

выражение1 -lt *выражение2*

Истинно, если *выражение1* меньше чем *выражение2*.

выражение1 -gt *выражение2*

Истинно, если *выражение1* больше чем *выражение2*.

выражение1 -le *выражение2*

Истинно, если *выражение1* меньше или равно *выражению2*.

выражение1 -ge *выражение2*

Истинно, если *выражение1* больше или равно *выражению2*.

В каждом из представленных выше выражений, если *файл* имеет вид */dev/fd/n*, где *n* - целое число, то проверка выполняется для открытого файла с дескриптором *n*.

Составное выражение может быть построено из этих примитивов с помощью одной из следующих конструкций, перечисленных в порядке понижения приоритета.

(*выражение*)

Истинно, если *выражение* истинно. Используется для группировки выражений.

! *выражение*

Истинно, если *выражение* ложно.

выражение1 && *выражение2*

Истинно, если и *выражение1*, и *выражение2* истинны.

выражение1 || *выражение2*

Истинно, если либо *выражение1*, либо *выражение2* истинно.

Ввод-вывод

Перед выполнением команды, ее входной и выходной потоки могут быть перенаправлены с помощью специальной нотации, обрабатываемой командным интерпретатором. Следующие конструкции могут появляться где угодно в простой команде, или идти перед или после любой команды, и они не передаются вызванной команде. Подстановка результатов выполнения команд и значений параметров выполняется перед использованием *слова* или *цифры*, с указанными ниже исключениями. Генерация имен файлов происходит только если шаблону соответствует один файл, а пробельные символы не интерпретируются.

<*слово*

Использовать файл *слово* как стандартный входной поток (файл с дескриптором 0).

>*слово*

Использовать файл *слово* как стандартный выходной поток (файл с дескриптором 1). Если *файл* не существует, он создается. Если файл существует и включена опция **noclobber**, выдается сообщение об ошибке; иначе файл усекается до нулевой длины.

>|*слово*

То же, что и >, но опция **noclobber** игнорируется.

>>*слово*

Использовать файл *слово* как стандартный выходной поток. Если файл существует, выходной поток добавляется в него (в конце файла); иначе файл создается.

<>*слово*

Открыть файл *слово* на чтение и запись как стандартный входной поток.

<< [-]*слово*

Входной поток интерпретатора читается вплоть до строки, совпадающей со *словом* или до конца файла. В *слове* не выполняется подстановка параметров, подстановка результатов выполнения команд или генерация имен файлов. Получившийся в результате документ, который называют "документ-здесь" (here-document), становится стандартным входным потоком команды. Если любой

символ в *слове* замаскирован, то символы в документе никак не интерпретируются; иначе выполняется подстановка значений параметров и результатов выполнения команд, символы *\новая_строка* игнорируются, а символ ** обязательно использовать для маскировки символов **, *\$*, *`* и первого символа слова. Если символ - добавляется к *<<*, то все начальные табуляции удаляются из *слова* и из документа.

<&цифра

Стандартный входной поток дублируется из файла с дескриптором *цифра* (см. **dup(2)**). Аналогично - для стандартного выходного потока, с помощью конструкции *>&цифра*.

<&-

Стандартный входной поток закрывается. Аналогично - для стандартного выходного потока, с помощью конструкции *>&-*.

<&p

Входной поток сопроцесса переносится на стандартный входной поток.

>&p

Выходной поток сопроцесса переносится в стандартный выходной поток.

Если одна из представленных выше конструкций предваряется цифрой, то используется файл с соответствующим дескриптором (вместо стандартных 0 или 1). Например:

... 2>&1

означает, что файл с дескриптором 2 должен открываться на запись как дубликат файла с дескриптором 1. Порядок указания перенаправлений имеет значение. Командный интерпретатор выполняет каждое перенаправление с точки зрения связывания (дескриптор файла, файл) по состоянию на момент обработки. Например, конструкция:

... 1>fname 2>&1

сначала связывает дескриптор файла 1 с файлом **fname**. Затем он связывает дескриптор файла 2 с файлом, связанным с дескриптором файла 1 (т.е. с **fname**). Если поменять перенаправления местами, дескриптор файла 2 будет связан с терминалом (предполагается, что дескриптор файла 1 был связан с ним), а затем дескриптор файла 1 будет связан с файлом **fname**.

Если после команды указан **&**, и управление заданиями не действует, то по умолчанию входным потоком команды является пустой файл **/dev/null**. В противном случае, среда выполнения для команды содержит дескрипторы файлов вызывающего командного интерпретатора, с учетом перенаправлений потоков ввода-вывода.

Среда

Среда (см. **environ(5)**) - это список пар имя-значение, который передается выполняемой программе так же, как обычный список аргументов. Имена должны быть идентификаторами, а значения - текстовые строки. Командный интерпретатор взаимодействует со средой несколькими способами. При вызове командный интерпретатор сканирует среду и создает переменную для каждого найденного имени, присваивая ей соответствующее значение и экспортируя ее. Выполняемые команды наследуют среду. Если пользователь изменяет значения этих переменных или создает новые с помощью команд **export** или **typeset -x**, они становятся частью среды. Среда, которую "видит" любая выполняемая команда, таким образом, состоит из всех пар имя-значение, изначально унаследованных командным интерпретатором, значения которых могли быть изменены в текущем интерпретаторе, а также любых добавлений, сделанных с помощью команд **export** или **typeset -x**.

Среду для любой простой команды или функции можно дополнить, предварив ее одним или несколькими присвоениями значений переменным. Присвоение значения переменной - это слово вида

идентификатор=значение. Таким образом, строки:

TERM=450 cmd args

и

(export TERM; TERM=450; cmd args)

эквиваленты (с точки зрения выполнения **cmd**, за исключением специальных команд, перечисленных ниже, перед которыми указана звездочка).

Если установлен флаг **-k**, все аргументы-присвоения значений переменным помещаются в среду, даже если они указаны после имени команды. Следующие команды сначала выдают **a=b c**, а затем - **c**:

echo a=b c

set -k

echo a=b c

Эта возможность предназначена для использования со сценариями, написанными для ранних версий командного интерпретатора, и ее использование в новых сценариях настоятельно не рекомендуется.

Вероятно, однажды она исчезнет.

Функции

Зарезервированное слово **function**, описанное в разделе "**Команды**" выше, используется для определения функций командного интерпретатора. Функции командного интерпретатора читаются и запоминаются внутренне. Псевдонимы разрешаются при чтении функции. Функции выполняются как команды с аргументами, передаваемыми как позиционные параметры. (См. раздел "**Выполнение**" ниже.)

Функции выполняются в том же процессе, что и вызывающий командный интерпретатор, и разделяют с ним все файлы и текущий рабочий каталог. Установленные вызывающим процессом обработчики сигналов сбрасываются в стандартные действия внутри функции.

Сигнал, не перехваченный и не проигнорированный функцией, вызывает прекращение ее работы и передачу сигнала вызывающему командному интерпретатору. Обработчик **EXIT**, установленный внутри функции, выполняется после завершения функции в вызывающей среде. Обычно, переменные совместно используются вызывающей программой и функцией. Однако, специальная команда **typeset** внутри функции определяет локальные переменные, область действия которых включает текущую функцию и все функции, которые она вызывает.

Специальная команда **return** используется для возврата из вызовов функции. Ошибки в функциях приводят к возврату управления вызывающему.

Имена всех функций можно получить с помощью команды **typeset +f**. Команда **typeset -f** выдает все имена функций вместе с их текстами. Команда **typeset -f имена_функций** выдает текст только указанных функций. Определения функций можно удалять (undefine) с помощью опции **-f** специальной команды **unset**.

Обычно определения функций недоступны при выполнении сценария командным интерпретатором. Опция **-xf** команды **typeset** позволяет экспортировать функцию сценариям, выполняемым без отдельного вызова командного интерпретатора. Функции, которые должны быть определены в нескольких отдельных вызовах командного интерпретатора, надо задать в файле **ENV** с помощью опции **-xf** команды **typeset**.

Команда определения функции

Функция - определенное пользователем имя, которое используется как простая команда для вызова составной команды с новыми позиционными параметрами. Функция определяется с помощью команды определения функции.

Формат команды определения функции следующий:

fname() *составная команда* [перенаправление ввода-вывода ...]

Функция получает имя **fname**; это должно быть допустимое имя. Реализация в качестве расширения может допускать в имени функции и другие символы. Для функций и переменных реализация будет поддерживать различные пространства имен.

Скобки **()** в команде определения функции состоят из двух операторов. Поэтому пробельные символы между **fname**, **(** и **)** допускаются, но необязательны.

Аргумент *составная команда* представляет собой составную команду.

При объявлении функции ни одна из подстановок в тексте составной команды или перенаправления ввода-вывода не выполняется; все подстановки будут выполняться как обычно при каждом вызове функции.

Аналогично, необязательные перенаправления потоков ввода-вывода и присвоения переменным в составной команде будут выполняться в ходе выполнения самой функции, а не обработки ее определения.

При выполнении функции особенности обработки синтаксических ошибок и присвоений переменным будут такими же, как описано для специальных встроенных утилит.

Составная команда будет выполняться при указании имени функции как имени простой команды.

