# 1 Тема 3. Теоретическая часть

Многие теоретические знания, представленные в данном разделе уже использова- лись при изучении материала предыдущих двух тем или известны студенам из учебного материала других дисциплин. Тем не менее, имеется ряд моментов, кото- рые необходимо хорошо усвоить, прежде чем изучать вопросы, изложенные в пос- ледующих темах. Кроме того, для правильного понимания сути многих вопросов, необходим адекватный язык, на котором эту суть можно объяснить. Для этой цели и предназначен учебный материал данного раздела.

Чтобы учебный материал получил должный уровень конкретизации, он опирается на практическую часть задач загрузки ОС УПК АСУ, изложенных в [4] и доступных в виде файла *upk\_asu.pdf* на рабочем столе пользователя ***upk***.

## Языки программирования и командные интерпретаторы

**Основным языком** программирования *ядра ОС* и другого *системного ПО* являет- ся *язык* ***С***. Этот язык специально создавался для написания ОС и постепенно вы- теснил *языки Ассемблера*, которые, в большей степени чем ***С***, зависели от архитек- туры процессора, способов адресации памяти и других архитектурных особеннос- тей ЭВМ.

**С другой стороны**, *язык* ***С*** *также сильно привязан к архитектуре ЭВМ и ОС*:

* + - *через машинный язык*, в который компилируется исходный текст языка ***С***, исполняющийся конкретным процессором;
    - *через структуру исполняемых программ*, которые определяются ОС;
    - *через библиотеку* ***libc***, связывающую, *через системные вызовы*, ПО режима пользователя с ПО ядра ОС.

**Очевидно**, что язык ***С*** мало пригоден для создания масштабных приложений. Для этих целей используются языки объектно-ориентированного программирования (***ООП***), такие как ***С++****,* ***C#****,* ***Java*** и другие.

Ранее отмечалось, что, являясь базовым ПО ЭВМ, ОС охватывает ту часть програм- много обеспечения компьютера, которое называется *системным ПО*.

**Целевое назначение ОС** — создание *виртуальной машины* или *среды исполнения*

для работы *системного*, *прикладного* и *инструментального* ПО компьютера. **Важнейшая функция** такой виртуальной машины — *управление программным обеспечением ЭВМ, работающим в режиме пользователя*.

**Все ОС**, для целей управления ПО ЭВМ, используют специальные языки програм- мирования, которые называются *командными интерпретаторами* или *shell*:

* + - ОС MS Windows, в качестве shell, использует язык ***batch*** или ***cmd***.
    - ОС Linux — ***bash*** (Bourne Again Shell) и ***sh*** (Bourne Shell).
    - ОС UNIX — ***sh*** (Bourne Shell), ***csh*** (C Shell), ***ksh*** (Korn Shell), ***tcl*** и другие.

Несмотря на имеющиеся различия, все ***командные языки*** имеют сходный синтак- сис и построены по одному принципу - *каждая строка языка* рассматривается как *команда с аргументами*, требующая немедленного исполнения:

***команда [ аргумент\_1 аргумент\_2 … ] конец\_строки***

*Строка* — последовательность слов, разделенных символами пробела или табуля- ции и заканчивающаяся символами конца строки.

*Команда* — слово, обозначающее действие:

* + - *встроенная команда* исполняется непосредственно интерпретатором;
    - *имя программы ОС*, которую интерпретатор запускает.

*Аргумент* — слово, интерпретируемое в контексте команды.

*Конец\_строки* — набор:

* + - *управляющих символов* языка;
    - *управляющих слов* языка.

**Во времена**, когда графический интерфейс ОС отсутствовал, командные интерпре- таторы были единственным средством взаимодействия человека и ЭВМ. И сейчас они являются таковыми, когда графическая система выходит из строя.

**Замечание** Работа любого командного языка опирается на специальное устройство ЭВМ, которое называется ***терминалом***.

**Терминал** — это последовательное (символьное) устройство ЭВМ, обычно обозначае- мое ***tty*** и обеспечивающее ввод с клавиатуры потока символов, обрабатывающее эти символы, а затем выводящее результат обработки на экран (дисплей) ЭВМ.

**Именно терминал** обеспечивает ввод команды с клавиатуры, отображение каждого сим- вола на экране (дисплее) и передачу введенной строки интерпретатору ***shell***, после нажатия коавиши «***Enter***».

*Подробное изучение терминалов выходит за рамки нашего курса!*

**По функциональным возможностям** все языки приблизительно одинаковые, хотя в деталях могут различаться синтаксисом.

**В частности**, язык ***tcl*** разрабатывался с возможностью использования превдогра- фики, что для своего времени было достаточно перспективно.

**Совместное** современное существование различных языков вызвано:

* + - *силой привычки*, авторскими правами и рядом корпоративных интересов;
    - *наличием достаточно большого количеста ПО*, написанного на них.

**Общая проблематика интерпретаторов** заключается в том, что увеличение функ- циональных возможностей ***shell*** влечет:

* + - *увеличение размера* интерпретирующей прогаммы и уменьшение скорости ее загрузки;
    - *повышенный расход* оперативной памяти компьютера.

## Базовый язык shell (sh)

Как отмечено выше, термин ***shell*** применяется в двух аспектах:

* *как расширительное обозначение* всех командных интерпретаторов ОС;
* *как конкретизация* интерпретатора ***sh*** (***Bourne Shell***).

**Выбор** языка ***sh*** обоснован следующими причинами:

* *стандартизация языка* в рамках проекта ***POSIX 1003.2*** — стандарта мобиль- ных систем;
* *современные ядра* ОС Linux запускают интерпретатор ***sh***, при обнаружении в корне файловой системы скриптов (*сценариев*) ***init*** или ***linuxrc***;
* *интерпретатор* ***bash***, используемый ОС Linux, можно рассматривать как прямое функциональное расширение интерпретатора ***sh***.

**POSIX** (***Portable Operating System Interface for Unix***) — переносимый интерфейс операционных систем UNIX.

**POSIX** — набор стандартов, описывающих интерфейсы между операционной сис- темой и прикладной программой. Закреплен международным стандартом ***ISO/IEC 9945*** и может использоваться не только для ОС UNIX.

**Определим** ряд ***метапонятий***, которые ***shell*** учитывает в своей работе:

* *shell* — это программа (утилита или командный интерпретатор) *sh*, обычно

*/bin/sh*, который работает в *среде ОС*: ***в пользовательском режиме****;*

* *запустить sh* может любой процесс, посредством одного из системных вызо- вов ***exec\*()***; при этом, ***sh*** будет использовать среду ОС, в которой работала вызывающая программа;
* *процесс sh* может сам порождать необходимое количество *дочерних процес- сов*, посредством системного вызова ***fork()***, отслеживая их работу и анализи- руя их *коды завершения*;
* *нулевой целочисленный код завершения* означает ***нормальное выполнение ко- манды*** дочерним процессом;
* *ненулевой целочисленный код завершения* означает ***ошибочное выполнение команды*** дочерним процессом и дополнительно интерпретируется, в зависи- мости от ситуации и режимов работы ***sh***.

**Замечание**

Если ***sh*** обнаружил *синтаксическую ошибку*, то выполнение ***shell*** *прекращается*, в про- тивном случае, возвращается код завершения ***последней*** выполненной команды.

**При запуске**, ***sh*** как и любая прикладная программа, *наследует все ресурсы* вызы- вающего процесса, включая открытые файлы.

