

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ  
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

**Кафедра автоматизированных систем управления (АСУ)**

## **ОПЕРАЦИОННЫЕ СИСТЕМЫ**

Тема 3. Языки управления ОС

### **Учебно-методическое пособие**

для студентов уровня основной образовательной программы: **бакалавриат**  
направление подготовки: **09.03.01 - Информатика и вычислительная техника**  
направление подготовки: **09.03.03 - Прикладная информатика**

Разработчик  
доцент кафедры АСУ

В.Г. Резник

2021

**Резник В.Г.**

Операционные системы. Тема 3. Языки управления ОС. Учебно-методическое пособие. – Томск, ТУСУР, 2021. – 38 с.

Учебно-методическое пособие предназначено для изучения теоретической части и выполнения лабораторной работы №3 по теме «Языки управления ОС» учебной дисциплины «Операционные системы» для студентов кафедры АСУ ТУСУР уровня основной образовательной программы бакалавриат направлений подготовки: «09.03.01 - Информатика и вычислительная техника» и «09.03.03 - Прикладная информатика».

## Оглавление

<b>Введение.....</b>	<b>4</b>
<b>1 Тема 3. Языки управления ОС.....</b>	<b>5</b>
1.1 Языки программирования и командные интерпретаторы.....	5
1.2 Базовый язык shell (sh).....	7
1.3 Среда исполнения программ.....	10
1.3.1 Структура файловой системы ОС Linux.....	10
1.3.2 Набор файлов конфигурации sh.....	10
1.3.3 Переменные среды sh.....	10
1.3.4 Окружение и экспорт переменных.....	12
1.4 Командная строка: опции и аргументы.....	13
1.4.1 Команды и аргументы команды.....	13
1.4.2 Опции sh.....	13
1.5 Переменные sh.....	15
1.6 Специальные символы и имена файлов.....	17
1.7 Стандартный ввод/вывод и переадресация.....	18
1.8 Программные каналы.....	21
1.9 Сценарии.....	23
1.9.1 Управляющие конструкции sh.....	23
1.9.2 Примеры сценариев.....	24
1.9.3 Встроенные команды sh.....	28
1.10 Фоновый и приоритетный режимы.....	32
1.11 Отмена заданий.....	33
1.12 Прерывания.....	33
1.13 Завершение работы ОС.....	34
<b>2 Лабораторная работа №3.....</b>	<b>35</b>
2.1 Среда исполнения программ.....	35
2.2 Переменные, опции и аргументы командной строки.....	36
2.3 Стандартный ввод/вывод и переадресация.....	36
2.4 Программные каналы и сценарии.....	36
2.5 Работа с процессами и заданиями среды.....	37
2.6 Сценарии ПО GRUB.....	37
<b>Список использованных источников.....</b>	<b>38</b>

## Введение

В предыдущих двух темах были изучены основные определения и понятия касающиеся назначения и функций ОС, а также освоены навыки их загрузки, на примере ОС УПК АСУ. Для более профессионального изложения и понимания последующего учебного материала необходим язык, более адекватно отражающий сущность изучаемого предмета.

Известно, что для создания ПО ОС используется язык *C*, а в ряде случаев и язык ассемблера, но такие средства необходимы разработчикам, которые уже достаточно хорошо изучили предмет. Кроме того, ОС уже содержит множество системных утилит и других полезных приложений, участвующих в процессе ее эксплуатации и требующих грамотного использования. Для управления таким ПО языки низкого уровня оказываются неэффективными, поскольку предполагают применение более высокого уровня мышления, ориентированного на объектный подход, скрывающий детали низкоуровневой реализации алгоритмов приложений.

Данная тема посвящена языкам управления ОС, с помощью которых собственно и организуется управление ей как в режиме системы, так и в режиме пользователя, хотя, во многих случаях, это является неочевидным потому, что скрыто за графическим интерфейсом пользователя. Детально, в данном учебном пособии, изучается язык *shell* (sh), который уже стал стандартом для организации управления процессами ОС. Как и в предыдущих темах, учебный материал разбит на два раздела:

- теоретическая часть охватывает как синтаксис языка *shell*, так и примеры его использования;
- лабораторная работа №3 предназначена для закрепления знаний теоретической части учебного материала и формирования практических навыков их использования в среде ОС УПК АСУ.

Хотя возможности языка *shell* являются универсальными, методическое изложение материала опирается на теоретические знания и практический опыт, полученный при изучении двух предыдущих тем. Такой подход позволит повысить качество обучения и обеспечит уже известными практическими задачами данную тему.

В процессе изучения данной темы, студент использует методическое пособие по самостоятельной и индивидуальной работе [1], а также учебники [2-3]. Учебное пособие [4] рассматривается как справочный материал, на основе которого формируются демонстрационные примеры. Хорошим помощником более углублённого изучения предмета является книга Паула Кобаута «*Фундаментальные основы Linux*» [5].

# 1 Тема 3. Языки управления ОС

Многие теоретические знания, представленные в данном разделе уже использовались при изучении материала предыдущих двух тем или известны студентам из учебного материала других дисциплин. Тем не менее, имеется ряд моментов, которые необходимо хорошо усвоить, прежде чем изучать вопросы, изложенные в последующих 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*.

## 1.1 Языки программирования и командные интерпретаторы

Основным языком программирования *ядра ОС* и другого *системного программного обеспечения (ПО)* является *язык С*. Этот язык специально создавался для написания ОС и постепенно вытеснил *языки Ассемблера*, которые, в большей степени чем язык *С*, зависели от архитектуры процессора, способов адресации памяти и других архитектурных особенностей ЭВМ.