Операнды команды становятся позиционными параметрами на время выполнения *составной команды*; специальный параметр **#** также будет изменен в соответствии с количеством операндов. Специальный параметр **0** меняться не будет. Когда функция завершает работу, значения позиционных параметров и специального параметра **#** будут восстановлены такими, как они были до выполнения функции. Если в *составной команде* выполняется специальная встроенная команда **return**, функция завершается и выполнение продолжается со следующей команды после вызова функции.

Пример того, как можно использовать определение функции в любом месте, где допускается простая команда:

```
# Если переменная i имеет значение "yes",
# определить функцию foo как ls -l
#
[ "$i" = yes ] && foo() {
  ls -l
}
```

Статус выхода определения функции будет 0, если функция была объявлена успешно; иначе он будет больше нуля. Статус выхода вызова функции будет равен статусу выхода последней команды, выполненной функцией.

Задания

Если опция **monitor** команды **set** включена, интерактивный командный интерпретатор ассоциирует задание с каждым конвейером. Он поддерживает таблицу текущих заданий, которую выдает команда **jobs** и присваивает им небольшие целые числа. Когда задание запускается асинхронно с помощью **&**, командный интерпретатор печатает строку, которая имеет вид:

[1] 1234

и показывает, что задание, запущенное асинхронно, получило номер задания 1 и имело один (верхнего уровня) процесс с идентификатором 1234.

Если вы выполняете задание и хотите сделать что-то еще, можно нажать комбинацию клавиш **^Z** (CTRL-Z), которая посылает сигнал **STOP** текущему заданию. Командный интерпретатор затем обычно выдает строку о том, что задание было остановлено ('**Stopped**'), и выдает другое приглашение. Затем вы можете управлять состоянием этого задания, переводя его в фоновый режим командой **bg**, или выполнить ряд других команд, а затем при необходимости снова вернуть команду в приоритетный режим с помощью команды **fg**.

Комбинация клавиш **^Z** срабатывает немедленно и похожа на прерывание в том отношении, что не выданные результаты и непрочитанные входные данные отвергаются, когда она вводится.

Задание, работающее в фоновом режиме, остановится при попытке прочитать с терминала. Фоновым заданиям обычно разрешается выдавать результаты, но это можно отменить, выполнив команду "**stty tostop**". Если установить эту опцию **tty**, то фоновые задания будут останавливаться при попытке выдать результат на терминал так же, как и при попытке чтения с терминала.

Есть несколько способов указать задания в командном интерпретаторе. Задание можно указать по идентификатору любого процесса, составляющего задание, или с помощью одной из следующих конструкций:

%номер

Задание с указанным *номером*.

%строка

Любое задание, командная строка которого начинается с указанной *строки*.

%?строка

Любое задание, командная строка которого содержит указанную *строку*.

%%

Текущее задание.

%+

Эквивалент **%%**.

%-

Предыдущее задание.

Командный интерпретатор немедленно узнает об изменении состояния процесса. Обычно он информирует пользователя, когда задание заблокировано так, что не может дальше продолжать работу, но только непосредственно перед выдачей очередного приглашения. Это сделано для того, чтобы не мешать вашей работе в остальных случаях.

Когда включен режим **monitor**, каждое фоновое задание по завершении работы вызывает срабатывание обработчика сигнала **CHLD**.

При попытке выйти из командного интерпретатора при наличии работающих или остановленных заданий, вы получите предупреждение '**You have stopped(running) jobs**'. Можно использовать команду **jobs** чтобы узнать, какие занятия остались. Если вы сделаете это или немедленно попытаетесь выйти снова, командный интерпретатор не предупредит вас второй раз и остановленные задания будут прекращены. При наличии непрерываемых заданий, запущенных с помощью **nohup**, при попытке выхода вы получите предупреждение: **You have jobs running**.

Для фактического завершения сеанса надо будет повторить выход еще раз; но ваши фоновые задания продолжат работать.

Сигналы

Сигналы **INT** и **QUIT** для вызванной команды игнорируются, если после команды идет **&** и не действует опция **monitor**. В противном случае, срабатывают обработчики сигналов, унаследованные командным интерпретатором от родительского процесса (но см. также описание специальной команды **trap** ниже).

Выполнение

При каждом выполнении команды выполняются описанные выше подстановки. Если имя команды совпадает с одной из перечисленных ниже специальных команд, она выполняется в текущем процессе командного

интерпретатора. Затем имя команды проверяется на совпадение с одной из определенных пользователем функций. При совпадении позиционные параметры сохраняются, а затем устанавливаются равными аргументам вызова функции. При завершении выполнения функции или выполнении команды **return** список позиционных параметров восстанавливается и выполняется любой обработчик сигнала **EXIT**, заданный в функции. Значением функции будет значение последней выполненной команды. Функция также выполняется в текущем процессе командного интерпретатора. Если имя команды не совпадает с именем специальной команды или именем определенной пользователем функции, создается процесс, а затем делается попытка выполнить команду с помощью **exec(2)**.

Переменная командного интерпретатора **PATH** задает последовательность просмотра каталогов в поисках команды. Имена каталогов разделяются двоеточием (:). Стандартная последовательность - **/bin:/usr/bin:** (задающая каталоги **/bin**, **/usr/bin** и текущий каталог именно в таком порядке). Текущий каталог можно указать двумя или более подряд идущими двоеточиями или двоеточием в начале или в конце списка каталогов. Если имя команды содержит /, последовательность просмотра не используется. Иначе в каждом из указанных каталогов ищется выполняемый файл. Если для файла установлено право доступа а выполнение, но он не является каталогом или двоичным выполняемым файлом (**a.out**), предполагается, что этот файл содержит команды интерпретатора. Для его чтения запускается порожденный командный интерпретатор (sub-shell). Все неэкспортированные псевдонимы, функции и переменные в этом случае удаляются. Команда в круглых скобках выполняется в порожденном командном интерпретаторе без удаления неэкспортированных сущностей.

Повторный ввод команд

Текст последних **HISTSIZE** (по умолчанию - 128) команд, введенных с терминального устройства, сохраняется в файле истории. Если не установлена переменная **HISTFILE** или указанный ею файл недоступен на запись, используется файл **\$HOME/.sh_history**. Командный интерпретатор может использовать команды всех интерактивных командных интерпретаторов, использующих тот же **HISTFILE**. Специальная команда **fc** используется для просмотра или редактирования части этого файла. Часть файла для просмотра или редактирования задается номером или начальными символами команды. Можно указать одну команду или диапазон команд. Если в качестве аргумента **fc** не указана программа-редактор, используется значение **FCEDIT**. Если переменная **FCEDIT** не определена, используется **/bin/ed**. Редактируемые команды выдаются при выходе из редактора. Имя редактора - используется для пропуска фазы редактирования и повторного выполнения команды. В этом случае, можно использовать подстановку параметров вида **старое=новое** для изменения команды перед выполнением. Например, если задать псевдоним **r** для **'fc -e -'**, то ввод **'r bad=good c'** выполнит повторно самую последнюю команду, которая начинается символом **c**, с заменой первого вхождения **bad** строкой **good**.

Опции редактирования вводимой строки

Обычно, каждая командная строка просто вводится с терминального устройства и завершается символами новой строки (RETURN или LINEFEED). Если включена одна из опций **emacs**, **gmacs** или **vi**, пользователь может редактировать командную строку. Чтобы оказаться в одном из этих режимов редактирования, установите соответствующую опцию. Опция редактирования автоматически выбирается при каждом присвоении переменной **VISUAL** или **EDITOR** значения, заканчивающегося на одну из этих имен опций. Для использования возможностей редактирования требуется, чтобы пользовательский терминал принимал RETURN как возврат каретки без перевода строки, и чтобы пробел переписывал текущий символ на экране. Режимы редактирования реализуют концепцию, по которой пользователь смотрит через окно на текущую строку. Ширина окна - значение переменной **COLUMNS**, если она определена, и 80 в противном случае. Если ширина окна слишком мала для показа приглашения с учетом как минимум 8 символов для ввода, приглашение усекается слева. Если строка - длиннее чем ширина окна минус два, в конце окна показывается отметка, чтобы уведомить пользователя. По мере перемещения курсора и достижения границ окна, окно будет центрироваться относительно курсора. Отметкой будет **>**, если строка продолжается справа окна, **<** - если слева, и ***** - если строка продолжается с обеих сторон окна. Команды поиска в каждом из режимов редактирования обеспечивают доступ к файлу истории. Сопоставление производится буквально, не с шаблонами, хотя начальный символ **^** в строке ограничивает сопоставление, требуя начинать его с первого символа в строке.

Режим редактирования emacs

Для входа в этот режим надо включить опцию **emacs** или **gmacs**. Единственное отличие между этими двумя режимами состоит в обработке комбинации клавиш **^T**. Для редактирования, переместите курсор в позицию, где надо сделать исправление, а затем вставьте или удалите соответствующие символы или слова. Все команды редактирования - управляющие символы или последовательности. Для указания управляющих

символов используется ^, за которым идет символ. Например, ^F означает **Ctrl+F**. Чтобы ввести этот управляющий символ, нажмите клавишу 'f', удерживая нажатой клавишу **Ctrl**. Клавиша **Shift** не нажимается. (Запись ^? обозначает клавишу **Del (Delete)**.)

