

# Содержание

Об авторах	15
<b>Введение</b>	17
Структура книги	18
От издательства	20
<b>Глава 1. Краткий обзор основ</b>	21
Некоторые основные команды	21
Отображение даты и времени: команда <code>date</code>	21
Выявление пользователей, зарегистрированных в системе: команда <code>who</code>	21
Эхоотображение символов: команда <code>echo</code>	22
Манипулирование файлами	22
Перечисление файлов: команда <code>ls</code>	23
Отображение содержимого файла: команда <code>cat</code>	23
Подсчет количества слов в файле: команда <code>wc</code>	23
Дополнительные параметры команд	24
Создание копии файла: команда <code>cp</code>	24
Переименование файла: команда <code>mv</code>	25
Удаление файла: команда <code>rm</code>	25
Манипулирование каталогами	26
Начальный каталог и пути к файлам	27
Отображение рабочего каталога: команда <code>pwd</code>	28
Смена каталогов: команда <code>cd</code>	28
Дополнительные сведения о команде <code>ls</code>	31
Создание каталога: команда <code>mkdir</code>	33
Копирование файла из одного каталога в другой	34
Перемещение файлов из одного каталога в другой	35
Связывание файлов: команда <code>ln</code>	36
Удаление каталога: команда <code>rmdir</code>	40
Подстановка имен файлов	40
Знак звездочки	41
Совпадение с одиночными символами	42
Особенности указания имен файлов	44
Пробелы в именах файлов	44
Другие необычные символы	45

Стандартный ввод-вывод и его переадресация	46
Стандартный ввод и вывод	46
Переадресация вывода	48
Переадресация ввода	50
Каналы	51
Фильтры	53
Стандартный вывод ошибок	53
Дополнительные сведения о командах	54
Ввод нескольких команд в одной строке	55
Передача команд на выполнение в фоновый режим	55
Команда <code>rs</code>	56
Сводка команд	57
<b>Глава 2. Назначение оболочки</b>	<b>59</b>
Ядро и утилиты	59
Исходная оболочка	60
Ввод команд в оболочке	63
Обязанности оболочки	65
Выполнение программ	65
Подстановка значений переменных и имен файлов	67
Переадресация ввода-вывода	68
Подключение конвейера	70
Контроль окружения	71
Интерпретируемый язык программирования	71
<b>Глава 3. Рабочие инструменты</b>	<b>73</b>
Регулярные выражения	73
Совпадение с любым одиночным символом: знак точки ( <code>.</code> )	73
Совпадение с началом строки: знак вставки ( <code>^</code> )	75
Совпадение с концом строки: знак денежной единицы ( <code>\$</code> )	76
Совпадение с набором символов: конструкция [ <code>...</code> ]	77
Совпадение с нулевым или большим количеством символов: знак звездочки ( <code>*</code> )	79
Совпадение с точным количеством подшаблонов: конструкция <code>\{...\}</code>	82
Сохранение совпавших символов: конструкция <code>\(...\)</code>	85
Команда <code>cut</code>	89
Параметры <code>-d</code> и <code>-f</code>	91
Команда <code>paste</code>	94
Параметр <code>-d</code>	95
Параметр <code>-s</code>	95
Команда <code>sed</code>	96

Параметр -n	98
Удаление строк	99
Команда tr	100
Параметр -s	103
Параметр -d	104
Команда grep	105
Регулярные выражения и команда grep	108
Параметр -v	109
Параметр -l	110
Параметр -n	111
Команда sort	111
Параметр -u	112
Параметр -r	112
Параметр -o	113
Параметр -n	114
Пропуск полей	114
Параметр -t	115
Другие параметры	116
Команда uniq	116
Параметр -d	117
Другие параметры	119
<b>Глава 4. Итак, приступим!</b>	121
Командные файлы	121
Комментарии	124
Переменные	125
Отображение значений переменных	126
Неопределенные переменные имеют пустое значение	128
Подстановка имен файлов и переменных	130
Конструкция \$ { <i>переменная</i> }	131
Встроенные целочисленные арифметические операции	132
<b>Глава 5. Заключение в кавычки</b>	135
Одиночная кавычка	135
Двойные кавычки	139
Обратная косая черта	142
Продолжение строк с помощью знака обратной косой черты	143
Употребление обратной косой черты в двойных кавычках	144
Подстановка команд	145
Обратные кавычки	145
Конструкция \$ ( . . . )	146
Команда expr	151