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

- *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*, программного обеспечения (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++*, *С#*, *Java* и другие.

Ранее отмечалось, что, являясь базовым ПО ЭВМ, ОС охватывает ту часть программного 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 программирования, 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), *cs*h (C Shell), *ksh* (Korn Shell), *tcl* и другие.

Несмотря на имеющиеся различия, все **командные языки** имеют сходный синтаксис 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, разделённых символами пробела или табуляции и заканчивающаяся символами конца строки.

*Команда* — слово, обозначающее действие:

- *встроенная команда* выполняется 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 интерпретаторы 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 обозначаемое **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 символа на экране (дисплее) и передачу введённой строки интерпретатору **shell**, после нажатия клавиши «**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 псевдографики, что для своего времени было достаточно перспективно.

**Совместное** современное существование различных языков вызвано:

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

**Общая проблематика интерпретаторов** заключается в том, что увеличение функциональных возможностей **shell** влечёт:

- *увеличение размера* интерпретатора и уменьшение скорости его загрузки;
- *повышенный расход* оперативной памяти компьютера.

## 1.2 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 мобильных 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 системой и прикладной программой. Закреплён международным стандартом *ISO/IEC 9945* и может использоваться не только для ОС UNIX.

Определим ряд *метанпонятий*, 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 вызовов *exec\*()*; при этом, *sh* будет использовать среду ОС, в которой работала вызывающая программа;
- *процесс sh* может сам порождать необходимое количество *дочерних процессов*, посредством системного вызова *fork()*, отслеживая их работу и анализируя их *коды завершения*;
- *нулевой целочисленный код завершения* означает *нормальное выполнение команды* дочерним процессом;
- *ненулевой целочисленный код завершения* означает *ошибочное выполнение команды* дочерним процессом и дополнительно интерпретируется, в зависимости от ситуации и режимов работы *sh*.

### Замечание

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

При запуске *sh*, как и любая прикладная программа, *наследует все ресурсы* вызывающего 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.

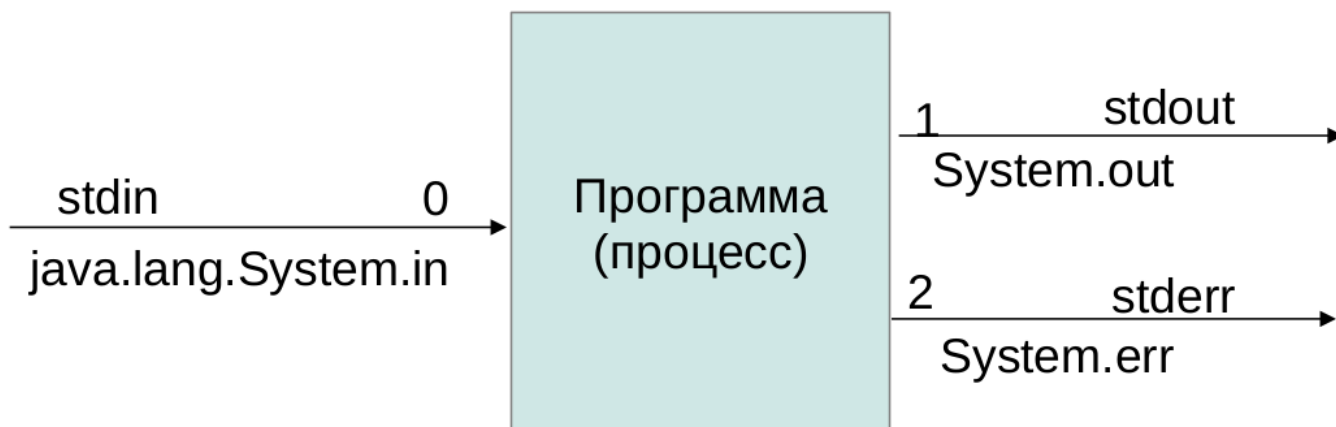


Рисунок 1.1 - Потоки ввода/вывода прикладной программы

**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 C*, 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 - stdin)* используются различные модификации функции **read(...)** языка **C**.

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

### 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, то дескриптор нового файла будет равен **0** и процесс будет читать данные из файла, как будто он читает с устройства клавиатуры.



Все *shell* используют свойства базовых категорий, определённых понятиями: *файл, пользователь и процесс*.

В частности, обычные файлы подразделяются на две категории:

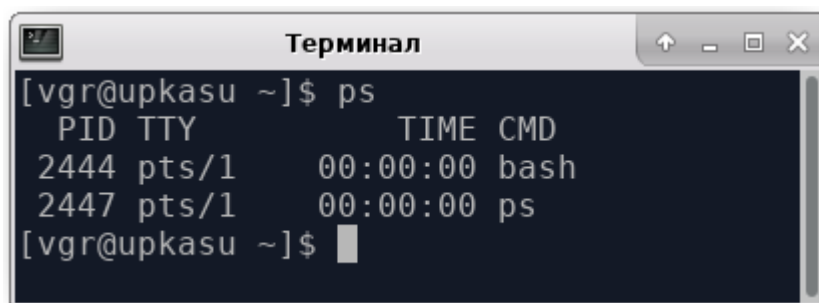
- *бинарные*, которые читаются процессом как последовательность байт, имеющих значения от 0 до 255;
- *текстовые (символьные)*, в которых, в зависимости от кодировки, ряд значений байт или не используются совсем или рассматриваются как управляющие, например: *10 — перевод строки; 13 — возврат каретки*.

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 интерпретатора *shell*.