Управляющие последовательности записываются как M-, за которыми идет символ. Например, последовательность **M-f** (произносится "Meta f") вводится путем нажатия клавиши ESC (ascii-код 033), а затем - 'f'. (**M-F** означает нажатие ESC, а затем SHIFT (для получения верхнего регистра) и 'F'.) Все команды редактирования работают в любом месте строки (а не только в начале). Ни RETURN, ни LINEFEED после команд редактирования вводить не надо, если об этом не сказано явно.

^F

Переместить курсор вперед (вправо) на один символ.

M-f

Переместить курсор вперед на одно слово. (Редактор **emacs** считает словом строку, состоящую только из букв, цифр и символов подчеркивания.)

^B

Переместить курсор назад (влево) на один символ.

M-b

Переместить курсор назад на одно слово.

^A

Переместить курсор на начало строки.

^E

Переместить курсор на конец строки.

^]char

Переместить курсор вперед на символ **char** в текущей строке.

M-^]char

Переместить курсор назад на символ **char** в текущей строке.

^X^X

Поменять местами курсор и отметку.

erase

(Определенный пользователем с помощью команды **stty(1)** символ **erase**, обычно - ^H или #.)
Удалить предыдущий символ.

^D

Удалить текущий символ.

M-d

Удалить текущее слово.

M-^H

(Meta-backspace) Удалить предыдущее слово.

M-h

Удалить предыдущее слово.

M-^?

(Meta-DEL) Удалить предыдущее слово (если ваш символ прерывания - ^? (DEL, как принято по умолчанию), то эта команда не сработает).

^T

Поменять местами текущий символ со следующим в режиме **emacs**. Поменять местами два предыдущих символа в режиме **gmacs**.

^C

Перевести текущий символ в верхний регистр.

M-c	Перевести текущее слово в верхний регистр.
M-l	Перевести текущее слово в нижний регистр.
^K	Удалить от курсора до конца строки. Если перед этим указан числовой параметр, значение которого меньше, чем текущая позиция курсора, то удалить от указанной позиции до курсора. Если перед этим указан числовой параметр, значение которого больше, чем текущая позиция курсора, удалить от курсора до указанной позиции.
^W	Удалить от курсора до отметки.
M-p	Поместить область от курсора до отметки в стек.
kill	(Определенный пользователем с помощью команды stty(1) символ kill , обычно - ^G или @ .) Полностью удалить текущую строку. Если введены два подряд символа kill , все последующие символы kill вызывают перевод строки (полезно при использовании телетайпов - paper terminals).
^Y	Восстановить последний элемент, удаленный из строки. (Вернуть (yank back) обратно в строку.)
^L	Перевести строку и напечатать текущую строку.
^@	(пустой символ) Установить отметку.
M-пробел	(Meta-пробел) Установить отметку.
J(Новая строка)	Выполнить текущую строку.
M(Return)	Выполнить текущую строку.
eof	Символ конца файла, обычно - ^D , обрабатывается как конец файла только если текущая строка пуста.
^P	Извлечь предыдущую команду. При каждом нажатии ^P происходит доступ к предыдущей по времени команде. Возвращает обратно на одну строку при нажатии на не первой строке многострочной команды.
M-<	Извлечь самую давнюю (старую) строку истории.
M->	Извлечь самую недавнюю (новую) строку истории.
^N	Извлечь следующую командную строку. При каждом нажатии ^N происходит обращение к следующей по времени командной строке.
^Rстрока	Обратный поиск по истории в поисках предыдущей команды, содержащей <i>строку</i> . Если указан параметр 0, выполняется прямой поиск. <i>Строка</i> завершается символами RETURN или NEW LINE.

Если **строка** начинается с ^, соответствующая команда должна начинаться со **строки**. Если **строка** не указана, то происходит доступ к следующей командной строке, содержащей последнюю искомую **строку**. В этом случае параметр 0 обращает направление поиска.

^O

Обработать. Выполнить текущую строку и выбрать следующую из файла истории.

М-цифры

(Escape) Определить цифровые параметры - цифры используются как параметры для следующей команды. Параметры принимают команды ^F, ^B, **erase**, ^C, ^D, ^K, ^R, ^P, ^N, ^J, М-., М-^J, М-_, М-b, М-c, М-d, М-f, М-h, М-l и М-^H.

М-буква

Функциональная клавиша. Ваш список псевдонимов просматривается в поисках псевдонима **буква** и, если соответствующий псевдоним определен, его значение будет вставлено во входную очередь. **Буква** не должна совпадать с одной из перечисленных выше метафункций.

М-[буква]

Функциональная клавиша. Ваш список псевдонимов просматривается в поисках псевдонима **буква** и, если соответствующий псевдоним определен, его значение будет вставлено во входную очередь. Может использоваться для программирования функциональных клавиш на многих терминалах.

М-.

Последнее слово предыдущей команды вставляется в строку. Если перед командой указан числовой параметр, значение этого параметра определяет, какое слово вставить вместо последнего.

М- _

То же, что и М-..

М-*

Звездочка добавляется в конце слова и выполняется попытка подстановки имен файлов.

М-ESC

Завершение имени файла. Заменить текущее слово самым длинным общим префиксом всех имен файлов, соответствующих текущему слову с добавлением звездочки. Если находится единственный соответствующий файл, добавляется /, если файл является каталогом, и пробел, если он каталогом не является.

М-=

Выдать файлы, соответствующие текущему слову как шаблону при условии добавления в конце звездочки.

^U

Умножить параметр следующей команды на 4.

Замаскировать следующий символ. Символы редактирования, заданные пользователем символы **erase**, **kill** и **interrupt** (обычно - ^?) можно вводить в командной строке или в строке поиска, если предварить их символом \. \ отменяет свойства редактирования следующего символа, если они ему присущи.

^V

Выдать версию командного интерпретатора.

М-#

Вставить # в начале строки и выполнить ее. Это приводит к вставке комментария в файл истории.

Режим редактирования vi

Поддерживается два режима набора текста. Первоначально, при вводе команды вы находитесь в *режиме ввода*. Для редактирования войдите в управляющий режим, нажав ESC (033), и переместите курсор в позицию, где требуется исправление, а затем вставьте или удалите соответствующие символы или слова.

Большинство управляющих команд можно предварять необязательным *числом*, задающим количество повторений.

При работе в режиме **vi** в большинстве систем, каноническая обработка изначально включена, и команда будет выдавать снова, если скорость обмена данными - 1200 бод или выше, и она содержит любые управляющие символы, или с момента выдачи приглашения прошло менее секунды. Ввод символа ESC прекращает каноническую обработку для остальной части команды, и пользователь затем может изменять командную строку. Эта схема имеет преимущества над канонической обработкой за счет опережающего ввода с клавиатуры в непосредственном режиме.

Если также установлена опция **viraw**, для терминала каноническая обработка будет всегда отключена. Этот режим принят неявно для систем, не поддерживающих два альтернативных ограничителя конца строки, и может оказаться полезным для некоторых терминалов.

Команды редактирования ввода

По умолчанию редактор находится в режиме ввода.

er**ase**

(Заданный пользователем с помощью команды **stty(1)** символ **er****ase**, обычно - ^H или #.) Удалить предыдущий символ.

^W

Удалить предыдущее слово, отделенное пробелами.

^D

Завершить работу командного интерпретатора.

^V

Замаскировать следующий символ. Символы редактирования и заданные пользователем символы **er****ase** или **kill** можно вводить в командной строке или строке поиска, если предварить комбинацией ^V. Комбинация ^V удаляет управляющие свойства следующего символа (если они у него есть).

\

Замаскировать следующий символ **er****ase** или **kill**.

Команды перемещения курсора

Эти команды будут перемещать курсор.

[*число*]**l**

Переместить курсор вперед (вправо) на один символ.

[*число*]**w**

Переместить курсор вперед на одно (алфавитно-цифровое) слово.

[*число*]**W**

Переместить курсор на начало следующего слова после пробела.

[*число*]**e**

Переместить курсор на конец слова.

[*число*]**E**

Переместить курсор на конец текущего слова, отделенного пробелом.

[*число*]**h**

Переместить курсор назад (влево) на один символ.

[*число*]**b**

Переместить курсор назад на одно слово.

[*число*]**B**

Переместить курсор на предыдущее слово, отделенное пробелом.

[*число*]**|**

Переместить курсор на столбец, заданный *числом*.

[*число*]f*c*

Найти следующий символ **c** в текущей строке.

[*число*]F*c*

Найти предыдущий символ **c** в текущей строке.

[*число*]t*c*

Эквивалентно последовательности команд **f** и **h**.

[*число*]T*c*

Эквивалентно последовательности команд **F** и **I**.

[*число*];

Повторить указанное *число* раз последнюю команду поиска символа, **f**, **F**, **t** или **T**.

[*число*],

Повторить указанное *число* раз последнюю команду поиска символа в противоположном направлении.

0

Переместить курсор в начало строки.

^

Переместить курсор на первый не пробельный символ в строке.

\$

Переместить курсор на конец строки.