**Замечание**

Когда программа ***login*** завершит проверку входа пользователя в систему, она три раза открывает соответствующий терминал: один раз — на чтение и два раза — на запись, а затем запускает ***shell***, обеспечивая ***системный ввод/вывод***.

На рисунке 1.1 показан ***системный ввод/вывод*** типового прикладного процесса.

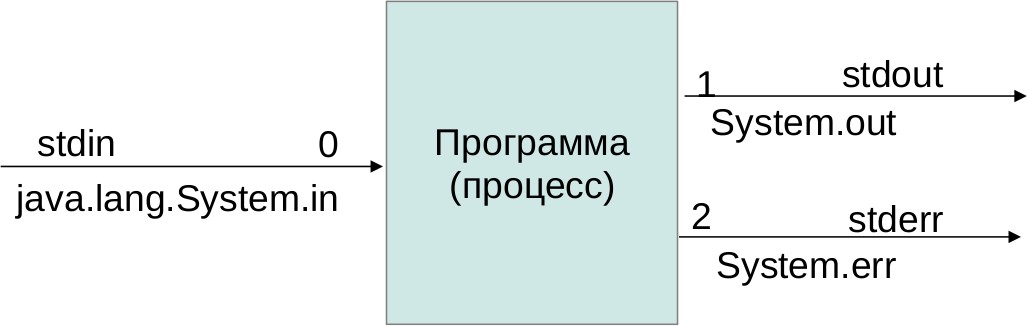


Рисунок 1.1 - Потоки ввода/вывода прикладной программы

**Каждая программа пользователя**, запущенная в виде процесса на компьютере, имеет:

* ***один*** системный ввод;
* ***два*** системных вывода.

*На уровне файловых дескрипторов shell*, говорят об устройствах:

* ***устройство 0*** — устройство ввода;
* ***устройство 1*** — устройство нормального вывода программы;
* ***устройство 2*** — устройство вывода ошибок.

*На уровне языка С*, мы имеем стандартные устройства:

* **stdin** — устройство ввода;
* **stdout** — устройство нормального вывода программы;
* **stderr** — устройство вывода ошибок.

*На уровне языка Java*, мы имеем три объекта:

* **java.lang.System.in** — объект канала ввода с клавиатуры;
* **java.lang.System.out** — объект канала нормального вывода;
* **java.lang.System.err** — объект канала вывода ошибок.

*Для чтения из потока ввода* (*обычно клавиатура*) используются различные моди- фикации функции ***read(...)***.

*Для вывода информации в потоки* **stdout** и **stderr** (*обычно консоль*) используются различные модификации функций ***write(…)*** и ***print(...)***.

**Замечание Особо**, следует обратить внимание на номера устройств (*целочисленные дескрипторы файлов*), которыми интенсивно манипулирует интерпретатор ***sh***.

**Например**, если процесс закрывает файл с дескриптором ***0***, а затем открывает новый файл. В результате, дескриптор нового файла будет равен ***0*** и процесс будет читать дан- ные из файла, как будто он читает с клавиатуры.

Все ***shell*** используют *свойства базовых категорий*, определенных понятиями:

*файл, пользователь и процесс*.

**В частности**, обычные файлы подразделяются на:

* *бинарные*, которые читаются процессом как последовательность байт, име- ющих значения от 0 до 255;
* *текстовые (символьные)*, в которых, в зависимости от кодировки, ряд значе- ний байт или не используются совсем или рассматриваются как управляю- щие, например: *10 — перевод строки; 13 — возврат каретки*.

Все интерпретаторы ***shell*** могут использовать текстовые файлы как программы. Такие файлы называют *скрипты* или ***сценарии***.

**В частности**, все shell следуют общим правилам:

* *символ* ***#*** используется как комментарий до конца строки;
* *сочетание символов* ***#!***, расположенных в первой позиции первой строки текстового файла, рассматривается как команда вызова конкретного интер-

претатора shell.

Чтобы определить на каком терминале работает пользователь, необходимо восполь- зоваться командой ***ps***, без аргументов. На рисунке 1.2 показано применение этой ко- манды в графической среде ОС Linux, которая показывает, что пользователь рабо- тает в псевдотерминале ***/dev/pts/1***.

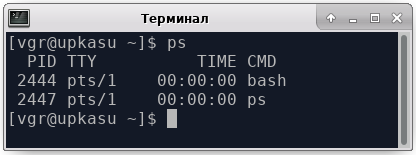


Рисунок 1.2 — Пример использования команды ps без аргументов в графической среде ОС Linux

**Замечание**

Операционная среда ОС Linux имеет 64 виртуальных терминала: ***/dev/tty0*** … ***/dev/tty63****.*

Переключение между терминалами ***/dev/tty1*** … ***/dev/tty7*** осуществляется комбинациями клавиш ***Alt-F1 … Alt-F7***.

**Программы** (*процессы*), которые не имеют ввода/вывода на какой-либо терминал, назы- ваются ***системными процессами***.

## Среда исполнения программ

**Среда выполнения** любой программы ОС подразделяется на:

* *структуру файловой системы ОС*, которую программа использует для ввода и вывода данных;
* *набор файлов конфигурации*, которые определяют параметры данных прог- раммы или дополнительные данные конфигурации среды исполнения;
* *системные переменные среды*, которые наследуются как из среды родитель- ского процесса, а также создаются или удаляются в процессе работы прог- раммы.

### Структура файловой системы ОС Linux

Для языка ***sh***, среда исполнения определяется условиями видимости *той части файловой системы*, которая соответствует *пользователю*, запустившему ***shell***. Как правило, для этих целей используется директория ***/home***. Например:

* *пользователь* ***asu*** имеет домашнюю директорию ***/home/asu***;
* *пользователь* ***upk*** имеет домашнюю директорию ***/home/upk***;

### Набор файлов конфигурации sh

В течении работы и запуска интерпретатора ***sh*** используются следующие конфигу- рационные файлы:

/etc/profile

$HOME/.profile

Для интерпретатора ***bash***, который является основным для обычных пользователей в ОС Linux, такими файлами являются:

/etc/profile

$HOME/.profile

$HOME/.bash\_profile

$HOME/.bash\_login

$HOME/.bash\_logout

$HOME/.bash\_history

### Переменные среды sh

Все интерпретаторы shell используют переменные среды, которые подразделяются:

* переменные ***системной среды*** — для системных процессов;
* переменные ***пользовательской среды*** — отдельно для каждого пользователя, вошедшего в систему.