Чтобы определить на каком терминале работает пользователь, необходимо воспользоваться командой *ps*, без аргументов. На рисунке 1.2 показано применение этой команды в графической среде ОС Linux, которая показывает, что пользователь работает в псевдотерминале */dev/pts/1*.



```
[vgr@upkasu ~]$ ps
  PID TTY          TIME CMD
 2444 pts/1        00:00:00 bash
 2447 pts/1        00:00:00 ps
[vgr@upkasu ~]$
```

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, называются **системными процессами**.

## 1.3 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 программы или дополнительные данные конфигурации среды исполнения;
- *системные переменные среды*, которые наследуются как из среды родительского процесса, а также создаются или удаляются в процессе работы программы.

### 1.3.1 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*;

### 1.3.2 Набор файлов конфигурации sh

В течении работы и запуска интерпретатора *sh* используются следующие конфигурационные 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
```

### 1.3.3 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 и использующего конкретный терминал.

**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 имён 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 каждым выводом приглашения.

## MAILPATH

Список имён файлов, разделённых двоеточием.

Если переменная определена, shell информирует пользователя о поступлении почты в каждый из указанных файлов.

После имени файла может быть указано (вслед за знаком %) сообщение, которое будет выводиться при изменении времени модификации 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 переменной. Если она определена и файловая часть её 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**.

### 1.3.4 Окружение и экспорт переменных

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

**Shell** взаимодействует с окружением несколькими способами:

- *при запуске*, окружение (среда работы) *shell* создаётся утилитой *login*, а затем интерпретатор передаёт 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*, удалённые командой *unset*;
- *плюс все* модифицированные и изменённые 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*.

## 1.4 Командная строка: опции и аргументы

Каждая строка символов, которая заканчивается символом «*Enter*», рассматривается *терминалом* как целостный набор данных, который необходимо передать интерпретатору *shell*.

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

### 1.4.1 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 разделитель — *точка с запятой*;
- в случае, когда команда не помещается в одну строку, для продолжения её на другой строке, используется символ — *обратный слэш*;
- когда *shell* запускается посредством системного вызова *exec\*(...)* и *первым символом нулевого аргумента является -*, то сначала читаются и выполняются команды из файлов */etc/profile* и *\$HOME/.profile*.

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

- *встроенные команды* — команды, которые интерпретатор выполняет самостоятельно;
- *внешние команды* — это программы и утилиты, которые *shell* ищет в файловой 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 подчёркивания*, начинающаяся с буквы или подчёркивания;
- *параметр* - это *имя, цифра* или любой из символов *\*, @, #, ?, -, \$, !*.

### 1.4.2 Опции *sh*

**В общем случае**, интерпретатор *shell* может рассматриваться как команда с аргументами. 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 следующим 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 стандартного ввода. Все оставшиеся аргументы рассматриваются как позиционные параметры.

*Вывод сообщений самого shell*, кроме специальных команд, направляется в файл с дескриптором 2 (стандартный протокол).

*-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 приведёт к завершению работы интерактивного shell'a).

Сигнал прерывания (2) перехватывается и игнорируется, поэтому 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**

## 1.5 Переменные sh

Все переменные **sh**, включая рассмотренные выше *переменные среды*, называются *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: \*, @, #, ?, -, \$, !.

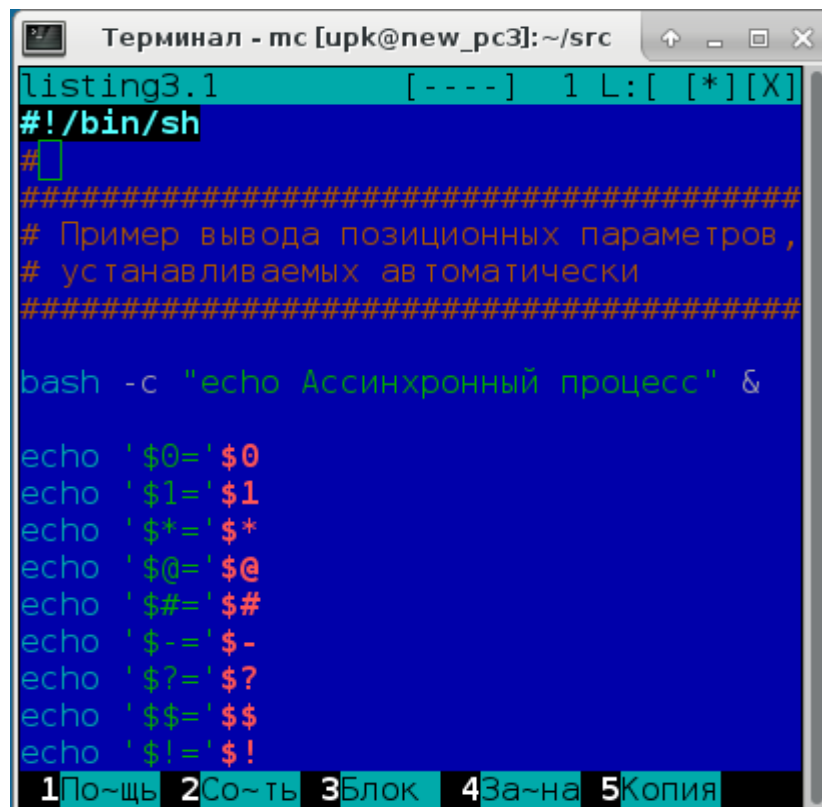
**Значения цифровых позиционных параметров** устанавливаются при вызове *shell-функций* или командой **set**:

- *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*, разделённые пробелами;
- # количество позиционных параметров (десятичное значение);
- - флаги, указанные при запуске *shell* или установленные командой **set**;
- ? содержит десятичное значение, возвращённое предыдущей командой;
- \$ содержит идентификатор процесса, в рамках которого выполняется *shell*;
- ! содержит идентификатор последнего асинхронно запущенного процесса.

**Пример вывода позиционных параметров** показан сценарием на рисунке 1.3.