%

Перейти к соответствующей скобке, (), { }, [или]. Если курсор не находится на одном из перечисленных выше символов, сначала в оставшейся части строки ищется первое вхождение одного из них.

Команды поиска

Эти команды обеспечивают доступ к истории выполненных интерпретатором команд.

[*число*]k

Выбрать предыдущую команду. При каждом вводе **k** возвращается предыдущая по времени команда.

[*число*]-

Эквивалентна **k**.

[*число*]j

Выбрать следующую команду. При каждом вводе **j** возвращается следующая по времени команда.

[*число*]+

Эквивалентна **j**.

[*число*]G

Выбирается команда с номером, задаваемым *числом*. По умолчанию - последняя выполненная.

/строка

Искать предыдущую по времени команду, содержащую *строку*. Строка завершается нажатием RETURN или символом NEWLINE. Если *строка* начинается символом ^, сопоставление ведется с начала команды. Если указана пустая *строка*, производится поиск строки, которую искали перед этим.

?строка

То же, что и /, но выполняется поиск следующей по времени команды.

n

Искать следующее соответствие последнему шаблону, заданном в команде / или ?.

N

Искать следующее соответствие последнему шаблону, заданном в команде / или ?, но в противоположном направлении. Просматривать историю в поисках *строки*, указанной в предыдущей команде /.

Команды изменения текста

Эти команды будут изменять строку.

a

Перейти в режим ввода и ввести (добавить) текст после текущего символа.

A

Добавить текст в конце строки. Эквивалентна **\$a**.

[число]с**перемещение**

c[число]**перемещение**

Удалить от текущего символа до символа, на который был бы установлен курсор при указанном *перемещении*, и перейти в режим ввода. Если в качестве перемещения указано **c**, удалить всю строку и перейти в режим ввода.

C

Удалить все от текущего символа до конца строки и перейти в режим ввода. Эквивалентна **c\$**.

[число]s

Удалить указанное *число* символов и перейти в режим ввода.

S

Эквивалентна **ss**.

D

Удалить от текущего символа до конца строка. Эквивалентна **d\$**.

[число]d**перемещение**

d[число]**перемещение**

Удалить от текущего символа до символа, на который был бы установлен курсор при указанном *перемещении*. Если в качестве перемещения указано **d**, удаляется вся строка.

i

Перейти в режим ввода и вставить текст перед текущим символом.

I

Вставить текст перед началом строки. Эквивалентна **0i**.

[число]P

Поместить предыдущее изменение текста перед курсором.

[число]p

Поместить предыдущее изменение текста после курсора.

R

Перейти в режим ввода и заменить символы на экране набираемыми символами, переписывая вверх.

[число]rc

Заменить указанное *число* символов, начиная с текущей позиции, символом **c**, и перевести курсор на следующий символ.

[число]x

Удалить текущий символ.

[число]X

Удалить предыдущий символ.

[число].

Повторить предыдущую команду изменения текста.

[число]~

Изменить на противоположный регистр указанного **числа** символов, начиная с текущей позиции курсора, и перевести курсор на следующий символ.

[число]_

Вызывает добавление указанного **числа** слов предыдущей команды и переход в режим ввода. Если **число** не указано, используется последнее слово.

Вызывает добавление ***** к текущему слову и выполняет попытку генерации имен файлов. Если соответствующие файлы не найдены, выдается звуковой сигнал. В противном случае, слово заменяется соответствующими файлами и выполняется переход в режим ввода.

Завершить имя файла. Заменяет текущее слово самым длинным общим префиксом всех имен файлов, соответствующих текущему слову с добавлением звездочки. Если соответствие ровно одно, добавляется **/**, если файл является каталогом, и пробел, если файл не является каталогом.

Другие команды редактирования

Команды различного назначения.

[число]у*перемещение*

у*[число]перемещение*

Копировать символы от текущего до символа, на который был бы помещен курсор при указанном *перемещении*, в буфер удаления. Текст и позиция курсора не меняются.

Y

Копировать в буфер от текущей позиции до конца строки. Эквивалентна **у\$**.

и

Отменить последнюю команду изменения текста.

U

Отменить все команды изменения текста, выполненные над строкой.

[число]v

Возвращает результат редактирования (с помощью **fc -e \${VISUAL:-\${EDITOR:-vi}}**) соответствующей **числу** команды во входном буфере. Если **число** не указано, используется текущая строка.

^L

Перевести строку и выдать текущую строку. Срабатывает только в управляющем режиме.

J(Новая строка)

Выполнить текущую строку, независимо от режима.

M(Return)

Выполнить текущую строку, независимо от режима.

#

Если первый символ команды - **#**, то эта команда удаляет этот символ **#** и каждый **#** после новой строки. В противном случае, вставляет символ **#** перед каждой строкой команды. Позволяет вставить текущую строку в историю выполненных команд как комментарий, и удалять комментарии с ранее закомментированных команд, выбранных из файла истории.

=

Выдать имена файлов, соответствующих текущему слову с добавлением звездочки.

@буква

Просматривается список псевдонимов в поисках псевдонима *_буква* и, если такой псевдоним определен, его значение будет вставлено во входную очередь для обработки.

Специальные команды

Следующие простые команды выполняются процессом командного интерпретатора. Перенаправление ввода-вывода разрешено. Если не указано иначе, результаты записываются в файл с дескриптором 1, а статус выхода, если нет синтаксической ошибки, - 0. Команды, перед которыми ниже указана одна или две звездочки (*), обрабатываются следующим специальным образом:

1. Списки присвоений переменным, предшествующие команде, остаются в силе после завершения команды.
2. Перенаправления ввода-вывода выполняются после присвоения значений переменным.
3. Ошибки вызывают прекращение работы сценария, в котором они содержатся.
4. Слова, следующие за командой, помеченной **, соответствующие формату присвоения значений переменным, обрабатываются по тем правилам, что и присвоения. Это означает, что подстановка тильды выполняется после знака =, а разбиение на слова и генерация имен файлов не выполняется.

* : [аргумент...]

Эта команда только подставляет параметры.

* . файл [аргумент...]

Прочитать весь файл и выполнить указанные в нем команды. Команды выполняются в среде текущего командного интерпретатора. Путь поиска, задаваемый переменной **PATH** используется для поиска каталога, содержащего *файл*. Если указаны *аргументы*, они становятся позиционными параметрами. В противном случае, позиционные параметры не меняются. Статусом выхода является статус выхода последней выполненной команды.

** alias [-tx] [имя[=значение]]...

Команда **alias** без аргументов выдает список псевдонимов в виде *имя=значение* в стандартный выходной поток. Псевдоним определен для каждого *имени*, значение которого выдано. Хвостовой пробел в *значении* вызывает проверку следующего слова на подстановку псевдонима. Флаг **-t** используется для установки и выдачи отслеживаемых (tracked) псевдонимов. Значение отслеживаемого псевдонима - полное имя файла, соответствующего данному *имени*. Значение становится неопределенным при изменении значения переменной **PATH**, но псевдоним остается отслеживаемым. Без флага **-t** для каждого *имени* в списке аргументов, для которого не указано *значение*, выдается имя и значение псевдонима. Флаг **-x** используется для установки или выдачи экспортированных псевдонимов. Экспортированный псевдоним определен для сценариев, вызываемых по имени. Статус выхода - ненулевой, если указано *имя*, но не указано *значение*, и для этого имени не был определен псевдоним.

bg [%задание...]

Эта команда работает только в системах, поддерживающих управление заданиями. Переводит каждое указанное задание в фоновый режим. Если задание не указано, в фоновый режим переводится текущее задание. Описание формата параметра *задание* см. в разделе "Задания" выше.

* break [n]

Выйти из охватывающего цикла **for**, **while**, **until** или **select**, если он есть. Если указано значение *n*, выйти из *n* уровней вложенных циклов.

* continue [n]

Перейти на следующую итерацию охватывающего цикла **for**, **while**, **until** или **select**. Если указано значение *n*, перейти на следующую итерацию *n*-ного охватывающего цикла.

cd [*аргумент*]
cd *старый* *новый*

Эта команда может быть в одной из двух форм. В первой форме она изменяет текущий каталог на указанный в качестве *аргумента*. Если указан аргумент -, каталог меняется на предыдущий. Значение переменной командного интерпретатора **HOME** является стандартным значением *аргумента*. Переменная **PWD** устанавливается равной текущему каталогу. Переменная командного интерпретатора **CDPATH** задает путь поиска для каталога, заданного в качестве *аргумента*. Альтернативные имена каталогов разделяются двоеточием (:). Стандартный путь - пустой (что задает текущий каталог). Учтите, что текущий каталог задается пустым именем, которое может указываться сразу после символа равенства или между двоеточиями в любом месте списка каталогов. Если *аргумент* начинается с /, то путь поиска не используется. В противном случае, каждый каталог в пути просматривается в поисках *аргумента*. Вторая форма команды **cd** подставляет строку *новая* вместо строки *старая* в имени текущего каталога, **PWD**, и пытается перейти в этот новый каталог. Команда **cd** не может быть выполнена командным интерпретатором **rksh**.

command [-p] [*имя_команды*] [*аргумент...*]
command [-v -V] *имя_команды*

Утилита **command** заставляет командный интерпретатор обрабатывать аргументы как простую команду, подавляя поиск функции командным интерпретатором. Флаг **-p** приводит к поиску команды с использованием стандартного значения **PATH**, по которому гарантировано будут найдены стандартные утилиты. Флаг **-v** записывает в стандартный выходной поток строку - полное имя или команду, которая будет использоваться командным интерпретатором, в текущей среде, для вызова *имени_команды*. Флаг **-V** выдает в стандартный выходной поток строку, показывающую, как указанное *имя_команды* будет интерпретироваться командным интерпретатором в текущей среде.

echo [*аргумент...*]

Описание и использование см. на странице справочного руководства **echo(1)**.