**Sh** использует следующие *переменные* пользовательской среды, где:

* ***красным*** цветом указаны наиболее важные переменные;
* ***синим*** цветом — устаревшие переменные, используемые для совместимости.

|  |
| --- |
| HOME |
| Определяет домашний каталог пользователя. Подразумеваемый аргумент команды **cd** (1) - основной каталог. |
| PATH |
| Список имен каталогов для поиска команд. Подобные списки называются *списками поиска*. Элементы списка разделяются:   * **двоеточием**, для ОС UNIX; * **точка с запятой** для MS Windows; * **точка** - означает текущий каталог. |
| CDPATH |
| Список поиска для команды **cd**. |
| MAIL |
| Имя файла, куда будет помещаться почта;  если переменная MAILPATH не определена, shell информирует пользователя о поступлении почты в указанный файл. |
| MAILCHECK |
| Интервал между проверками поступления почты в файл, указанный переменными MAIL или MAILPATH.  По умолчанию интервал составляет 600 секунд (10 минут). При установлении значения 0 проверка будет производиться перед каж- дым выводом приглашения. |
| MAILPATH |
| Список имен файлов, разделенных двоеточием.  Если переменная определена, shell информирует пользователя о поступлении почты в каждый из указанных файлов.  После имени файла может быть указано (вслед за знаком %) со- общение, которое будет выводиться при изменении времени моди- фикации указанного файла (сообщение по умолчанию "You have mail"). |
| PS1 |
| Основное приглашение (по умолчанию "$ "). |
| PS2 |
| Вспомогательное приглашение (по умолчанию "> "). |
| IFS |
| Цепочка символов, являющихся разделителями в командной строке (по умолчанию это пробел, табуляция и перевод строки). |

|  |
| --- |
| SHACCT |
| Если значением этой переменной является имя файла, доступного для записи пользователем, shell будет помещать в него сведения о каждой выполняемой им процедуре. Для анализа сведений могут быть применены такие программы, как acctcom (1) и acctcms (1M). |
| SHELL |
| При запуске ***shell*** просматривает окружение в поисках этой пере- менной. Если она определена и файловая часть ее значения есть ***rsh***, shell становится *ограниченным* [см. ***rsh(1)***]. |

**Замечание**

Для переменных **PATH, PS1, PS2, MAILCHECK** и **IFS** имеются значения по умолчанию. Значения переменных **HOME** и **MAIL** устанавливаются командой ***login***(1).

Значения всех переменных можно вывести на консоль командой **env**.

### Окружение и экспорт переменных

**Окружение** [см. ***environ(3P)***] - это ***набор пар*** *(имя, значение)*, которые передается выполняемой программе так же, как и обычный список аргументов.

**Shell** взаимодействует с окружением несколькими способами:

* *при запуске*, окружение (среда работы) ***shell*** создается утилитой ***login***, а затем интерпретатор передает это окружение всем запускаемым программам;
* *присвоение значения какому-либо слову* не оказывает никакого влияния на окружение, пока не будет использована команда **export** (см. также **set -a**);
* *переменную среды можно удалить* из окружения командой ***unset***.

**Таким образом**, окружение каждой команды формируется из всех унаследованных языком shell:

* *пар* (имя, значение);
* *минус пары*, удаленные командой **unset**;
* *плюс все* модифицированные и измененные пары, к которым была применена команда **export**.

*Окружение простой команды* может быть модифицировано, если указать перед командой одно или несколько присваиваний переменным. Так, строки:

**TERM=vt100 команда**

и

**(export TERM; TERM=vt100; команда)**

*являются эквивалентными*, по крайней мере *с точки зрения окружения команды*.

## Командная строка: опции и аргументы

Каждая строка символов, которая заканчивается символом «***Enter***», рассматривает- ся ***терминалом*** как целостный набор данных, который необходимо передать интер- претатору ***shell***.

Сам интерпретатор анализирует полученную строку, проверяя ее синтаксис и раз- деляя ее на последовательность отдельных команд и их аргументов.

### Команды и аргументы команды

**Первое слово** в строке ***shell*** всегда воспринимается *как команда*, а остальные слова

* как *аргументы команды*:
  + чтобы в явном виде разделить команды в строке, следует использовать разде- литель — *точка с запятой*;
  + в случае, когда команда не помещается в одну строку, для продолжения ее на другой строке, используется символ — *обратный слэш*;
  + когда ***shell*** запускается посредством системного вызова **exec\*(...)** и *первым символом* ***нулевого аргумента*** *является* ***-***, то сначала читаются и выполняю- тся команды из файлов */etc/profile* и *$HOME/.profile*.

**Все команды** *shell*, условно, разделяются на две группы:

* + *встроенные команды* — команды, которые интерпретатор выполняет самос- тоятельно;
  + *внешние команды* — это программы и утилиты, которые ***shell*** ищет в файло- вой системе и, после проверки прав доступа, пытается запустить, используя системные вызовы *fork(...)* и *exec\*(...)*.

**Кроме того**, следует учесть что:

* + *под пробелом*, в дальнейшем, понимается не только собственно ***пробел***, но также и ***символ табуляции***;
  + *имя* - это последовательность ***букв***, ***цифр*** и ***символов подчеркивания***, начина- ющаяся с буквы или подчеркивания;
  + *параметр* - это ***имя***, ***цифра*** или любой из символов ***\*, @, #, ?, -, $, !***.

### Опции sh

**В общем случае**, интерпретатор ***shell*** может рассматриваться как команда с аргу- ментами. Синтаксис запуска интерпретатора ***shell*** имеет вид:

sh [-a] [-c цепочка\_символов][-e][-f][-h][-i][-k][-n][-r][-s][-t][-u][-v]

**[-x] [аргумент ...]**

где - квадратные скобки обозначают необязательные конструкции.

**Перечисленные флаги** (***опции***) интерпретируются ***shell*** при его запуске следую- щим образом:

* + если не указаны опции **-s** или **-c**, то первый аргумент рассматривается как

*имя файла*, содержащего команды;

* + остальные аргументы передаются этому командному файлу как *позиционные параметры*.

Наиболее часто используемые опции ***sh*** имеют следующую семантику:

|  |
| --- |
| *-c цепочка\_символов* |
| Команды берутся из ***цепочки\_символов***. |
| *-s* |
| Если аргументов больше нет, то команды читаются со стан- дартного ввода. Все оставшиеся аргументы рассматриваются как позиционные параметры.  *Вывод сообщений самого shell*, кроме специальных команд, направляется в файл с дескриптором 2 (стандартный прото- кол). |
| *-i* |
| если ввод и вывод ***shell*** ассоциированы с терминалом, shell выполняется в интерактивном режиме.  В этом случае сигнал завершения (0) игнорируется (то есть команда *kill 0* не приведет к завершению работы интерактив- ного shell'а).  Сигнал прерывания (2) перехватывается и игнорируется, поэ- тому выполнение системной функции ***wait*** (2) может быть прервано.  В любом случае, сигнал выхода (3) игнорируется. |
| *-r* |
| shell запускается как ограниченный [см. rsh(1)]. |

**Замечание**

Описание остальных флагов и аргументов приведено в описании команды **set**. Используйте команду:

***man set***

## Переменные sh

**Все переменные sh**, включая рассмотренные выше ***переменные среды***, называ- ются *параметрами*. Следует учесть,что:

* различаются ***два типа*** параметров: *позиционные* и *ключевые*;
* знак **$** используется *для подстановки значений параметра*.

**Позиционные параметры** обозначаются *цифрой* или одним из символов: *\*, @, #, ?, -, $, !*.

**Значения** *цифровых позиционных параметров* устанавливаются при вызове ***shell- функций*** или командой ***set***:

* *0 — параметр 0* — имя вызываемой функции;
* *1 — параметр 1* — аргумент 1;
* *2 — параметр 2* — аргумент 2 и далее.

**Значения** *следующих параметров* ***shell*** устанавливает автоматически:

* **\*** или **@** содержат все позиционные параметры, начиная с ***1***, разделенные пробелами;
* **#** количество позиционных параметров (десятичное);
* - флаги, указанные при запуске ***shell*** или установленные командой **set**;
* **?** десятичное значение, возвращенное предыдущей командой;
* **$** идентификатор процесса, в рамках которого выполняется ***shell***;
* **!** идентификатор последнего асинхронно запущенного процесса.