```
Терминал - mc [upk@new_pc3]: ~/src
listing3.1 [ - - - ] 1 L: [ [*] [X]
#!/bin/sh
#
#####
# Пример вывода позиционных параметров,
# устанавливаемых автоматически
#####

bash -c "echo Ассинхронный процесс" &

echo '$0=' '$0'
echo '$1=' '$1'
echo '$+=' '$+'
echo '$@=' '$@'
echo '$#=' '$#'
echo '$-=' '$-'
echo '$?=' '$?'
echo '$$=' '$$'
echo '$!=' '$!'

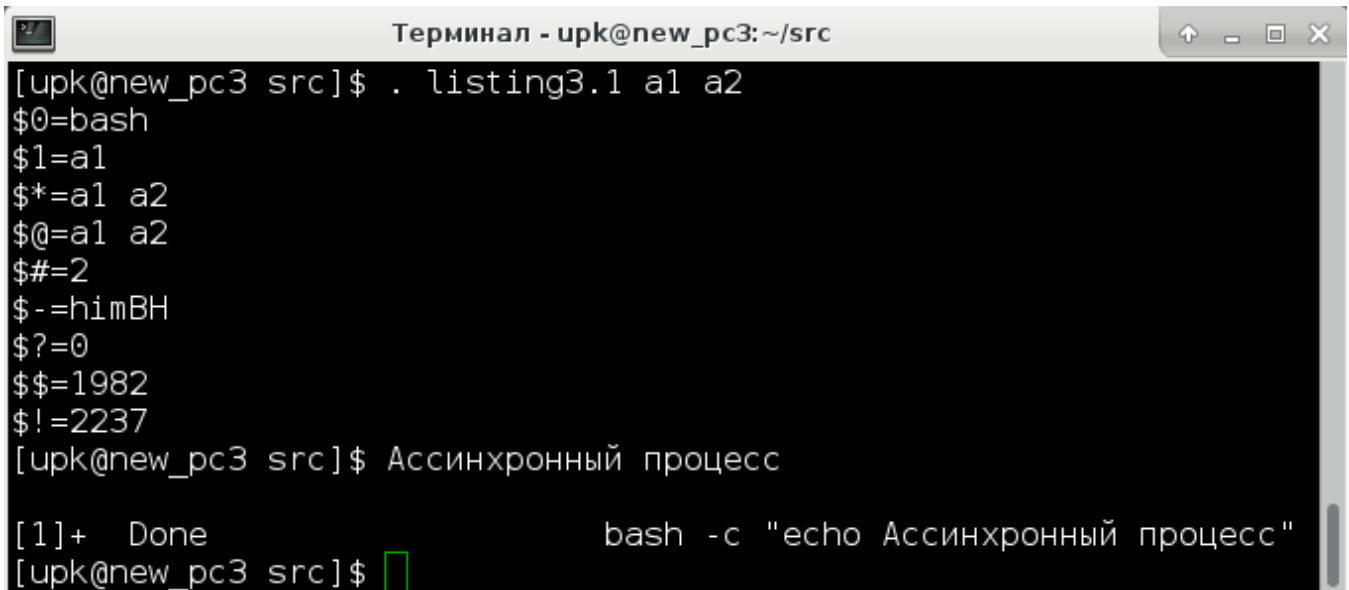
1По-щъ 2Со-ть 3Блок 4За-на 5Копия
```

Рисунок 1.3 — Сценарий вывода позиционных параметров



Результат работы сценария показан рисунком 1.4.

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



```
Терминал - upk@new_pc3: ~/src
[upk@new_pc3 src]$ . listing3.1 a1 a2
$0=bash
$1=a1
$*=a1 a2
$@=a1 a2
$#=2
$-=himBH
$?=0
$$=1982
$!=2237
[upk@new_pc3 src]$ Ассинхронный процесс

[1]+  Done                  bash -c "echo Ассинхронный процесс"
[upk@new_pc3 src]$
```

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 используются, только если за параметром следует буква, цифра или знак подчёркивания, и их нужно отделить от имени параметра. Вместо параметров *\** и *@* подставляются все позиционные параметры, начиная с *\$1*, разделённые пробелами.

### **`${параметр:-слово}`**

Будем говорить, что параметр *пуст*, если его значение **не определено** или является *пустой цепочкой*. При данном способе подстановки если параметр не пуст, подставляется его значение; в противном случае подставляется слово.

### **`${параметр:=слово}`**

Если параметр *пуст*, ему присваивается слово; после этого подставляется значение параметра. Таким способом нельзя изменять значения позиционных параметров.

### **`${параметр:?слово}`**



Если параметр **не пуст**, подставляется его значение; в противном случае в стандартный протокол выдаётся сообщение "параметр:слово" и выполнение shell'a завершается. Если слово опущено, то выдаётся 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 разделителей, 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 слова, 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.

## **1.6 Специальные символы и имена файлов**

В командах, работающих с *именами файлов*, возможно использование **шаблонов**.

**Шаблон** — набор символов, который добавляет или изменяет имена файлов, используемые 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 символов, разделённых знаком **-**, рассматривается как отрезок алфавита. Если за символом **[** стоит знак **!**, то шаблону удовлетворяет любой символ, не перечисленный 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 именами начинающимися с «точки», для текущего каталога (каталог, в котором находится пользователь);
- `ls .x*` - вывод списка имён файлов, начинающихся с *.x*, для текущего каталога;
- `ls .[a-c,x]*` - вывод списка имён файлов, начинающихся с *.a, .b, .c, .x*, для текущего каталога;
- `ls .config` — вывод списка имён каталога *.config*;
- `ls .config/*` — вывод списка имен файлов каталога *.config* и его каталогов.

### Замечание

*Не следует надеяться на интуицию!*

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

## 1.7 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 создаётся; 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 создаётся.

### <<[-]слово

Читается информация со стандартного ввода, пока не встретится строка, совпадающая 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 ещё 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 дескриптор (вместо подразумеваемых дескрипторов 0 или 1), который будет ассоциирован с файлом, указанным в конструкции. Например, строка:

```
... 2>&1
```

ассоциирует дескриптор 2 (стандартный протокол) с файлом, связанным в данный 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, затем - для каждого 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'a, модифицированные спецификациями ввода/вывода.

## Замечание

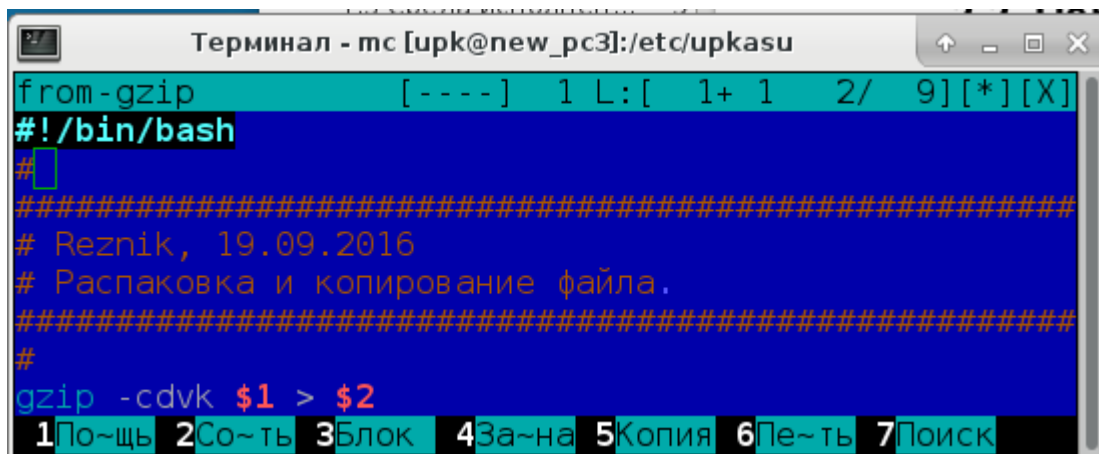
Следует очень внимательно работать с перенаправлениями, поскольку они являются разделителями прав доступа. Например, две команды, подключающие архив студента к рабочей области ОС УПК АСУ, являются разными по правам доступа и не обеспечивают нужный результат:

```
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
```

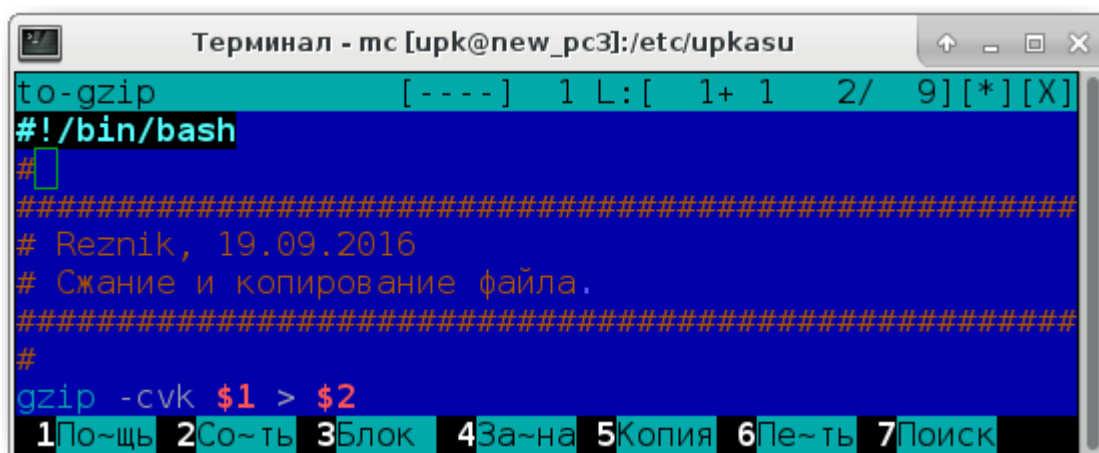
Хотя вторая команда подвергается воздействию команды *sudo*, ее действие распространяется только до оператора перенаправления, что не позволяет записывать результат на защищённые устройства.

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.