* **eval** [*аргумент...*]

Аргументы читаются как входные данные для командного интерпретатора, а получившаяся команда или команды выполняются.

* **exec** [*аргумент...*]

Если указаны *аргументы*, соответствующая команда выполняется вместо данного командного интерпретатора, без создания нового процесса. Среди аргументов могут быть перенаправления ввода-вывода, которые влияют на текущий процесс. Если аргументы не заданы, эта команда изменяет дескрипторы файлов как предписывается списком перенаправления ввода-вывода. В этом случае, любые дескрипторы файлов больше 2, открытые с помощью этого механизма, закрываются при вызове другой программы.

* **exit** [*n*]

Требует от вызывающего командного интерпретатора или сценария завершить работу со статусом выхода *n*. Значением будут младшие 8 битов указанного статуса. Если значение *n* не указано, статусом выхода будет статус выхода последней выполненной команды. При выходе, если срабатывает обработчик сигнала, последней выполненной считается команда, выполненная перед срабатыванием обработчика. Символ EOF (конец файла) также приводит к завершению работы командного интерпретатора, если только для него не установлена опция **ignoreeof**(см. команду **set** ниже).

** **export** [*имя[=значение]*]...

Указанные имена помечаются для автоматического экспортирования в среду последующих команд.

fc [-e *редактор*] [-nlr] [*первая* [*последняя*]]
fc -e - [*старое=новое*] [*команда*]

В первой форме, из списка последних введенных с терминала **HISTSIZE** команд выбирается диапазон команд с *первой* по *последнюю*. Аргументы *первая* и *последняя* могут быть указаны как число или как строка. Строка используется для нахождения самой недавней команды, которая с нее начинается. Отрицательное число используется как смещение от номера текущей команды. Если выбран флаг **-l**, соответствующие команды выдаются в стандартный выходной поток. В противном случае, вызывается указанный *редактор* для файла, содержащего соответствующие команды. Если

редактор не указан, то используется редактор, заданный значением переменной **FCEDIT** (по умолчанию - **/bin/ed**). По завершении редактирования, отредактированные команды выполняются. Если параметр **последняя** не указан, он будет установлен равным параметру **первая**. Если не указан параметр **первая**, по умолчанию используется предыдущая команда для редактирования и -16 для выдачи. Флаг **-r** изменяет порядок следования команд, а флаг **-n** подавляет показ номеров команд при выдаче. Во второй форме **команда** повторно выполняется после выполнения подстановки **старое=новое**. Если аргумент **команда** не задан, выполняется самая последняя команда, введенная с этого терминала.

fg [%задание...]

Эта команда работает только в системах, поддерживающих управление заданиями. Каждое указанное **задание** переводится в приоритетный режим. В противном случае, в приоритетный режим переводится текущее задание. Описание формата параметра **задание** см. в разделе "**Задания**" выше.

getopts строка_опций имя [аргумент...]

Проверяет **аргументы** на соответствие синтаксису опций. Если ни один **аргумент** не указан, используются позиционные параметры. Аргумент-опция начинается с + или -. Аргумент, не начинающийся с + или -, а также аргумент - заканчивают список опций. **Строка_опций** содержит список букв, которые распознает команда **getopts**. Если после буквы идет :, предполагается, что у этой опции есть аргумент. Опции можно отделять от аргумента пробелами.

Команда **getopts** помещает следующую найденную букву опции в переменную **имя**, при каждом вызове, добавляя +, если аргумент начинается с +. Индекс следующего аргумента запоминается в переменной **OPTIND**. Аргумент опции, если есть, запоминается в переменной **OPTARG**.

Начальный символ : в **строке_опций** требует от **getopts** сохранять букву недопустимой опции в **OPTARG**, а также устанавливать переменной **имя** значение ?, если опция неизвестна, и значение :, если требуемая опция отсутствует. В противном случае, команда **getopts** выдает сообщение об ошибке. Статус выхода - ненулевой, когда опций больше нет. Описание и принципы использования см. на странице справочного руководства **getoptcv(1)**.

hash [имя...]

Для каждого **имени** командный интерпретатор определяет и запоминает местонахождение в пути поиска соответствующей команды. Опция **-r** заставляет командный интерпретатор забыть все запомненные местонахождения. Если аргументы не указаны, выдается информация о запомненных командах. В поле **hits** выдается количество вызовов соответствующей команды процессом командного интерпретатора. В поле **cost** выдается оценка объема работы для нахождения команды в пути поиска. Если команда найдена в "относительном" каталоге в пути поиска, после перехода в этот каталог, запомненное местонахождение этой команды перевычисляется. Команды, для которых это было сделано, помечаются звездочкой (*) по соседству с информацией о количестве вызовов. При перевычислении стоимость (cost) увеличивается.

jobs [-lnp] [%задание...]

Выдает информацию о каждом из указанных **заданий** или обо всех активных заданиях, если задания явно не указаны. Флаг **-l** приводит к выдаче идентификаторов процессов, помимо обычной информации. Флаг **-n** выдает только задания, остановленные или завершившиеся с момента последнего уведомления. Флаг **-p** вызывает выдачу только группы процесса. Описание формата параметра **задание** см. в разделе "**Задания**" выше и на странице справочного руководства **jobs(1)**.

kill [-сигнал] %задание...

kill [-sig] идентификатор_процесса...

kill -l

Посылает либо сигнал **TERM** (завершить), либо указанный **сигнал** перечисленным заданиям или процессам. Сигналы можно задавать по номеру или по имени (как описано на странице справочного руководства **signal(3HEAD)**, но без префикса "**SIG**", за исключением сигнала **SIGCHD**, для которого надо использовать имя **CHLD**). Если посылается сигнал **TERM** (завершить) или **HUP** (отключение), то остановленному заданию или процессу будет послан сигнал **CONT** (продолжить). В качестве аргумента **задание** можно указывать идентификатор процесса, не входящего ни в одно из активных заданий. Описание формата параметра **задание** см. в разделе "**Задания**". Во второй форме, **kill -l**, выдаются имена и номера сигналов.

let аргумент...

Каждый **аргумент** обрабатывается как отдельное арифметическое выражение для вычисления. Описание вычисления арифметического выражения см. в разделе "**Вычисление арифметических выражений**".

Статусом выхода будет 0, если значение последнего выражения - ненулевое, и 1 в противном случае.

login *аргумент...*

Эквивалентна команде **exec login *аргумент...***. Описание см. на странице справочного руководства [login\(1\)](#).

*** newgrp [*аргумент...*]**

Эквивалентна команде **exec /bin/newgrp *аргумент...***.

print [-Rnprsu]*[n]* [*аргумент...*]

Механизм вывода командного интерпретатора.

При вызове без флагов или с флагом **-**, **аргументы** выдаются в стандартный выходной поток, как описано на странице справочного руководства **echo(1)**. Статус выхода будет 0, если только получится открыть выходной файл на запись.

-n

подавляет выдачу символа новой строки при выводе.

-R | -r

Непосредственный (raw) режим. Игнорируются соглашения по управляющим последовательностям команды **echo**. Опция **-R** будет выдавать все последующие аргументы и опции, кроме **-n**.

-p

Записывает аргументы в программный канал процесса, запущенного с помощью **|&**, а не в стандартный выходной поток.

-s

Выдает аргументы в файл истории, а не в стандартный выходной поток.

-u [*n*]

Указывает одну цифру - номер дескриптора файла ***n***, в который будет выполнен вывод. По умолчанию используется 1.

pwd

Эквивалентна **print -r - \$PWD**.

read [-prsu]*[n]* [*имя?приглашение*] [*имя...*]

Механизм ввода командного интерпретатора. Одна строка читается и разбивается на поля по разделителям, указанным в переменной **IFS**. Символ маскировки, (****), используется для отмены специального значения следующего символа и для продолжения ввода на следующей строке. В непосредственном (raw) режиме, **-r**, символ **** не имеет специального значения. Первое поле присваивается первому **имени**, второе - второму и т.д., а оставшиеся поля присваиваются последнему имени. Опция **-p** вызывает получение входной строки из входного канала процесса, порожденного командным интерпретатором с помощью **|&**. Если указан флаг **-s**, ввод будет сохранен как команда в файле истории. Флаг **-u** можно использовать для указания одной цифры дескриптора файла, ***n***, из которого надо читать. Дескриптор файла можно открыть с помощью специальной команды [exec](#). Значение ***n*** по умолчанию - 0. Если **имя** не указано, в качестве стандартного используется **REPLY**. Статус выхода будет 0, за исключением ситуаций, когда не удалось открыть входной файл на чтение или обнаружен символ EOF. EOF при использовании опции **-p** вызывает очистку для соответствующего процесса, так что можно будет породить новый процесс. Если первый аргумент содержит **?**, оставшаяся часть этого слова используется как приглашение, выдаваемое в стандартный поток ошибок при работе командного интерпретатора в интерактивном режиме. Статус выхода - 0, если только не обнаружит символ EOF.