**Пример** *вывода позиционных параметров*, показан сценарием рисунка 1.3.

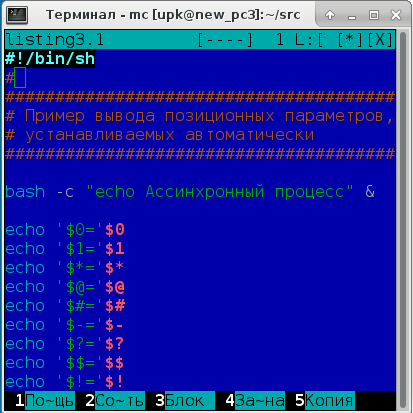


Рисунок 1.3 — Сценарий вывода позиционных параметров

Результат работы сценария показан рисунком 1.4.

Сам сценарий *listing3.1* находится в директории *~/src* пользователя **upk**.

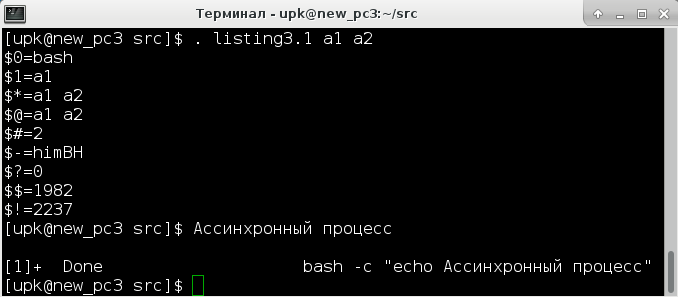


Рисунок 1.4 — Результат вывода сценария

**Ключевые параметры** *(переменные)* обозначаются именами.

**Значения** им присваиваются обычным способом:

имя=значение [имя=значение] ...

**Различаются** следующие *виды подстановок параметров*:

**${параметр}**

Подставляется значение параметра, если оно определено. Скобки ис- пользуются, только если за параметром следует буква, цифра или знак подчеркивания, и их нужно отделить от имени параметра. Вмес- то параметров **\*** и **@** подставляются все позиционные параметры, на- чиная с **$1**, разделенные пробелами.

**${параметр:-слово}**

Будем говорить, что параметр ***пуст***, если его значение ***не опреде- лено*** или является ***пустой цепочкой***. При данном способе подста- новки если параметр не пуст, подставляется его значение; в против- ном случае подставляется слово.

**${параметр:=слово}**

Если параметр ***пуст***, ему присваивается слово; после этого подстав- ляется значение параметра. Таким способом нельзя изменять значе- ния позиционных параметров.

**${параметр:?слово}**

Если параметр ***не пуст***, подставляется его значение; в противном случае в стандартный протокол выдается сообщение "параметр:сло- во" и выполнение shell'а завершается. Если слово опущено, то вы- дается сообщение "параметр:parameter null or not set".

**${параметр:+слово}**

Если параметр ***не пуст***, то подставляется слово; в противном случае - не подставляется ничего.

**Замечание После** проведения подстановок, полученная строка просматривается в поисках разде- лителей, которые берутся из системной переменной **IFS**, и расщепляется на аргументы. **Явные** пустые аргументы ***сохраняются***.

**Неявные** пустые аргументы ***удаляются***.

**Поскольку любой Shell** интерпретирует команды и аргументы команд как ***слова***, то следующие символы, если они не экранированы, *завершают предыдущее слово*:

**; & ( ) | ^ < > пробел табуляция перевод\_строки**

**Символ \** используется для экранирования *одиночных символов* и удаляется из сло- ва, перед выполнением команды, но сам экранируется одинарными кавычками.

Все эти символы могут экранироваться *одинарными* или *двойными кавычками*.

* **двойные кавычки** могут экранировать одинарную кавычку;
* **двойные кавычки** не мешают подстановке параметров.

## Специальные символы и имена файлов

В командах, работающих с *именами файлов*, возможно использование ***шаблонов***. **Шаблон** — набор символов, который добавляет или изменяет имена файлов, ис- пользуемые в командах интерпретаторов ***shell***, как аргументы.

**Типичные шаблоны**, применяемые к именам файлов:

* **\*** сопоставляется с произвольной цепочкой символов, в том числе и пустой;
* **?** сопоставляется с произвольным символом;
* **[...]** сопоставляется с любым, перечисленным в скобках символом. Пара сим- волов, разделенных знаком **-**, рассматривается как отрезок алфавита. Если за

символом **[** стоит знак **!**, то шаблону удовлетворяет любой символ, не пере- численный в скобках.

**Примеры** использования шаблонов:

* ls .. - вывод списка файлов родительского каталога;
* ls . - вывод списка файлов текущего каталога (каталог, в котором находится пользователь);
* ls .\* - вывод *всех* списка файлов и списка содержимого каталогов, с име- нами начинающимися с «точки», для текущего каталога (каталог, в котором

находится пользователь);

* ls .x\* - вывод списка имен файлов, начинающихся с ***.x***, для текущего ката- лога;
* ls .[a-c,x]\* - вывод списка имен файлов, начинающихся с ***.a, .b, .c, .x***, для текущего каталога;
* ls .config — вывод списка имен каталога *.config*;
* ls .config/\* — вывод списка имен файлов каталога *.config* и его катало- гов.

**Замечание**

*Не следует надеяться на интуицию!*

Обязательно следует проверить результаты вывода шаблонов в командной строке тер- минала.

## Стандартный ввод/вывод и переадресация

**Типичный процесс**, показанный ранее на рисунке 1.1, читает данные с клавиатуры (дескриптор файла 0) и выводит данные на экран терминала (дескрипторы файлов 1 и 2).

**В случае**, когда для чтения и записи данных используются другие источники информации, применяются следующие правила перенаправления (***переадресации***) ввода и вывода:

**<слово**

Использовать файл слово для стандартного ввода (дескриптор файла 0).

**>слово**

Использовать файл слово для стандартного вывода (дескриптор файла 1). Если файла нет, он создается; если есть, он опустошается.

**>>слово**

Использовать файл слово для стандартного вывода. Если файл существует, то выводимая информация добавляется в конец, то есть, сначала производится поиск конца файла; в противном случае файл создается.

**<<[-]слово**

Читается информация со стандартного ввода, пока не встретится строка, сов- падающая со словом, или конец файла. Если после **<<** стоит -, то сначала из слова, а затем, по мере чтения, из исходных строк удаляются начальные

символы табуляции, после чего проверяется совпадение строки со словом. Если какой-либо из символов слова экранирован, никакой другой обработки исходной информации не производится; в противном случае делается еще следующее:

1. Выполняется подстановка параметров и команд.
2. Пара символов \перевод\_строки игнорируется.
3. Для экранирования символов \, $, ` нужно использовать \.

Результат описанных выше действий становится стандартным вводом команды.

**<&цифра**

Производить стандартный ввод из файла, ассоциированного с дескриптором цифра.

**>&цифра**

Производить стандартный вывод в файл, ассоциированный с дескриптором цифра.

**<&-** Стандартный ввод команды закрыт.

**>&-** Стандартный вывод команды закрыт. **цифра<&-** Закрыть ***ввод*** дескриптора **цифра**. **цифра>&-** Закрыть ***вывод*** дескриптора **цифра**.

Если любой из этих конструкций предшествует цифра, она определяет дескрип- тор (вместо подразумеваемых дескрипторов 0 или 1), который будет ассоции- рован с файлом, указанным в конструкции. Например, строка:

**... 2>&1**

ассоциирует дескриптор 2 (стандартный протокол) с файлом, связанным в дан- ный момент с дескриптором 1.