```
Терминал - mc [upk@new_pc3]:/etc/upkasu
from-gzip [----] 1 L: [ 1+ 1 2/ 9][*][X]
#!/bin/bash
#
#####
# Reznik, 19.09.2016
# Распаковка и копирование файла.
#####
#
gzip -cdvk $1 > $2
1По~щъ 2Со~ть 3Блок 4За~на 5Копия 6Пе~ть 7Поиск
```

Рисунок 1.5 — Сценарий распаковки архива темы



```
Терминал - mc [upk@new_pc3]:/etc/upkasu
to-gzip [----] 1 L: [ 1+ 1 2/ 9][*][X]
#!/bin/bash
#
#####
# Reznik, 19.09.2016
# Сжатие и копирование файла.
#####
#
gzip -cvk $1 > $2
1По~щъ 2Со~ть 3Блок 4За~на 5Копия 6Пе~ть 7Поиск
```

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

## 1.8 Программные каналы

Рассмотренные выше метаопределения и элементарные понятия языка *shell*, опираются на понятие *простой команды*.

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

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

Общие конструкции языка *shell* используют *специальные файлы ОС*, которые называются *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, разделённых знаком *|* (*вертикальная черта*).

При этом:

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

**Список** — это последовательность *одного или нескольких конвейеров*, разделённых символами *;*, *&*, *&&* или *||* и может заканчиваться символом *;* или *&*.

Из четырёх 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 использовать *символ перевод строки*.

## Замечание

Ранее отмечено, что двойные и одинарные кавычки используются *shell* для *экранирования последовательности слов*, с целью рассмотрения этой последовательности как *отдельный аргумент*.

**Двойные кавычки** разрешают подстановки ключевых и позиционных параметров, одинарные кавычки не разрешают подстановки.

**Дополнительно**, *shell* использует **обратные кавычки**: *shell* читает цепочки символов, заключённые в обратные кавычки, и интерпретирует их **как команды**.

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

Например,

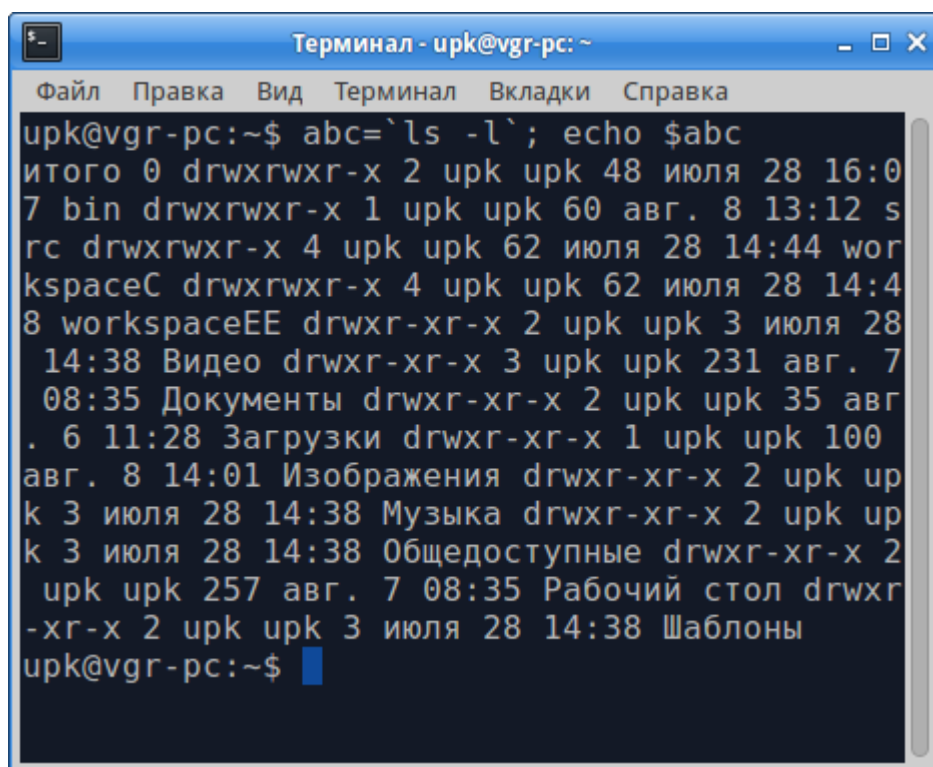
```
abc=`ls -l`;
```

здесь — ключевому параметру **abc** будет присвоен результат выполнения команды **ls -l** — список имён файлов текущей директории. Результат работы данного примера приведён на рисунке 1.7.

## Замечание

Такого же результата можно добиться командой:

```
abc=$(ls -l);
```



```
Терминал - upk@vgr-pc: ~
Файл  Правка  Вид  Терминал  Вкладки  Справка
upk@vgr-pc:~$ abc=`ls -l`; echo $abc
итого 0 drwxrwxr-x 2 upk upk 48 июля 28 16:0
7 bin drwxrwxr-x 1 upk upk 60 авг. 8 13:12 s
rc drwxrwxr-x 4 upk upk 62 июля 28 14:44 wor
kworkspaceC drwxrwxr-x 4 upk upk 62 июля 28 14:4
8 workspaceEE drwxr-xr-x 2 upk upk 3 июля 28
14:38 Видео drwxr-xr-x 3 upk upk 231 авг. 7
08:35 Документы drwxr-xr-x 2 upk upk 35 авг
. 6 11:28 Загрузки drwxr-xr-x 1 upk upk 100
авг. 8 14:01 Изображения drwxr-xr-x 2 upk up
k 3 июля 28 14:38 Музыка drwxr-xr-x 2 upk up
k 3 июля 28 14:38 Общедоступные drwxr-xr-x 2
upk upk 257 авг. 7 08:35 Рабочий стол drwxr
-xr-x 2 upk upk 3 июля 28 14:38 Шаблоны
upk@vgr-pc:~$
```

Рисунок 1.7 — Пример использования обратных кавычек



## 1.9 Сценарии

*Простую команду* или *простой конвейер* можно набрать и выполнить в окне консоли (терминала). *Сложные конструкции* языка *sh* — *программы* — пишутся в файлах, которые называются *сценариями*.

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

### 1.9.1 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 значение 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 используемый для генерации имён 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 выполняется **список\_4** и т.д. Если же коды завершения всех списков, использованных в качестве условий, оказались ненулевыми, выполняется **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, выполняются 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 завершения устанавливается 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
{ }
```