<b>Глава 6. Передача аргументов</b>	155
Переменная \$#	156
Переменная \$*	157
Программа для поиска абонентов в телефонном справочнике	158
Программа для ввода абонентов в телефонный справочник	160
Программа для удаления абонентов из телефонного справочника	161
Конструкция \${n}	163
Команда shift	163
<b>Глава 7. Выбор по условию</b>	165
Код завершения	165
Переменная \$?	166
Команда test	170
Строковые операции	170
Альтернативная форма команды test	175
Целочисленные операции сравнения	176
Файловые операции	177
Операция логического отрицания !	179
Операция -a логическое И	179
Круглые скобки	180
Операция -o логическое ИЛИ	180
Конструкция else	181
Команда exit	184
Повторное рассмотрение программы rem	184
Конструкция elif	185
Еще одна версия программы rem	188
Команда case	190
Специальные символы совпадения с шаблоном	192
Параметр -x для отладки программ	194
Возвращение к команде case	197
Пустая команда :	198
Конструкции && и	199
<b>Глава 8. Повторное выполнение команд</b>	203
Команда for	203
Переменная \$#	206
Цикл for без списка	208
Команда while	208
Команда until	210
Дополнительные сведения о циклах	216
Прерывание цикла	216
Пропуск или оставление команд в цикле	217

Выполнение цикла в фоновом режиме	218
Конвейеризация ввода из вывода данных из цикла	220
Ввод цикла в одной строке	220
Команда <code>getopts</code>	221
<b>Глава 9. Ввод и вывод данных</b>	<b>227</b>
Команда <code>read</code>	227
Пример программы копирования файлов	227
Употребление специальных управляющих символов в команде <code>echo</code>	229
Усовершенствованная версия программы <code>mysp</code>	230
Окончательная версия программы <code>mysp</code>	232
Управляемая через меню программа ведения телефонного справочника	236
Переменная <code>\$\$</code> и временные файлы	241
Код завершения, возвращаемый командой <code>read</code>	243
Команда <code>printf</code>	245
<b>Глава 10. Рабочая среда</b>	<b>253</b>
Локальные переменные	253
Подоболочки	254
Экспортируемые переменные	256
Команда <code>export -p</code>	260
Переменные <code>PS1</code> и <code>PS2</code>	260
Переменная <code>HOME</code>	261
Переменная <code>PATH</code>	262
Текущий каталог пользователя	270
Переменная <code>CDPATH</code>	271
Дополнительные сведения об подоболочках	272
Команда <code>.</code>	272
Команда <code>exec</code>	276
Конструкции <code>(...)</code> и <code>{ ...; }</code>	277
Другой способ передачи переменных подоболочке	280
Файл <code>.profile</code>	281
Переменная <code>TERM</code>	282
Переменная <code>TZ</code>	283
<b>Глава 11. Дополнительные сведения о параметрах</b>	<b>285</b>
Подстановка значений параметров	285
Конструкция <code>\${ параметр }</code>	285
Конструкция <code>\${ параметр:-значение }</code>	286
Конструкция <code>\${ параметр:=значение }</code>	287
Конструкция <code>\${ параметр:?значение }</code>	288
Конструкция <code>\${ параметр:+значение }</code>	288