Важен порядок переназначения: shell производит переназначение слева направо. Так, строка:

**... 1>f 2>&1**

сначала ассоциирует дескриптор ***1*** с файлом ***f***, а затем дескриптор ***2*** с тем же файлом. Если изменить порядок переназначения, стандартный протокол будет назначен на терминал (если туда был назначен стандартный вывод), а затем стандартный вывод будет переназначен в файл ***f***.

Если команда состоит из нескольких простых команд, переназначение для всей команды будет выполнено перед переназначениями для простых команд. Таким образом, shell выполняет переназначения сначала для всего списка, за тем: для каждого входящего в него конвейера, для каждой команды конвейера, для каждого списка из каждой команды.

Если команда заканчивается знаком **&**, то стандартный ввод команды переназначается на пустой файл ***/dev/null***. В противном случае, окружение для выполнения команды

содержит дескрипторы файлов запустившего ее shell'а, модифицированные специфи- кациями ввода/вывода.

**Замечание** Следует очень внимательно работать с перенаправлениями, поскольку они являются разделителями прав доступа. Например, две команды, подключающие архив студента к рабочей области ОС УПК АСУ, являются разными по правам доступа и не обеспечивают нужный результат:

gzip -cdvk /run/media/FC99-4744/asu64upk/themes/os-home.ext4fs.gz > \

/run/basefs/asu64upk/themes/os-home.ext4fs

sudo gzip -cdvk /run/media/FC99-4744/asu64upk/themes/os-home.ext4fs.gz > \

/run/basefs/asu64upk/themes/os-home.ext4fs

Хотя вторая команда подвергается воздействию команды *sudo*, ее действие распрост- раняется только до оператора перенаправления, что не позволяет записывать результат на защищенные устройства.

*Чтобы устранить указанный недостаток*, используются отдельные сценарии *from-gzip* и *to-gzip*, содержимое которых показано на рисунках 1.5 и 1.6. Применение команды *sudo* к сценарию обеспечивает командам нужный уровень привилегий.

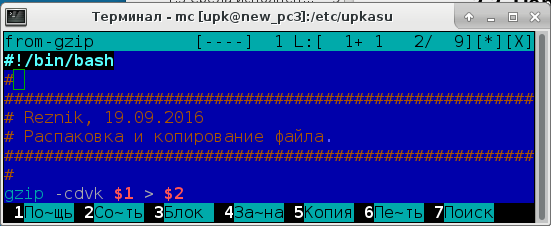


Рисунок 1.5 — Сценарий распаковки архива темы

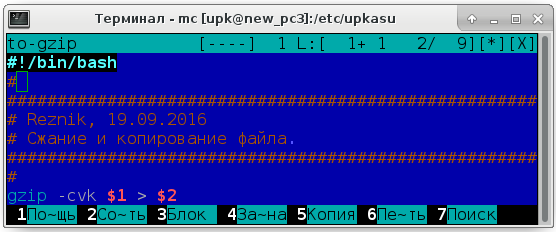


Рисунок 1.6 — Сценарий сжатия рабочей области и записи его в нужное место

## Программные каналы

Рассмотренные выше метаопределения и элементарные понятия языка ***shell***, опира- ются на понятие *простой команды*.

**Простая команда** - это последовательность слов, разделенных пробелами:

* *Первое слово* определяет имя команды, которая будет выполняться, а остав- шиеся слова передаются команде в качестве аргументов.
* *Имя команды* передается как ***аргумент 0*** [см. ***exec(2)***].
* *Значение* простой команды - это ее код завершения: ***0*** - если она выполнилась нормально, или (**128 + код ошибки**), если ненормально [см. также ***signal(2)***].

Общие конструкции языка ***shell*** используют *специальные файлы ОС*, которые назы- ваются *каналами* (***неименованными каналами***).

**Каналы** — специальные файлы ОС, создаваемые посредством системного вызова ***pipe(...)*** и служащие для *организации обмена данными* (***сообщениями***) между *процессами* (***программами***).

В языке ***sh***, для организации программных каналов между ***простыми командами***, используется понятие *конвейер*.

**Конвейер** - это последовательность команд, разделенных знаком **|** (***вертикальная черта***).

При этом:

* *Стандартный вывод* всех команд, кроме последней, направляется посред- ством системного вызова ***pipe(2)*** на стандартный ввод следующей команды конвейера.
* *Каждая команда* выполняется как ***самостоятельный процесс***.
* *shell ожидает* завершения последней команды. Ее код завершения станови- тся ***кодом завершения конвейера***.

*Список* - это последовательность *одного* или *нескольких конвейеров*, разделенных символами **;**, **&**, **&&** или **||** и, быть может, заканчивающаяся символом **;** или **&**.

Из четырех указанных операций:

* **;** и **&** имеют равные приоритеты, ***меньшие***, чем у **&&** и **||**.
* Приоритеты последних также равны между собой.
* Символ **;** означает, что конвейеры будут выполняться **последовательно**.
* Символ **&** означает, что конвейеры будут выполняться ***параллельно*** (то есть shell не ожидает завершения конвейера).
* Операция **&&** означает, что список, следующий за ней, будет выполняться лишь в том случае, если ***код завершения*** предыдущего конвейера ***нулевой***.
* Операция **||** означает, что список, следующий за ней, будет выполняться лишь в том случае, если код завершения предыдущего конвейера ненулевой.
* В списке, в качестве ***разделителя конвейеров***, вместо символа **;** можно ис- пользовать ***символ перевод строки***.

**Замечание**

Ранее отмечено, что двойные и одинарные кавычки используются ***shell*** для *экраниро-*

*вания последовательности слов*, с целью рассмотрения этой последовательности как

*отдельный аргумент*.

**Двойные кавычки** разрешают подстановки ключевых и позиционных параметров.

**Дополнительно**, ***shell*** использует ***обратные кавычки***: ***shell*** читает цепочки симво- лов, заключенные в обратные кавычки, и интерпретирует их ***как команды***.

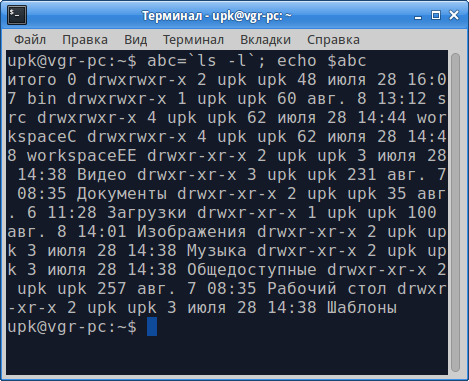
**Такие команды** выполняются в месте их использования. Например,

abc=`ls -l`;

здесь — ключевому параметру **abc** будет присвоен результат выполнения команды **ls -l** — список имен файлов текущей директории. Результат работы данного при- мера приведен на рисунке 1.7.

**Замечание**

Такого же результата можно добиться командой:



abc=$(ls -l);

Рисунок 1.7 — Пример использования обратных кавычек

## Сценарии

*Простую команду* или *простой конвейер* можно набрать и выполнить в окне консо- ли (терминала). *Сложные конструкции* языка ***sh*** — *программы* — пишутся в фай- лах, которые называются *сценариями*.

**Сценарий** — последовательность *простых команд* и *конвейеров*, оформленных с помощью *управляющих конструкций*.

### Управляющие конструкции sh

Язык ***sh*** содержит следующие *управляющие конструкции* (см. ***man sh***).

**for имя [in слово ...] do список**

**done**

При каждой итерации переменная **имя** принимает следующее значе-

ние из набора ***in слово ....***

Если конструкция ***in слово ...*** опущена, то **список** выполняется для каждого позиционного параметра.