### 1.9.2 Примеры сценариев

**Изучив** понятие сценария, закрепим предыдущий учебный материал тремя примерами, решающими следующие задачи:

- *пример конфигурационного файла*, формируемого во время выполнения загрузки ОС и содержащего параметры передаваемые из GRUB в ядро ОС (см. рисунок 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 приведённые примеры сценариев можно найти в директории **/etc/upkasu**.



```
Терминал
/etc/upkasu/upkasu.conf 1202/1202 100%
#!/bin/sh
#
#####
# Reznik, 08.08.2016
# Файл конфигурации УПК АСУ (/etc/upkasu/upkasu.conf)
#####
#
# Данный файл конфигурации определяет основные параметры,
# используемые специальным ПО ОС УПК АСУ.
#
# Файл содержит описание соответствия между параметрами:
# 1) задаваемыми при старте ОС в меню grub;
# 2) чтения и формирования среды в initramfs-upkasu3.img;
# 3) как источник данных адреса архива студента.

# На рабочей станции разработчика, он используется в прямом виде.
# При старте ОС с flashUSB студента, он формируется динамически.

# Dynamic set...

export UPK_MOUNT=/run/basefs
export ROOT_DIST=0a4cc779-85ff-40db-aec8-26b7af34ecec
export ROOT_ARCH=FC99-4744
export UPK_BOOT=3
export UPK_HOST=new_pc2
export UPK_PATH=asu64upk
export UPK_UEFI=13D8-62DE

1По~щъ 2Ра~рн 3Выход 4Нех 5Пе~ти 6 7Поиск 8Ис~ый 9Фо~ат
```

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$list" != "x" ] || if_exit "${red}Нет тем${gray}, доступных для подключения..."

echo -e "${green}Список доступных тем${gray} обучения..."
echo "-----"
for xx in $list; do
    x1=$(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 подклюена...${gray} \n\
"Вам следует выйти из сессии пользователя${yellow} asu${gray} и подключиться \n\
"как пользователь${yellow} upk ${gray}!"

    if_exit " "
fi

if_exit "Тема $THEME - подключена (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 параметры 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}"

```

### 1.9.3 Встроенные команды *sh*

Кроме управляющих конструкций, *sh* содержит ряд встроенных (*специальных*) команд.

#### 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, то используется значение переменной **HOME**. Переменная **CDPATH** определяет список 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'a**.

**exit [код\_завершения]**

Завершить выполнение **shell'a** с указанным кодом. При отсутствии аргумента код завершения определяется последней выполненной командой. Чтение символа конца файла также приводит к завершению **shell'a**.

**export [переменная ...]**

Заданные переменные отмечаются для автоматического экспорта в окружение выполняемых команд. Если аргументы не указаны, выводится список всех экспортируемых переменных. Имена функций не могут экспортироваться. Имена переменных, экспортированных из родительского **shell'a**, выдаются только в том случае, если они были экспортированы и из текущего **shell'a**.

**getopts**

Используется в **shell**-процедурах для поддержания синтаксических стандартов [см. **intro(1)**]; разбирает позиционные параметры и проверяет 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'a** к данной команде; **cost** - объем работы для обнаружения команды в соответствии со списком поиска; **command** - полное имя команды. Звездочкой в колонке **hits** помечаются команды, маршрут поиска которых будет перевычисляться после смены текущего каталога [см. **cd(1)**]; расходы на перевычисление учитываются в колонке **cost**.

**newgrp [аргумент ...]**

Выполняет 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 учётом разделителей, перечисленных в значении переменной IFS (обычно это пробел или табуляция); первое слово присваивается первой переменной, второе - второй и т.д., причём все оставшиеся слова присваиваются последней переменной.

Исходная строка имеет продолжение, если в конце ее стоит последовательность \перевод\_строки. Символы, отличные от перевода строки, также могут быть экранированы с помощью \, который удаляется перед 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 аргументы не указаны, то выводится информация обо всех переменных, доступных только на чтение.

**return** [код\_завершения]

Выйти из функции с указанным кодом\_завершения. Если аргумент опущен, то код завершения наследуется от последней выполненной команды.

**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'a, если какая-либо команда возвращает ненулевой код завершения.

-f Запретить генерацию имен файлов.

-h Определить и запомнить местоположение всех команд, входящих в тело функции, во время ее определения, 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'a после ввода и выполнения одной команды.

-u Рассматривать подстановку параметров, не получивших значений, как ошибку.

-v Выводить исходные для shell'a строки сразу после 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 использоваться при запуске shell'a. 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 аргументы, 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 выполняются в порядке номеров сигналов. Нельзя установить обработку прерывания по сигналу, игнорируемому данным shell'ом. Попытка установить обработку прерывания по сигналу 11 (выход за допустимые границы памяти) приводит к ошибке. Если имя\_команды опущено, то для прерываний с указанными номерами n восстанавливается первоначальная 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'a. Trap без аргументов выводит список команд, связанных с каждым сигналом.

**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 активных фоновых процессов. В этом случае код завершения будет нулевым [см. wait(1)].

## 1.10 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, выполняющимися 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
$
```

## 1.11 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          mousepad .upk_theme &
[2] + Выход 1          cat *.c > myprogs
$ kill %1
$
```

или, тоже самое:

```
$ kill 547
$
```

## 1.12 Прерывания

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

При этом:

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

## 1.13 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 [MESSAGE]

### Замечание

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

В этом случае, команды и сам процесс выключения ОС является *скрытым от наблюдения пользователем*.

## 2 Лабораторная работа №3

Цель *лабораторной работы №3* — практическое закрепление учебного материала по теме «*Языки управления ОС*».

Метод *достижения указанной цели* — чтение учебного материала, изложенного в первом разделе данного пособия и выполнение указанных в тексте команд в окне терминала.

*Чтобы успешно выполнить данную работу*, студенту следует:

- *запустить ОС УПК АСУ*, подключить личный архив и переключиться в сеанс пользователя *upk*;
- *запустить на чтение* данное пособие и на редактирование личный отчёт;
- *открыть одно или несколько окон терминалов*, причём хотя бы в одном окне терминала открыть *Midnight Commander*, для удобства работы с файловой системой ОС;
- *приступить к выполнению работы*, последовательно пользуясь рекомендациями представленных ниже подразделов.

### Замечание

Многие команды ОС студенту ещё не известны, поэтому следует:

- для вывода на консоль руководства по интересующей команде, использовать: ***man имя\_команды***;
- для выяснения существования команды, ее доступности и местоположения, использовать: ***command -v имя\_команды***;
- для уточнения правил запуска конкретной команды, можно попробовать один из вариантов: ***команда --help*** или ***команда -h*** или ***команда -?***.

### 2.1 Среда исполнения программ

Прочитайте и усвойте материал подразделов 1.1-1.3 данного пособия.

Повторно, в подразделе 1.2, разберитесь с потоками ввода/вывода программы.

В подразделе 1.3:

- выполните в терминале команду ***env***; изучите содержимое вывода и отразите в отчёте;
- выведите содержимое основных переменных среды командой: ***echo \$имя***;
- выполните команду: ***echo \$UPK\_THEME***;
- просмотрите содержимое файла командой: ***cat ./upk\_theme***;
- выполните команду: ***./upk\_theme***;
- выполните команду: ***echo \$UPK\_THEME***;
- выполните команду: ***unset UPK\_THEME***;
- выполните команду: ***echo \$UPK\_THEME***;
- результаты исследования отразите в отчете.

## 2.2 Переменные, опции и аргументы командной строки

Прочитайте и усвойте материал подразделов 1.4-1.6 данного методического пособия.

Повторно прочитайте подраздел 1.5.

Просмотрите и выполните сценарий *listing3.1*, как показано на рисунках 1.3-1.4.

Разберитесь и самостоятельно проверьте работу подстановок в параметрах и отразите результаты в отчёте.

В подразделе 1.6, исследуйте работу шаблонов:

- выполняйте команды: ***ls шаблон***;
- результаты исследования отражайте в отчёте.

## 2.3 Стандартный ввод/вывод и переадресация

Прочитайте и усвойте материал подразделов 1.7 данного методического пособия.

Повторно изучите переадресацию и выполните:

- `echo 'Текст 1' > файл`
- `cat ./файл`
- `echo 'Текст 2' >> файл`
- `cat ./файл`
- `echo 'Текст 2' > файл`
- `cat ./файл`

- результаты исследования отразите в отчёте.

## 2.4 Программные каналы и сценарии

Прочитайте и усвойте материал подразделов 1.8-1.9 данного методического пособия.

Перечитайте подраздел 1.8 и выполните демонстрационные примеры.

Перечитайте часть подраздела 1.9, касающуюся управляющих операторов, и выполните демонстрационные примеры, показанные на рисунке 1.8 и листингах 1.1, 1.2.

Результаты опишите в отчёте.

Перечитайте оставшуюся часть подраздела 1.9 и выполните интересные вам команды.

Результаты опишите в отчёте.

## 2.5 Работа с процессами и заданиями среды

Прочитайте и усвойте материал подразделов 1.10-1.13 данного методического пособия.

Повторно перечитайте и выполните исследовательские действия по учебному материалу подразделов 1.10-1.12.

Отразите результаты исследований в отчёте.

Проведите создание архива и запись его на личный flashUSB.

Попробуйте командами, описанными в подразделе 1.13, перезапустить и выключить компьютер.

Отразите результаты исследований в отчёте.

## 2.6 Сценарии ПО GRUB

Запустите на просмотр файл [upk\\_asu.pdf](#).

Перечитайте его подразделы 2.5 и 2.6.

Опишите в отчёте основные различия языка shell и языка GRUB.

Проведите создание архива и запись его на личный flashUSB.

Выключите компьютер и завершите лабораторную работу.

## **Список использованных источников**

- 1 Резник В.Г. Операционные системы. Самостоятельная и индивидуальная работа студента по направлению подготовки бакалавра 09.03.03. Учебно-методическое пособие. – Томск, ТУСУР, 2016. – 13 с.
- 2 Гордеев А.В. Операционные системы: учебное пособие для вузов. — СПб.: Питер, 2004. — 415с.
- 3 Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.: ил. — (Серия «Классика computer science»). ISBN 978-5-496-01395-6
- 4 Резник В.Г. Учебный программный комплекс кафедры АСУ на базе ОС ArchLinux. Учебно-методическое пособие. – Томск, ТУСУР, 2017. – 38 с.
- 5 Поль Коббо. Фундаментальные основы Linux, 2014/Перевод А. Панина. - [Электронный ресурс]. - Режим доступа: - Фундаментальные основы Linux\_\_by Paul Cobbaut .pdf.