Конструкции для сопоставления с шаблоном	289
Конструкция <code>\$(#переменная)</code>	291
Переменная <code>\$0</code>	291
Команда <code>set</code>	292
Параметр <code>-x</code>	293
Команда <code>set</code> без аргументов	294
Переназначение позиционных параметров с помощью команды <code>set</code>	294
Параметр <code>--</code>	295
Другие параметры команды <code>set</code>	298
Переменная <code>IFS</code>	298
Команда <code>readonly</code>	301
Команда <code>unset</code>	302
<b>Глава 12. Невыясненные вопросы</b>	<b>303</b>
Команда <code>eval</code>	303
Команда <code>wait</code>	305
Переменная <code>\$!</code>	306
Команда <code>trap</code>	306
Команда <code>trap</code> без аргументов	308
Игнорирование сигналов	309
Сброс прерываний	310
Дополнительные сведения об организации ввода-вывода	310
Конструкции <code>&lt;&amp;-</code> и <code>&gt;&amp;-</code>	312
Встраиваемая переадресация ввода	312
Архивные файлы оболочек	314
Функции	318
Удаление определения функции	321
Команда <code>return</code>	321
Команда <code>type</code>	321
<b>Глава 13. Возращение к программе <code>rol0</code></b>	<b>323</b>
Вопросы форматирования данных	323
Программа <code>rol0</code>	324
Программа <code>add</code>	327
Программа <code>lu</code>	328
Программа <code>display</code>	329
Программа <code>rem</code>	330
Программа <code>change</code>	332
Программа <code>listall</code>	334
Образец выводимого результата	335

<b>Глава 14. Интерактивные и нестандартные средства оболочки</b>	339
Выбор подходящей оболочки	339
Файл ENV	340
Редактирование в режиме командной строки	342
Предыстория команд	342
Режим правки строк в редакторе vi	343
Доступ к командам из предыстории	345
Режим правки строк в редакторе emacs	347
Доступ к командам из предыстории	349
Другие способы доступа к предыстории команд	352
Команда history	352
Команда fc	353
Команда r	353
Функции	355
Локальные переменные	355
Автоматически загружаемые функции	355
Целочисленные арифметические операции	355
Целочисленные типы данных	357
Числа с разным основанием системы счисления	357
Команда alias	359
Удаление псевдонимов	362
Массивы	362
Управление заданиями	368
Остановленные задания и команды fg и bg	369
Разные средства	371
Другие возможности команды cd	371
Замена знака тильды	372
Порядок поиска	373
Краткий итог совместимости оболочек	374
<b>Приложение А. Краткое изложение оболочки</b>	377
Запуск	377
Команды	377
Комментарии	378
Переменные оболочки	378
Позиционные параметры	378
Специальные параметры	378
Подстановка параметров	380
Повторный ввод команд	382
Команда fc	382
Режим редактирования строк в редакторе vi	382

Заключение в кавычки	385
Замена знака тильды	386
Арифметические выражения	386
Подстановка имен файлов	387
Переадресация ввода-вывода	388
Экспортируемые переменные и выполнение подоболочек	389
Конструкция (...)	389
Конструкция { ...; }	389
Дополнительные сведения о переменных оболочки	389
Функции	390
Управление заданиями	390
Задания в оболочке	390
Остановка заданий	391
Сводка команд	391
Команда :	391
Команда .	392
Команда alias	392
Команда bg	393
Команда break	393
Команда case	393
Команда cd	394
Команда continue	395
Команда echo	395
Команда eval	396
Команда exec	397
Команда exit	397
Команда export	397
Команда false	398
Команда fc	398
Команда fg	399
Команда for	399
Команда getopts	400
Команда hash	402
Команда if	402
Команда jobs	405
Команда kill	405
Команда newgrp	406
Команда pwd	406
Команда read	406
Команда readonly	407
Команда return	408