**case слово in**

**[шаблон [| шаблон] ...) список ;;]**

**...**

**esac**

Выполняется **список**, соответствующий первому **шаблону**, успешно сопоставленному со словом. Формат шаблона тот же, что и исполь- зуемый для генерации имен файлов, за исключением того, что в шаблоне не обязательно явно указывать символ **/**, начальную точку и их комбинацию: элемент шаблона **\*** может успешно сопоставляться и с ними.

**if список\_1 then список\_2**

**[elif список\_3 then список\_4]**

**...**

**[else список\_5] fi**

Выполняется **список\_1** и если код его завершения 0, то выполняется

**список\_2**, иначе - **список\_3** и если код его завершения 0, то выпол- няется **список\_4** и т.д. Если же коды завершения всех списков, ис- пользованных в качестве условий, оказались ненулевыми, выполня- ется ***else***-часть (**список\_5**). Если ***else***-часть отсутствует, и ни одна ***then***-часть не выполнялась, возвращается нулевой код завершения.

**while список\_1 do список\_2**

**done**

Пока код завершения последней команды ***списка\_1*** есть 0, выпол-

няются команды ***списка\_2***. При замене служебного слова ***while*** на ***until***, условие продолжения цикла меняется на противоположное. Если команды из ***списка\_2***, не выполнялись вообще, код заверше- ния станавливается равным нулю.

**(список)**

Группировка команд для выполнения их *порожденным shell'ом*.

**{список;}**

Группировка команд для выполнения их *текущим shell'ом*.

**имя ( )**

**{список;}**

Определение функции с заданным ***именем***. Тело функции - ***список***, заключенный между { и }.

Следующие слова трактуются языком ***sh*** как ***ключевые***, если они являются первым словом команды и не экранированы:

**if then elif else fi case in esac**

**for while until do done**

**{ }**

### Примеры сценариев

**Изучив** понятие сценария, закрепим предыдущий учебный материал тремя приме- рами, решающими следующие задачи:

* *пример конфигурационного файла*, формируемого во время выполнения заг- рузки ОС и содержащего параметры передаваемые из GRUB в ядро ОС (см.

рисункок 1.8);

* *чтение архива* и *подключение его* к рабочей директории студента (см. листинг 1.1);
* *запись архива* на личный flashUSB (см. листинг 1.2).

**Замечание**

Все приведенные примеры сценариев можно найти в директории ***/etc/upkasu***.

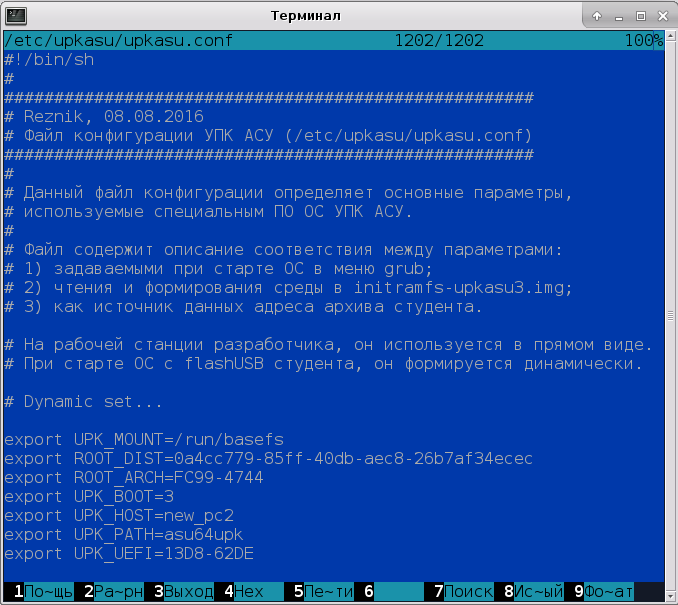


Рисунок 1.8 — Сценарий экспорта параметров ПО УПК АСУ

*Листинг 1.1 — Сценарий mount-upk.sh — подключение архива студента*

#!/bin/bash

###########################

# Специально для ОС УПК АСУ # Reznik, 02.09.2016

# Подключение темы обучения ###########################

# Используем функцию для вывода сообщения и продолжения работы программы: if\_msg()

{

echo -e "$@"

echo -en "Нажми клавишу$green Enter$gray для продолжения ... " read aa

}

# Используем функцию для вывода сообщения и выхода из программы: if\_exit()

{

echo -e "$@"

echo -e " "

echo -en "Нажми клавишу$green Enter$gray для завершения работы программы ...

"

read aa exit 0

}

#------------------------------------

# Подготовительная часть

#------------------------------------

. /etc/upkasu/upk.colors

echo -e "${green}Проверяем пользователя!$gray" [ "$USER" != "upk" ] || \

if\_exit "Ползователь upk не может устанавливать тему обучения!!!"

echo "Читаем парметры окружения..."

. /etc/upkasu/upkasu.conf

echo "Останавливаем все процессы пользователя upk, подключенные к директории /home/ upk..."

for xx in $(sudo lsof -t /home/upk/); do [ -d "/proc/$xx" ] || continue

echo -e "Удаляю процесс$red $xx $gray" sudo kill -9 "$xx"

done

echo "Проверяем наличие монтирования к директории /home/upk ..." x1=$(mount | grep "/home/upk")

[ "x$x1" == "x" ] || sudo umount -f /home/upk

x1=$(mount | grep /home/upk | cut -d ' ' -f 1 )

[ "x$x1" == "x" ] || if\_exit "${red}Не могу отмонтировать файл${gray}: $x1" #------------------------------------

# Определяем устройства архива:

xdev=$(blkid -U "$ROOT\_ARCH")

echo "Проверяем монтирование устройства архива. "

[ "x$xdev" != "x" ] || \

if\_exit "${red}Подмонтируйте$gray устройство архива... и попробуйте снова!.."

echo "Находим директорию архива. "

xdir=$(mount | grep "$xdev" | cut -d ' ' -f 3 )

[ "x$xdir" != "x" ] || if\_exit "${red}Не могу найти$gray директорию архива. "

if [ "x$xdir" == "x/" ]; then xdir=""

fi

echo "Находим директорию дистрибутива..." wdir="${UPK\_MOUNT}"

[ "x$wdir" != "x" ] || if\_exit "${red}Не могу найти$gray директорию дистрибутива... "

if [ "x$wdir" == "x/" ]; then wdir=""

fi

xdir="${xdir}/${UPK\_PATH}/themes"

[ -d "$xdir" ] || if\_exit "${red}Отсутствует$gray директория архива... " wdir="${wdir}/${UPK\_PATH}/themes"

[ -d "$wdir" ] || if\_exit "${red}Отсутствует$gray директория дистрибутива... "

#------------------------------------

# Выбираем тему обучения

echo "Читаем список доступных тем обучения..."

[ "$xdir" == "$wdir" ] && fl=home.ext4fs || fl=home.ext4fs.gz

xlist=$(ls "$xdir" | grep $fl )

[ "x$xlist" != "x" ] || if\_exit "${red}Нет тем${gray}, доступных для подключения...