**** readonly [*имя* [=значение]]...**

Указанные имена помечаются как доступные только для чтения и они не могут быть изменены последующими присвоениями.

*** return [*n*]**

Вызывает выход из функции или сценария, запущенного через **'.'**, в вызывающий сценарий со статусом возврата ***n***. Значением будут младшие 8 битов указанного статуса. Если ***n*** не указано, возвращается статус выхода последней выполненной команды. Если команда **return** вызвана не в функции и не в сценарии, запущенном через **'.'**, она соответствует команде [exit](#).

set [±abCefhkmnopstuvx**] [**±o** опция]... [**±A** *имя*] [*аргумент...*]**

Флаги этой команды имеют следующие значения:

-A

Присвоение массиву. Сбросить переменную **имя** и присвоить значения последовательно из списка **аргументов**. Если использован флаг **+A**, переменная **имя** первоначально не сбрасывается.

-a

Все определенные в дальнейшем переменные автоматически экспортируются.

-b

Требует от командного интерпретатора уведомлять пользователя асинхронно о завершении фоновых заданий. В стандартный поток ошибок будет выдаваться следующее сообщение:

```
"[%d]%%c  %s%s\n", <номер_задания>, <текущее>, <статус>, <имя_задания>
```

где поля имеют следующие значения:

<текущее>

Символ **+** идентифицирует задание, которое будет использоваться по умолчанию для команд **fg** и **bg**; это задание также можно указать с помощью идентификатора задания **%+** или **%%**. Символ **-** идентифицирует задание, которое стало бы используемым по умолчанию при завершении текущего; это задание также можно указать с помощью идентификатора задания **%-**. Для других заданий в этом поле выдается пробел. Не более одного задания может быть помечено символом **+** и не более одного задания - знаком **-**. Если есть приостановленное задание, приостановленное задание будет текущим. Если есть не менее двух приостановленных заданий, то предыдущее задание тоже будет приостановленным.

<номер_задания>

Номер, который может использоваться для идентификации группы процессов для команд **wait**, **fg**, **bg** и **kill**. При использовании этих команд задание можно указать, предварив номер задания символом **%**.

<статус>

Не определяется.

<имя_задания>

Не определяется.

Когда командный интерпретатор уведомляет пользователя о том, что задание завершено, он может удалить идентификатор процесса задания из своего списка известных в текущей среде выполнения. По умолчанию асинхронное уведомление не будет включено.

-C

Предотвращает перезапись существующих файлов с помощью оператора перенаправления командного интерпретатора, **>**; оператор перенаправления **>|** будет переопределять эту опцию **noclobber** для отдельного файла.

-e

Если команда имеет ненулевой статус выхода, выполнить обработчик сигнала **ERR**, если он установлен, и завершить работу. Этот режим отключен при чтении профилей.

-f

Отключает генерацию имен файлов.

-h

Каждая команда при первом вводе становится отслеживаемым псевдонимом.

-k

Все аргументы-присвоения переменным помещаются в среду для команды, а не только те, что предшествуют имени команды.

-m

Фоновые задания будут работать в отдельной группе процессов и при завершении будет выдаваться строка. В сообщении о завершении будет выдаваться статус выхода фоновых заданий. В системах, поддерживающих управление заданиями, этот флаг автоматически устанавливается для интерактивных командных интерпретаторов.

-n

Читать команды и проверять их на синтаксические ошибки, но не выполнять. Для интерактивных командных интерпретаторов игнорируется.

-o

Следующий аргумент может быть одной из опций:

allexport

То же, что и опция [-a](#).

errexist

То же, что и опция [-e](#).

bgnice

Все фоновые задания запускаются с пониженным приоритетом. Это - стандартный режим работы.

emacs

Позволяет при вводе командной строки использовать команды в стиле редактора **emacs**.

gmacs

Позволяет при вводе командной строки использовать команды в стиле редактора **gmacs**.

ignoreeof

Командный интерпретатор не завершит работу при получении символа конца файла (EOF). Надо использовать команду **exit**.

keyword

То же, что и опция [-k](#).

markdirs

Ко всем именам каталогов, получающимся в результате генерации имен файлов, добавляется завершающий символ **/**.

monitor

То же, что и опция [-m](#).

noclobber

Предотвращает перезапись существующих файлов при перенаправлении с помощью **>**. Если эта опция включена, для перезаписи файла требуется использовать **>|**. Эквивалент опции [-u](#).

noexec

То же, что и опция [-n](#).

noglob

То же, что и опция [-f](#).

nolog

Не сохранять определения функций в файле истории.

notify

Эквивалент опции [-b](#).

nounset

То же, что и опция [-u](#).

privileged

То же, что и опция [-p](#).

verbose

То же, что и опция [-v](#).

trackall

То же, что и опция [-h](#).

vi

Переводит в режим ввода в стиле редактора **vi**, пока вы не введете управляющий символ 033 (клавиша **Esc**). Он переводит в управляющий режим. Нажатие клавиши **Enter** посылает строку на выполнение.

viraw

Каждый символ обрабатывается по мере ввода в режиме **vi**.

xtrace

То же, что и опция **-x**.

Если имя опции не указано, выдаются текущие установки опций.

-p

Отключает обработку файла **\$HOME/.profile** и использует файл **/etc/suid_profile** вместо файла **ENV**. Этот режим включен, когда эффективный идентификатор пользователя не совпадает с реальным или когда эффективный идентификатор группы не совпадает с реальным. Отключение этой опции вызывает установку эффективных идентификаторов пользователя и группы равным реальным.

-s

Сортировать позиционные параметры лексикографически.

-t

Выйти после чтения и выполнения одной команды.

-u

Считать неустановленные параметры ошибкой при подстановке.

-v

Выдавать входные строки командного интерпретатора по мере чтения.

-x

Выдавать команды и их аргументы по мере выполнения.

-

Отключает флаги **-x** и **-v** и останавливает проверку аргументов на совпадение с флагами.

--

Не изменять никакие флаги; полезно для установки параметру **\$1** значения, начинающегося с **-**. Если после этого флага аргументов нет, позиционные параметры сбрасываются.

Использование **+** вместо **-** вызывает выключение этих флагов. Перечисленные флаги можно также указывать при вызове порожденного командного интерпретатора. Текущий набор флагов можно найти в **\$-**. Если не установлен флаг **-A**, остальные аргументы считаются позиционными параметрами и присваиваются, последовательно, **\$1**, **\$2**...

...

Если аргументы для команды не заданы, имена и значения всех переменных выдаются в стандартный выходной поток.

*** shift [n]**

Позиционные параметры, начиная с **\$n+1**..., переименовываются в **\$1**...; по умолчанию значение **n** - 1. Параметр **n** может быть любым арифметическим выражением, которое вычисляется в неотрицательное число меньше или равное **\$#**.

stop %идентификатор_задания...

stop идентификатор_процесса...

Команда **stop** останавливает выполнение фоновых заданий с указанными идентификаторами или любого процесса с указанным идентификатором. (see ps(1)).

suspend

Приостанавливает выполнение текущего командного интерпретатора (если только он не является начальным).

test выражение

Вычисляет условные выражения. См. описание использования в разделе "**Условные выражения**" ранее и на странице справочного руководства **test(1)**.

*** times**

Выдает накопленные времена выполнения в пользовательском и системно режиме для командного интерпретатора и всех запущенных из него процессов.

* **trap** [*аргумент сигнал...*]

Аргумент - это команда (*обработчик*), которая читается и выполняется при получении командным интерпретатором указанных *сигналов*. *Аргумент* просматривается при выполнении команды **trap** и при получении сигнала. *Сигнал* можно задать по номеру или по имени. Команды **trap** выполняются в порядке номеров сигналов. Любая попытка задать обработчик для номера сигнала, который игнорировался при входе в текущий командный интерпретатор, не сработает.

Если указан *аргумент* -, командный интерпретатор сбросит обработчик каждого *сигнала* в стандартное значение. Если *аргумент* пустой (""), командный интерпретатор будет игнорировать все указанные *сигналы* при их получении. В противном случае, *аргумент* будет читаться и выполняться командным интерпретатором при получении одного из соответствующих сигналов. Указанное в команде **trap** действие (*обработчик*) будет переопределять предыдущее действие (будь-то стандартное или явно установленное). После завершения действия значение **\$?** будет таким же, как и перед вызовом обработчика.

В качестве *сигнала* можно указывать **EXIT**, 0 (эквивалентно **EXIT**) или имя сигнала без префикса **SIG**, например, **HUP**, **INT**, **QUIT**, **TERM**. Если *сигнал* имеет значение 0 или **EXIT** и команда **trap** выполняется в теле функции, то команда *аргумент* выполняется после завершения функции. Если указан *сигнал* 0 или **EXIT** в команде **trap** вне функции, команда *аргумент* выполняется при выходе из командного интерпретатора. Если указан сигнал **ERR**, *аргумент* будет выполняться каждый раз, когда какая-то команда завершится с ненулевым статусом выхода. Если указан *сигнал* **DEBUG**, *аргумент* будет выполняться после каждой команды.