**14** Содержание

Команда <code>set</code>	408
Команда <code>shift</code>	410
Команда <code>test</code>	411
Команда <code>times</code>	413
Команда <code>trap</code>	413
Команда <code>true</code>	415
Команда <code>type</code>	415
Команда <code>umask</code>	415
Команда <code>unalias</code>	416
Команда <code>unset</code>	416
Команда <code>until</code>	416
Команда <code>wait</code>	417
Команда <code>while</code>	417
<b>Приложение Б. Дополнительные источники информации</b>	<b>419</b>
Оперативно доступная документация	419
Документация, доступная в Интернете	420
Литература	421
Издательство O'Reilly & Associates	421
Издательство Pearson	421
<b>Предметный указатель</b>	<b>423</b>

# Передача аргументов

Пользы от программ оболочки станет намного больше, если научиться обрабатывать передаваемые им аргументы. Из материала этой главы вы узнаете, как писать программы, принимающие аргументы в командной строке. Напомним, что в главе 4 был рассмотрен пример однострочной программы `run`, предназначенной для обработки файла `sys.caps` по командам `tbl`, `nroff` и `lp`:

```
$ cat run
tbl sys.caps | nroff -mm -Tlp | lp
$
```

Допустим, с помощью той же самой последовательности команд требуется обработать не только файл `sys.caps`, но и другие файлы. Для обработки каждого такого файла можно было бы создать отдельную версию программы `run` или видоизменить ее таким образом, чтобы иметь возможность указывать имя обрабатываемого файла в командной строке. Такое видоизменение программы `run` означало бы возможность набрать ее в командной строке следующим образом:

```
run new.hire
```

чтобы указать имя файла `new.hire` для обработки с помощью данной последовательности команд, или же таким образом:

```
run sys.caps
```

чтобы указать имя обрабатываемого файла `sys.caps`.

Всякий раз, когда выполняется программа оболочки, первый ее аргумент сохраняется в специальной переменной оболочки `1`, второй аргумент — в переменной оболочки `2` и т.д. (Для большего удобства здесь и далее эти переменные будут обозначаться как `$1`, `$2` и т.д., несмотря на то, что знак `$` фактически является частью обозначения ссылки на переменную, а не ее имени.) Эти специальные переменные формально называются *позиционными параметрами*, поскольку они опираются на позиции значений, указываемых в командной строке и присваиваемых после обычной процедуры обработки содержимого командной строки в оболочке, т.е. переадресации ввода-вывода, подстановки переменных и имен файлов и т.д.

Чтобы программа `run` принимала имя файла в качестве аргумента, достаточно заменить в ней ссылку на файл `sys.caps` ссылкой на первый аргумент, набранный в командной строке:

```
$ cat run
tbl $1 | nroff -mm -Tlp | lp
$
$ run new.hire          Выполнить с именем файла new.hire
                        в качестве аргумента
request id is laser1-24 (standard input)
$
```

Всякий раз, когда выполняется программа `run`, все, что набрано после ее имени в командной строке, сначала сохраняется оболочкой в первом позиционном параметре, а затем передается самой программе. Так, в приведенном выше примере имя файла `new.hire` сохраняется в этом параметре.

Подстановка позиционных параметров осуществляется таким же образом, как и подстановка других типов переменных. Следовательно, когда оболочка обнаруживает команду

```
tbl $1
```

она заменяет ссылку `$1` первым аргументом, которым снабжается программа `run` (в данном случае это имя файла `new.hire`).

В качестве еще одного примера ниже приведена программа `ison`, которая позволяет выяснить, зарегистрирован ли указанный пользователь в системе.

```
$ cat ison
who | grep $1
$ who          Показать, кто зарегистрирован в системе
root      console Jul 7 08:37
barney    tty03    Jul 8 12:28
fred      tty04    Jul 8 13:40
joanne    tty07    Jul 8 09:35
tony      tty19    Jul 8 08:30
lulu      tty23    Jul 8 09:55
$ ison tony
tony      tty19    Jul 8 08:30
$ ison pat
$          Этот пользователь не зарегистрирован в системе
```

## Переменная \$#

Помимо позиционных параметров, в оболочке имеется специальная переменная `$#`, принимающая ряд аргументов, набранных в командной строке. Как будет показано в следующей главе, содержимое этой переменной нередко проверяется в программах на правильность количества аргументов, указанных пользователем.