"

echo -e "${green}Список доступных тем$gray обучения..." echo " "

for xx in ${xlist}; do

x1=$(echo $xx | cut -d '-' -f 1 ) echo -e "${yellow}${x1}$gray"

done

echo " "

echo -en "${green}Введи имя учебной темы$gray для подключения: " read aa

export THEME="$aa"

echo "Проверяем наличие архива. "

xarch="${xdir}/${THEME}-$fl"

[ -f "$xarch" ] || if\_exit "${red}Отсутствует файл архива${gray}: ${xarch}" warch="${wdir}/${THEME}-home.ext4fs"

#------------------------------------

# Копируем тему обучения

echo "Запоминаю тему обучения. "

echo "export UPK\_THEME=$THEME" > ~/.upk\_theme if [ "$xarch" != "$warch" ]; then

echo "Копирую файл архива в рабочую область. "

### sudo cp -v "$xarch" "$warch"

sudo /etc/upkasu/from-gzip "$xarch" "$warch"

fi

#------------------------------------

# Монтирование рабочей области:

echo -e "${green}Монтируем$gray файловую систему пользователя$yellow upk$gray \ в директорию$blue /home/upk$gray ..."

sudo mount -t ext4 "$warch" /home/upk -o loop

[ "$?" == "0" ] || if\_exit "${red}Не могу смонтировать$gray $warch на /home/upk" if [ -f "/home/upk/.upk\_theme" ]; then

echo -e "\n"\

"Тема$blue $THEME -$green успешно подлючена...$gray \n"\

"Вам следует выйти из сессии пользователя$yellow asu$gray и подключитья \n"\ "как пользователь$yellow upk ${gray}!"

if\_exit " "

fi

if\_exit "Тема $THEME - подлючена (Необходима дополнительная проверка!!!)"

*Листинг 1.2 — Сценарий copy-to-flash.sh — копирование архива студента*

#!/bin/sh #

############################################################# # Reznik, 02.09.2016

# Копирование рабочей области пользователя upk на flashUSB #

# Файл сценария: /etc/upkasu/copy-to-flash.sh ############################################################# #

# Данный сценарий обеспечивает копирование рабочей области

# темы обучения на личный flashUSB студента.

# Используем функцию для вывода сообщения и выхода из программы: if\_exit()

{

echo -e "$@"

read -p "Нажми клавишу Enter..." aa exit 0

}

# Читаем парметры окружения:

. /etc/upkasu/upkasu.conf

. /etc/upkasu/upk.colors

. ~/.upk\_theme

echo "##########################################"

echo -e "${red}Копирование архива на flashUSB студента...${gray}\n" [ "$ROOT\_ARCH" != "$ROOT\_DIST" ] || \

if\_exit "${green}Рабочая область и архив - одно и тоже устройство!!!${gray}"

# Находим устройства архива и дистрибутива ОС:

xdev=$(blkid -U "$ROOT\_ARCH") [ "x$xdev" != "x" ] || \

if\_exit "${yellow}Необходимо вставить устройство архива и попробовать снова!..${gray}"

xdir=$(mount | grep "$xdev" | cut -d ' ' -f 3 ) [ "x$xdir" != "x" ] || \

if\_exit "${yellow}Подмонтируйте устройство архива... и попробуйте снова!..${gray}"

[ "x$xdir" != "x/" ] || xdir="" dira="${xdir}/${UPK\_PATH}/themes" [ -d "$dira" ] || \

if\_exit "Не могу найти директорию архива... Обратитесь к преподавателю!.."

# Проверяем монтирование рабочей области:

wdir="$UPK\_MOUNT"

[ "x$wdir" != "x/" ] || wdir="" fa="${wdir}/${UPK\_PATH}/themes/${UPK\_THEME}-home.ext4fs" xx=$(mount | grep "$fa" )

[ "x$xx" == "x" ] || \

if\_exit "${yellow}Сначала отключите тему, а потом - архивируйте!${gray}"

sudo /etc/upkasu/to-gzip "$fa" "${dira}/${UPK\_THEME}-home.ext4fs.gz" || \

if\_exit "${yellow}Не могу перенести архив... Обратитесь к преподавателю!..${gray}"

# Закончили копирование архива:

if\_exit "${green}Перенесено:$blue из ${fa}$red в ${dira}/${UPK\_THEME}-home.ext4fs.gz${gray}"

### Встоенные команды sh

Кроме управляющих конструкций, ***sh*** содержит ряд встроенных (*специальных*) ко- манд.

**Замечание**

*Если не оговорено иное*, *команды* выводят результаты в файл с *дескриптором 1*.

**:** Пустая команда. Возвращает нулевой код завершения.

**. файл**

Shell читает и выполняет команды из файла, затем возобновляется чтение со стандартного ввода; при поиске файла используется значе- ние переменной **PATH**.

**break [n]**

Выйти из внутреннего ***for*** или ***while*** цикла; если указано **n**,то выйти из **n** внутренних циклов.

**continue [n]**

Перейти к следующей итерации внутреннего ***for*** или ***while*** цикла; если указано **n,** то перейти к следующей итерации **n**-ого цикла.

**cd [каталог]**

Сделать текущим заданный каталог. Если каталог не указан, исполь- зуется значение переменной **HOME**. Переменная **CDPATH** определяет спи- сок поиска каталога. По умолчанию этот список пуст (то есть поиск производится только в текущем каталоге). Если каталог начинается с символа **/**, список поиска не используется.

**echo [аргумент ...]**

Выдать аргументы на стандартный вывод, разделяя их пробелами [см. также echo(1)].

**eval [аргумент ...]**

Выполнить команду, заданную аргументами ***eval***.

**exec [аргумент ...]**

Сменить программу процесса: в рамках текущего процесса команда, заданная аргументами exec, заменяет shell. В качестве аргументов могут быть указаны спецификации ввода/вывода и, если нет никаких других аргументов, будет лишь переназначен ввод/вывод текущего shell'а.

**exit [код\_завершения]**

Завершить выполнение shell'а с указанным кодом. При отсутствии аргумента код завершения определяется последней выполненной коман- дой. Чтение символа конца файла также приводит к завершению shell'а.

**export [переменная ...]**

Заданные переменные отмечаются для автоматического экспорта в ок- ружение выполняемых команд. Если аргументы не указаны, выводится список всех экспортируемых переменных. Имена функций не могут экс- портироваться. Имена переменных, экспортированных из родительского shell'а, выдаются только в том случае, если они были экспортирова- ны и из текущего shell'а.

**getopts**

Используется в shell-процедурах для поддержания синтаксических стандартов [см. intro(1)]; разбирает позиционные параметры и про- веряет допустимость задания опций [см. getopts(1)].

**hash [-r] [имя\_команды ...]**

Для каждого из указанных имен\_команд определяется и запоминается маршрут поиска - место в списке поиска, где удалось найти команду. Опция -r удаляет все запомненные данные. Если не указан ни один аргумент, то выводится информация о запомненных командах: **hits** - количество обращений shell'а к данной команде; **cost** - объем работы для обнаружениякоманды в соответствии со списком поиска; command

- полное имя команды. Звездочкой в колонке hits помечаются коман- ды, маршрут поиска которых будет перевычисляться после смены теку- щего каталога [см. cd(1)]; расходы на перевычисление учитываются в колонке cost.

**newgrp [аргумент ...]**

Выполяет регистрацию пользователя в новой группе [man 1 newgrp].

**pwd** Выводит имя текущего каталога. [см. pwd(1)].

**read [переменная ...]**

Со стандартного ввода читается одна строка и делится на слова с учетом разделителей, перечисленных в значении переменной IFS (обычно это пробел или табуляция); первое слово присваивается первой переменной, второе - второй и т.д., причем все оставшиеся слова присваиваются последней переменной.