Среда, в которой командный интерпретатор выполняет обработчик сигнала **EXIT**, будет идентичной среде сразу после выполнения последней команды и до срабатывания обработчика сигнала **EXIT**.

При каждом вызове обработчика *аргумент* будет обрабатываться как в команде:

```
eval "$аргумент"
```

Сигналы, игнорировавшиеся при входе в неинтерактивный командный интерпретатор, не могут быть перехвачены или переустановлены, хотя при попытке сделать это никакое сообщение об ошибке выдавать не нужно. Интерактивный командный интерпретатор может переустанавливать или перехватывать сигналы, игнорируемые при входе. Обработчики будут действовать в данном командном интерпретаторе, пока не будут явно изменены другой командой **trap**.

При входе в порожденный командный интерпретатор обработчики устанавливаются в стандартные значения. Это не означает, что команду **trap** нельзя использовать в порожденном командном интерпретаторе для установки новых обработчиков.

Команда **trap** без аргументов будет выдавать в стандартный выходной поток список команд, ассоциированных с каждым сигналом. При этом используется следующий формат:

```
trap -- %s %s... <аргумент>, <сигнал>...
```

Командный интерпретатор будет форматировать результат, включая правильную расстановку кавычек, так, что он подходит для повторного ввода в командном интерпретаторе в качестве команд, задающих те же перехватчики. Например:

```
save_traps=$(trap)
```

```
...
```

```
eval "$save_traps"
```

Если указанное имя или номер сигнала недопустимы, будет возвращен ненулевой статус выхода; иначе возвращается 0. Как для интерактивных, так и для неинтерактивных командных

интерпретаторов недопустимые имена или номера сигналов не считаются синтаксической ошибкой и не вызывают аварийного прекращения работы командного интерпретатора.

Обработчики не выполняются, пока задание ждет завершения приоритетного процесса. Таким образом, обработчик сигнала **CHLD** не будет выполнен, пока не завершится работа приоритетного задания.

type *имя...*

Для каждого *имени* показывать, как оно будет интерпретироваться при указании в качестве имени команды.

** **typeset** [**±HLRZfirtux**[*n*]] [*имя*[=*значение*]]...

Устанавливает атрибуты и значения для переменных и функций командного интерпретатора. Когда команда **typeset** вызывается в функции, создается новый экземпляр каждой переменной *имя*.

Значения и типы переменных восстанавливаются при завершении выполнения функции. Можно задавать атрибуты из следующего списка:

-H

Этот флаг обеспечивает сопоставление имен файлов UNIX файлам хоста на не-UNIX машинах.

-L

Выравнивать влево и удалять начальные пробелы из значения. Если *n* - не ноль, оно определяет ширину столбца; в противном случае, она определяется по размеру первого присвоенного значения. При присвоении значения переменной, оно дополняется справа пробелами или усекается, при

необходимости, чтобы поместиться в поле. Начальные нули удаляются, если установлен также флаг **-Z**. Флаг **-R** отключается.

-R

Выравнивать вправо и дополнять значение начальными пробелами. Если **n** - не ноль, оно определяет ширину столбца; в противном случае, она определяется по размеру первого присвоенного значения. При присвоении значения переменной, оно дополняется слева пробелами или усекается, при необходимости, чтобы поместиться в поле. Флаг **-L** отключается.

-Z

Выравнивать вправо и заполнять начальными нулями, если первый непробельный символ - цифра и флаг **-L** не установлен. Если **n** - не ноль, оно определяет ширину столбца; в противном случае, она определяется по размеру первого присвоенного значения.

-f

Имена задают функции, а не переменные. Присвоения им невозможны и допускаются только флаги **-t**, **-u** и **-x**. Флаг **-t** включает трассировку выполнения для данной функции. Флаг **-u** вызывает пометку этой функции как неопределенной. Переменная **FRATH** будет просматриваться в поисках определения функции при ссылке на нее. Флаг **-x** позволяет использовать определение функции при вызове процедур командного интерпретатора по имени.

-i

Параметр - целочисленный. Это ускоряет выполнение арифметических операций. Если **n** - не ноль, она задает основание системы счисления; в противном случае система счисления для выдачи значения определяется по первому присвоенному значению.

-l

Все символы верхнего регистра переводятся в нижний регистр. Флаг верхнего регистра, **-u**, сбрасывается.

-r

Указанные **имена** помечаются как доступные только для чтения и их значения нельзя будет изменить в дальнейшем с помощью присвоений.

-t

Помечает переменные тегами. Теги определяет пользователь и они не имеют специального значения для командного интерпретатора.

-u

Все символы нижнего регистра переводятся в верхний регистр. Флаг нижнего регистра, **-l**, сбрасывается.

-x

Указанные **имена** помечаются для автоматического экспорта в среду выполняемых в дальнейшем команд.

Атрибут **-i** нельзя задавать вместе с **-R**, **-L**, **-Z**, или **-f**.

Использование **+** вместо **-** вызывает сброс этих флагов. Если **имена** не заданы, но флаги есть, выдается список имен переменных (и, опционально, их значений), для которых эти флаги установлены. (Использование **+** вместо **-** не позволяет выдавать значения.) Если не заданы ни имена, ни флаги, выдаются имена и атрибуты всех переменных.

ulimit [-HSacdfnstv] [ограничение]

Установить или выдать ограничение ресурсов. Ресурсы, которые можно ограничивать, перечислены ниже. Многие системы не позволяют задавать некоторые из этих ограничений. Ограничение указанного ресурса устанавливается, когда задан параметр **ограничение**. **Ограничение** может быть числом в единицах, указанных для каждого ресурса ниже, или значением **unlimited**. Флаги **H** и **S** задают, устанавливается ли для данного ресурса жесткое или мягкое ограничение. Жесткое ограничение, если установлено, не может быть превышено. Мягкое ограничение может быть превышено, вплоть до соответствующего жесткого ограничения. Если не указано ни **H**, ни **S**, задаются оба ограничения. Если **ограничение** не указано, выдается текущее ограничение.

В этом случае выдается мягкое ограничение, если явно не указана опция **H**. Если указано более одного ресурса, то перед каждым значением выдается имя и единица измерения.

-a

Выдает все текущие ограничения ресурсов.

-c
Максимальный размер дампов памяти в блоках по 512 байтов.

-d
Ограничение, в Кбайтах, на размер области данных.

-f
Максимальное количество 512-байтовых блоков в файлах, записываемых порожденными процессами (читать можно файлы любого размера).

-n
Количество дескрипторов файлов плюс 1.

-s
Ограничение, в Кбайтах, на размер области стека.

-t
Количество секунд процессорного времени, которые может использовать каждый процесс.

-v
Размер виртуальной памяти, в Кбайтах.
Если опции не указаны, предполагается опция **-f**.

umask [-S] [*маска*]

Пользовательская маска создания файла устанавливается равной указанной *маске* (см. **umask(2)**). *Маску* можно задавать в виде восьмеричного числа или в символьном виде, как описано на странице справочного руководства **chmod(1)**. Если указано символьное значение, новое значение **umask** является дополнением к результату применения *маски* к дополнению предыдущего значения **umask**. Если *маска* не указана, выдается текущее значение маски. Флаг **-S** приводит к выдаче маски в символьном виде.

unalias *имя...*

Указанные в списке *имен* псевдонимы удаляются из списка псевдонимов.

unset [-f] *имя...*

Указанные в списке *имен* переменные теряют значения, т.е. их значения и атрибуты удаляются. Переменные, доступные только для чтения, нельзя сбросить с помощью **unset**. Если установлен флаг **-f**, то *имена* считаются именами функций. Сброс переменных **ERRNO**, **LINENO**, **MAILCHECK**, **OPTARG**, **OPTIND**, **RANDOM**, **SECONDS**, **TMOUT** и **_** лишает их специального смысла, даже если им в дальнейшем присваиваются значения.

*** wait [*задание*]**

Ждать завершения указанного *задания* и сообщать о его статусе завершения. Если *задание* не указано, командный интерпретатор ждет завершения всех активных на момент выполнения этой команды порожденных процессов. Статус выхода этой команды совпадает со статусом выхода процесса, завершения которого ждали. Описание и формат *задания* см. в разделе "**Задания**".

whence [-pv] *имя...*

Для каждого *имени* показать, как оно будет интерпретироваться при указании в качестве имени команды.

Флаг **-v** выдает более подробный отчет.

Флаг **-p** требует искать указанное *имя* по путям, даже если это имя является псевдонимом, функцией или зарезервированным словом.