В следующем примере демонстрируется программа `args`, написанная с единственной целью — показать, каким образом аргументы передаются программе оболочки. Внимательно проанализируйте каждый результат ее выполнения, чтобы хорошо понимать, как она действует.

```

$ cat args                Показать программу
echo $# arguments passed
echo arg 1 = :$1: arg 2 = :$2: arg 3 = :$3:
$ args a b c              Выполнить программу
3 arguments passed
arg 1 = :a: arg 2 = :b: arg 3 = :c:
$ args a b                Опробовать программу с другими аргументами
2 arguments passed
arg 1 = :a: arg 2 = :b: arg 3 = ::                Неприсвоенные аргументы
                                                    оказываются пустыми
$ args                    Попробовать выполнить программу
                                                    без аргументов
0 arguments passed
arg 1 =:: arg 2 =:: arg 3 = ::
$ args "a b c"           Попробовать указать аргументы в кавычках
1 arguments passed
arg 1 = :a b c: arg 2 = :: arg 3 = ::
$ ls x*                  Выяснить имена файлов, начинающихся
                                                    на букву x
xact
xtra
$ args x*                Попробовать подставить имя файла
2 arguments passed
arg 1 = :xact: arg 2 = :xtra: arg 3 = ::
$ my_bin=/users/steve/bin
$ args $my_bin           И подставить переменную
1 arguments passed
arg 1 = :/users/steve/bin: arg 2 = :: arg 3 = ::
$ args $(cat names)     Передать содержимое файла names
7 arguments passed
arg 1 = :Charlie: arg 2 = :Emanuel: arg 3 = :Fred:
$

```

Как видите, оболочка обрабатывает содержимое командной строки обычным образом, когда она выполняет написанные для нее программы. Это означает, что, указывая аргументы в своих программах оболочки, можно выгодно воспользоваться такими привычными удобствами, как подстановка имен файлов и переменных.

## Переменная \$\*

Специальная переменная \$\* служит для ссылки на *все* аргументы, передаваемые программе. Это нередко оказывается удобным в тех программах, которые принимают неопределенное или *переменное* количество аргументов. Далее будут представлены более практические примеры, а в следующем примере программы демонстрируется применение переменной \$\*:

```

$ cat args2
echo $# arguments passed
echo they are :$:
$ args2 a b c

```

```

3 arguments passed
they are :a b c:
$ args2 one                two
2 arguments passed
they are :one two:
$ args2
0 arguments passed
they are ::
$ args2 *
8 arguments passed
they are :args args2 names nu phonebook stat xact xtra:
$

```

## Программа для поиска абонентов в телефонном справочнике

Ниже приведено содержимое файла `phonebook` из предыдущих примеров.

```

$ cat phonebook
Alice Chebba      973-555-2015
Barbara Swingle  201-555-9257
Liz Stachiw      212-555-2298
Susan Goldberg   201-555-7776
Susan Topple     212-555-4932
Tony Iannino     973-555-1295
$

```

Как известно, найти абонента в этом файле нетрудно по команде `grep`:

```

$ grep Cheb phonebook
Alice Chebba      973-555-2015
$

```

Известно также, что если требуется найти абонента по имени и фамилии, их следует указать в кавычках как единый аргумент:

```

$ grep "Susan T" phonebook
Susan Topple     212-555-4932
$

```

Но было бы неплохо написать отдельную программу оболочки для поиска абонентов в телефонном справочнике. Такая программа может называться `lu` и должна принимать в качестве своего аргумента имя искомого абонента:

```

$ cat lu
#
# Найти абонента в телефонном справочнике
#

grep $1 phonebook
$

```

Ниже приведены конкретные примеры применения программы `lu`.

```
$ lu Alice
Alice Chebba      973-555-2015
$ lu Susan
Susan Goldberg    201-555-7776
Susan Topple      212-555-4932
$ lu "Susan T"
grep: can't open T
phonebook:Susan Goldberg    201-555-7776
phonebook:Susan Topple      212-555-4932
$
```