Исходная строка имеет продолжение, если в конце ее стоит последо- вательность \перевод\_строки. Символы,отличные от перевода строки, также могут быть экранированы с помощью \, который удаляется пе- ред присваиванием слов. Возвращается нулевой код завершения, если только не встретился конец файла.

**readonly [переменная ...]**

Указанные переменные отмечаются как доступные только на чтение. Присваивания таким переменным трактуются как ошибки. Если аргумен- ты не указаны,то выводится информация обо всех переменных, доступ- ных только на чтение.

**return [код\_завершения]**

Выйти из функции с указанным кодом\_завершения. Если аргумент опу- щен, то код завершения наследуется от последней выполненной ко- манды.

**set [-a] [-e] [-f] [-h] [-k] [-n] [-t] [-u] [-v] [-x] [--] [аргумент …]]**

-a Экспортировать переменные, которые изменяются или создаются, в окружение.

-e Выйти из shell'а, если какая-либо команда воз- вращает ненулевой код завершения.

-f Запретить генерацию имен файлов.

-h Определить и запомнить местоположение всех ко- манд, входящих в тело функции, во время ее оп- ределения, а не во время выполнения.

-k Поместить в окружение команды все переменные, получившие значение в командной строке, а не только те, что предшествуют имени команды.

-n Читать команды, но не выполнять их.

-t Выйти из shell'а после ввода и выполнения одной команды.

-u Рассматривать подстановку параметров, не полу- чивших значений, как ошибку.

-v Выводить исходные для shell'а строки сразу пос- ле их ввода.

-x Выводить команды и их аргументы непосредственно перед выполнением.

-- Не изменяет флаги. Полезно использовать для присваивания позиционному параметру $1 значения

-:

set -- -

При указании + вместо - перечисленные выше режимы выключаются. Описанные флаги могут также использо- ваться при запуске shell'а. Набор текущих флагов

есть значение переменной $-. Следующие за флагами аргументы будут присвоены позиционным параметрам

$1, $2 и т.д. Если не заданы ни флаги, ни аргумен- ты, выводятся значения всех переменных.

**shift [n]**

Позиционные параметры, начиная с (n+1)-го, переименовываются в $1 и т.д. По умолчанию n=1.

**test** Вычислить условное выражение [см. test(1)].

**times** Вывести суммарные времена пользователя и системы, затраченные на выполнение процессов, запущенных данным shell'ом.

**trap [имя\_команды] [n] ...**

Команда с указанным именем будет прочитана и выполнена, когда shell получит сигнал(ы) n. Заметим, что имя\_команды обрабатывается при установке прерывания и при получении сигнала. Команды выполня- ются в порядке номеров сигналов. Нельзя установить обработку пре- рывания по сигналу, игнорируемому данным shell'ом. Попытка устано- вить обработку прерывания по сигналу 11 (выход за допустимые гра- ницы памяти) приводит к ошибке. Если имя\_команды опущено, то для прерываний с указанными номерами n восстанавливается первоначаль- ная реакция. Если имя\_команды есть пустая строка, то этот сигнал будет игнорироваться shell'ом и вызываемыми им программами. Если n равно 0, то указанная команда выполняется при выходе из shell'а. Trap без аргументов выводит список команд, связанных с каждым сиг- налом.

**type [имя ...]**

Для каждого имени указывается, как оно будет интерпретироваться при использовании в качестве имени команды.

**ulimit [размер\_в\_блоках]**

Установить максимальный размер\_в\_блоках (по 1 Кб) тех файлов, в которые пишут данный shell и его потомки (читать можно файлы любого размера) [см. ulimit(1)]. Если размер не указан, выдается текущий лимит. Каждый пользователь может уменьшить собственный лимит, но только суперпользователь может его увеличить.

**umask [nnn]**

Пользовательская маска создания файлов становится равной nnn (восьмеричное) [см. umask(1)]. Если nnn опущено, выдается текущее значение маски.

**unset [имя ...]**

Для каждого указанного имени удалить соответствующую переменную или функцию. Переменные PATH, PS1, PS2, MAILCHECK и IFS не могут быть удалены.

**wait [идентификатор\_процесса]**

Ждать завершения указанного фонового процесса и вывести код его завершения. При отсутствии аргумента ждать завершения всех актив- ных фоновых процессов. В этом случае код завершения будет нулевым [см. wait(1)].

## Фоновый и приоритетный режимы

**В интерактивном** *режиме*, ***shell*** взаимодействует с конкретным *пользователем*

посредством *консоли* (терминала):

* пользователь в консоли набирает (редактирует) цепочку символов и, в конце цепочки нажимает клавишу «***Ввод***»;
* shell проводит синтаксический анализ введенной цепочки, выделяет простые команды, формирует конвейер команд и запускает **задание**;
* когда задание, которое может состоять из множества процессов, завершится, shell выдаст на консоль приглашение на ввод новой цепочки символов.

*Задания*, выполняющиеся указанным способом, называются *заданиями, выполняю- щимися в приоритетном режиме*. Shell блокирует ввод новых цепочек символов до завершения таких заданий.

*Если пользователь*, перед нажатием клавиши **«*Ввод*»** укажет символ ***&***, то задание будет выполняться *в фоновом режиме*. В этом случае:

* shell выводит на консоль *номер задания*, заключенный в квадратные скобки, и

*номер PID* родительского процесса задания;

* после этого, shell выводит на консоль приглашение пользователю для ввода новой цепочки символов.

**Например**,

**$**

**ls -l & «Ввод»**

[1] 534

… - список файлов текущей директории

$

***Для просмотра*** списка запущенных заданий используется команда **jobs**. Например,

**$ mousepad .upk\_theme &**

[1] 547

$ cat \*.c > myprogs & [2] 548

$ jobs

[1] - Запущен mousepad .upk\_theme &

[2] + Выход 1 cat \*.c > myprogs

$

здесь знак ***плюс*** означает выполняемое в данный момент задание, а знак ***минус*** — задание, ожидающее выполнения.

Для перевода фонового задания *в приоритетный режим работы*, используется команда **fg (foreground)**.

Например,

**$ fg %2**

cat \*.c > myprogs

$

## Отмена заданий

*Для отмены заданий*, выполняющихся в фоновом режиме, используется команда **kill**, которая в качестве аргумента может использовать *номер задания* или *PID*. В результате применения этой команды, задание ***прекращает работу***, а созданные им процессы ***уничтожаются***.

Например,

$ jobs

[1] - Запущен

[2] + Выход 1

mousepad .upk\_theme & cat \*.c > myprogs

$ kill %1

$

или, тоже самое:

$ kill 547

$

## Прерывания

Выполнение задания *в приоритетном режиме* можно прервать, используя комби- нацию клавиш **Ctrl-Z**.

При этом:

* *выполнение задания приостанавливается* и ***shell*** выдает пользователю *приг- лашение* на ввод новой цепочки символов;
* командой **fg (*foreground*)** задание можно перевести в *приоритетный режим*;
* командой **bg (*background*)** задание можно перевести в *фоновый режим*.

## Завершение работы ОС

Если запустить ОС может любой пользователь, *который включит питание ЭВМ* и, возможно, *выберет в меню тип загружаемой ОС*, то для выключения компьютера, *пользователь должен иметь права на запуск команд*:

* halt [OPTION] …
* poweoff [OPTION] …
* reboot [OPTION] …
* shutdown [OPTION] … TIME [MASSAGE]

**Замечание** *Работая в графической оболочке*, пользователь для выключения ЭВМ использует соответствующее меню.

В этом случае, команды и сам процесс выключения ОС — *скрыт от пользователя*.