Вызов

Если командный интерпретатор вызван с помощью **exec(2)** и первый символ нулевого аргумента (**\$0**) представляет собой **-**, то предполагается, что это начальный командный интерпретатор, и команды читаются из файла **/etc/profile** и либо из файла **.profile** в текущем каталоге, либо из **\$HOME/.profile**, если хоть один из них существует. Затем команды читаются из файла, имя которого задается переменной среды **ENV**, если он существует. Если флаг **-s** не указан, но есть *аргументы*, то выполняется поиск по путям для первого аргумента, чтобы определить имя сценария для выполнения. Указанный в виде аргумента сценарий должен быть доступен для чтения, а любые установки битов **setuid** и **setgid** игнорируются. Если сценарий не найден по путям, аргумент обрабатывается так, как если бы он задавал имя встроенной команды или функции. Затем

команды читаются так, как описано ниже. При вызове командным интерпретатором обрабатываются следующие флаги:

-c

Читать команды из *строки_команд*. Установить значение специального параметра 0 равным значению операнда *имя_команды*, а позиционные параметры (\$1, \$2 и т.д.) - равными остальным *аргументам*, последовательно. Из стандартного входного потока команды читаться не будут.

-s

Если указан флаг -s, или аргументов больше не осталось, команды читаются из стандартного входного потока. Результаты работы командного интерпретатора, за исключением результатов перечисленных выше специальных команд, выдаются в файл с дескриптором 2.

-i

Если указан флаг -i или входной и выходной потоки командного интерпретатора связаны с терминалом (как говорит вызов **ioctl(2)**), такой командный интерпретатор является интерактивным. В этом случае сигнал **TERM** игнорируется (так что **kill 0** не завершает работу интерактивного командного интерпретатора), а сигнал **INTR** перехватывается и игнорируется (так что команду **wait** прервать нельзя). В любом случае, **QUIT** игнорируется командным интерпретатором.

-r

Если указан флаг -r, командный интерпретатор является ограниченным. Остальные флаги и аргументы представлены ранее, в описании команды **set**.

Только rksh

rksh используется для настройки регистрационных имен и сред выполнения, возможности которых строже контролируются, чем при использовании стандартного командного интерпретатора. Действия **rksh** идентичны **ksh**, за исключением того, что следующие действия не разрешены:

- изменение каталога (см. **cd(1)**)
- установка значения переменных **SHELL**, **ENV** и **PATH**
- задание путей или имен команд, содержащих /
- перенаправление вывода (>, >|, <> и >>)
- изменение группы (см. **newgrp(1)**).

Перечисленные выше ограничения вступают в силу после интерпретации файлов **.profile** и **ENV**. Когда надо выполнить команду, являющуюся процедурой командного интерпретатора, **rksh** вызывает **ksh** для ее выполнения. Таким образом, можно создавать для конечного пользователя процедуры, использующие все возможности стандартного командного интерпретатора, но ограничить ему выбор команд; эта схема предполагает, что конечный пользователь не имеет прав на запись и выполнение в том же каталоге. Общий эффект этих правил состоит в том, что создатель файла **.profile** имеет полный контроль над действиями пользователя, выполняя гарантированные действия по настройке и оставляя пользователя в соответствующем каталоге (вероятно, не в начальном). Системный администратор часто создает каталог с командами (т.е., **/usr/rbin**), которые можно затем безопасно вызывать в **rksh**.

ОШИБКИ

Ошибки, выявленные командным интерпретатором, например, синтаксические, вызывают возврат командным интерпретатором ненулевого статуса выхода. В противном случае, командный интерпретатор возвращает статус выхода последней выполненной команды (см. также описание команды **exit** выше). Если

командный интерпретатор используется не интерактивно, то выполнение файла интерпретатора не контролируется. При выявлении ошибок времени выполнения, командный интерпретатор выдает имя команды или функции, а также причину ошибки. Если номер строки, в которой произошла ошибка, больше единицы, то выдается также номер строки в квадратных скобках ([]) после имени команды или функции. Для неинтерактивного командного интерпретатора, ошибка, возникшая в специальной встроенной команде или в утилите другого типа, вызовет выдачу командным интерпретатором диагностического сообщения в стандартный поток ошибок и завершение работы, как показано в следующей таблице:

Ошибка	Специальная встроенная команда	Другие утилиты
Синтаксическая ошибка с точки зрения языка командного интерпретатора	работа завершается	работа завершается
Синтаксическая ошибка с точки зрения утилиты (ошибка в опции или операнде)	работа завершается	работа не завершается
Ошибка перенаправления	работа завершается	работа не завершается
Ошибка при присвоении переменной	работа завершается	работа не завершается
Ошибка подстановки	работа завершается	работа завершается
Команда не найдена	не бывает	работа может завершиться
Сценарий, вызываемый через точку, не найден	работа завершается	не бывает

Ошибка подстановки возникает при выполнении подстановок командным интерпретатором (например, `${x!u}`, поскольку `!` - недопустимый оператор); реализация может считать их синтаксическими ошибками, если сможет выявить их при разбиении команды на лексемы, а не при выполнении подстановок.

Если любая из ошибок, для которой в таблице указано "работа завершается (может завершиться)", происходит в порожденном командным интерпретатором, порожденный командный интерпретатор завершает (может завершить) работу с ненулевым статусом, но сценарий, из которого он порожден, не закончит работу из-за этой ошибки.

Во всех представленных в таблице случаях, интерактивный командный интерпретатор будет выдавать диагностическое сообщение в стандартный поток ошибок без завершения работы.

ИСПОЛЬЗОВАНИЕ

Описание поведения **ksh** и **rksh** при работе с файлами, большими или равными 2 Гбайтам (2*31 байтам), см. на странице справочного руководства **largefile(5)**.

СТАТУС ВЫХОДА

Каждая команда имеет статус выхода, который может влиять на поведение других команд. Статус выхода команд, не являющихся внешними утилитами, описан в этом разделе. Статус выхода стандартных утилит описан в соответствующих им разделах.

Если команда не найдена, статус выхода будет равен 127. Если файл, соответствующий имени команды, найден, но он не является выполняемой утилитой, статус выхода будет равен 126. Приложения, вызывающие утилиты без использования командного интерпретатора, должны использовать эти значения статуса выхода для сообщения об аналогичных ошибках.

Если в команде возникает ошибка при разбиении на слова или перенаправлении, ее статус выхода будет больше нуля.

При выдаче статуса выхода через специальный параметр `?` командный интерпретатор будет выдавать все имеющиеся восемь битов статуса выхода. Статус выхода команды, прекратившей работу из-за получения сигнала, будет иметь значение больше 128.

ФАЙЛЫ

`/etc/profile`
`/etc/suid_profile`


```
$HOME/.profile
/tmp/sh*
/dev/null
```

АТТРИБУТЫ

Описание следующих атрибутов см. на странице справочного руководства **attributes(5)**:

/usr/bin/ksh
/usr/bin/rksh

ТИП АТТРИБУТА	ЗНАЧЕНИЕ АТТРИБУТА
Доступен в пакете	SUNWcsu
CSI	Включено

/usr/xpg4/bin/ksh

ТИП АТТРИБУТА	ЗНАЧЕНИЕ АТТРИБУТА
Доступен в пакете	SUNWxcu4
CSI	Включено

ССЫЛКИ

[cat\(1\)](#), [cd\(1\)](#), [chmod\(1\)](#), [cut\(1\)](#), [echo\(1\)](#), [env\(1\)](#), [getoptcv\(1\)](#), [jobs\(1\)](#), [login\(1\)](#), [newgrp\(1\)](#), [paste\(1\)](#), [ps\(1\)](#), [shell_builtins\(1\)](#), [stty\(1\)](#), [test\(1\)](#), [vi\(1\)](#), [dup\(2\)](#), [exec\(2\)](#), [fork\(2\)](#), [ioctl\(2\)](#), [lseek\(2\)](#), [pipe\(2\)](#), [ulimit\(2\)](#), [umask\(2\)](#), [wait\(2\)](#), [rand\(3C\)](#), [signal\(3C\)](#), [a.out\(4\)](#), [profile\(4\)](#), [attributes\(5\)](#), [environ\(5\)](#), [largefile\(5\)](#), [signal\(3HEAD\)](#), [XPG4\(5\)](#)

[Morris I. Bolsky and David G. Korn, The KornShell Command and Programming Language, Prentice Hall, 1989.](#)

ПРЕДУПРЕЖДЕНИЯ

Использование сценариев командного интерпретатора с установленным битом **setuid** настоятельно не рекомендуется.

ПРИМЕЧАНИЯ

Если выполняется команда, представляющая собой отслеживаемый псевдоним, а затем команда с тем же именем устанавливается в каталог, находящийся в пути поиска перед каталогом, в котором находилась исходная команда, командный интерпретатор будет продолжать выполнять исходную команду. Используйте опцию **-t** команды **alias** для исправления этой ситуации.

Некоторые очень старые сценарии командного интерпретатора содержат **^** как синоним символа конвейера, **|**.

Использование встроенной команды **fc** в составной команде приведет к исчезновению всей команды из файла истории.

Встроенная команда **.** *файл* читает весь файл до выполнения любых команд. Поэтому команды **alias** и **unalias** в файле не будут применяться к функциям, определенным в файле.

Когда командный интерпретатор выполняет сценарий, пытающийся вызвать несуществующий командный интерпретатор, выдается ошибочное диагностическое сообщение о том, что файл сценария не существует.

Последнее изменение: 26 февраля 1999 года

Copyright (no c) 2005 [В. Кравчук](#), [OpenXS Initiative](#), перевод на русский язык