В последнем из приведенных выше примеров имя и фамилия `Susan T` были аккуратно заключены в кавычки. В чем же тогда дело? Чтобы выяснить недоразумение, рассмотрим еще раз следующий вызов команды `grep` из программы `lu`:

```
grep $1 phonebook
```

Как видите, заключение в кавычки имени и фамилии `Susan T` приводит к тому, что они передаются программе `lu` как единый аргумент `$1`. Но когда оболочка подставляет его значение из командной строки в самой программе, то команде `grep` оно фактически передается как *два* аргумента. Этот недостаток можно устранить, заключив в двойные кавычки аргумент `$1` непосредственно в программе `lu`:

```
$ cat lu
#
# Найти абонента в телефонном справочнике - версия 2
#
grep "$1" phonebook
$
```

Одиночные кавычки в данном случае не подойдут. А теперь попробуем снова выполнить программу `lu`, как показано ниже.

```
$ lu Tony
Tony Iannino 973-555-1295
$ lu "Susan T"
Susan Topple      212-555-4932
$
```

*Действует по-прежнему  
А теперь попробуем ввести  
тот же самый аргумент снова*

## Программа для ввода абонентов в телефонный справочник

Продолжим разработку программ для обработки файла `phonebook`. В какой-то момент может возникнуть потребность ввести нового абонента в этот файл телефонного справочника, особенно если он невелик. С этой целью напишем приведенную ниже программу `add`, принимающую два аргумента: имя абонента, вводимого в телефонный справочник, и номер его телефона.

```
$ cat add
#
# Ввести нового абонента в телефонный справочник
#

echo "$1          $2" >> phonebook
$
```

И хотя этого не видно, на самом деле аргументы `$1` и `$2` разделяются в приведенной выше команде `echo` знаком табуляции. Этот знак должен быть указан в кавычках, чтобы воспроизводиться по команде `echo`, а не “поглощаться” оболочкой. Опробуем новую программу следующим образом:

```
$ add 'Stromboli Pizza' 973-555-9478
$ lu Pizza          Проверить, удастся ли найти новую запись
Stromboli Pizza 973-555-9478          Пока что все нормально
$ cat phonebook      Вывести содержимое телефонного справочника
Alice Chebba        973-555-2015
Barbara Swingle     201-555-9257
Liz Stachiw         212-555-2298
Susan Goldberg      201-555-7776
Susan Topple        212-555-4932
Tony Iannino        973-555-1295
Stromboli Pizza    973-555-9478
$
```

Имя и фамилия `Stromboli Pizza` были заключены в одиночные кавычки, чтобы оболочка передала их программе `add` как единый аргумент. (А что бы произошло, если бы имя и фамилия не были заключены в одиночные кавычки?) После программы `add` в приведенном выше примере была выполнена программа `lu`, чтобы выяснить, удастся ли найти новую запись в телефонном справочника, что и было благополучно сделано. А затем была выполнена команда `cat`, чтобы посмотреть, каким образом теперь выглядит видоизмененный телефонный справочник. Как и предполагалось, в самом его конце появилась новая запись.

К сожалению, новый файл стал неотсортированным. И хотя сортировка содержимого этого файла не оказывает никакого влияния на выполнение программы `lu`, тем не менее, она является полезным свойством. С этой целью в программу `add` можно ввести команду сортировки `sort`, как показано ниже.

```
$ cat add
#
# Ввести нового абонента в телефонный справочник - версия 2
#

echo "$1          $2" >> phonebook
sort -o phonebook phonebook
$
```

Напомним, что параметр `-o` команды `sort` определяет место для вывода отсортированного результата. В данном случае это тот же самый входной файл, как показано ниже. Всякий раз, когда в файл `phonebook` вводится новая запись, его содержимое сортируется снова, чтобы многострочные совпадения с критерием поиска всегда располагались в алфавитном порядке.

```
$ add 'Billy Bach' 201-555-7618
$ cat phonebook
Alice Chebba          973-555-2015
Barbara Swingle      201-555-9257
Billy Bach            201-555-7618
Liz Stachiw           212-555-2298
Stromboli Pizza      973-555-9478
Susan Goldberg        201-555-7776
Susan Topple          212-555-4932
Tony Iannino          973-555-1295
$
```

## Программа для удаления абонентов из телефонного справочника

Ни один комплект программ, позволяющих искать или вводить абонентов в телефонный справочник, не будет полным без программы удаления абонентов из телефонного справочника. Поэтому напишем программу `rem`, позволяющую удалить из телефонного справочника абонента по имени, указанному в качестве аргумента в командной строке.

Какой должна быть стратегия разработки такой программы? По существу, из файла требуется удалить строку, содержащую указанное имя абонента, что противоположно совпадению с шаблоном. В данном случае можно воспользоваться параметром `-v` команды `grep`, поскольку он позволяет сделать именно то, что требуется: вывести из файла все строки, которые *не* совпадают с шаблоном:

```
$ cat rem
#
# Удалить абонента из телефонного справочника
#

grep -v "$1" phonebook > /tmp/phonebook
mv /tmp/phonebook phonebook
$
```



По команде `grep` с параметром `-v` все строки из файла `phonebook`, не совпадающие с заданным шаблоном, выводятся в файл `/tmp/phonebook`. (*Примечание:* каталог `/tmp` служит в системах Unix для хранения временных файлов и обычно удаляется при перезапуске системы.) А после выполнения команды `grep` прежний файл `phonebook` заменяется новым файлом из каталога `/tmp`. Ниже приведены некоторые примеры применения вновь созданной программы `rm`.

```
$ rm 'Stromboli Pizza'           Удалить эту запись
$ cat phonebook
Alice Chebba      973-555-2015
Barbara Swingle  201-555-9257
Billy Bach        201-555-7618
Liz Stachiw       212-555-2298
Susan Goldberg    201-555-7776
Susan Topple      212-555-4932
Tony Iannino      973-555-1295
$ rm Susan
$ cat phonebook
Alice Chebba      973-555-2015
Barbara Swingle  201-555-9257
Billy Bach        201-555-7618
Liz Stachiw       212-555-2298
Tony Iannino      973-555-1295
$
```

В первом случае из файла `phonebook` была благополучно удалена запись `Stromboli Pizza`. Но во втором случае из него были удалены обе записи с именем `Susan`, поскольку обе они совпали с заданным шаблоном, а это не совсем верно! Чтобы ввести эти записи обратно в файл `phonebook`, можно воспользоваться программой `add` следующим образом:

```
$ add 'Susan Goldberg' 201-555-7776
$ add 'Susan Topple' 212-555-4932
$
```

В главе 7 будет показано, как проверять действие прежде, чем выполнять его, чтобы в рассматриваемом здесь примере программы можно было определить факт обнаружения не одной, а нескольких совпадающих записей. Но в программе может, например, возникнуть потребность известить пользователя об обнаружении нескольких совпавших записей вместо слепого их удаления. (Это было бы очень удобно, поскольку в большинстве реализаций команды `grep` произойдет совпадение со всем, что угодно, если в качестве шаблона передать ей пустую символьную строку. По существу, это привело бы к удалению всего содержимого файла `phonebook`, что было бы неверно!)

Между прочим, совпавшую запись можно было бы удалить и по команде `sed`. В таком случае команду `grep` можно было бы заменить следующей командой:

```
sed "/$1/d" phonebook > /tmp/phonebook
```

чтобы добиться аналогичного результата. Аргумент команды `sed` необходимо заключить в двойные кавычки, чтобы обеспечить подстановку значения аргумента `$1` и в то же время гарантировать, что оболочка не отреагирует на команду, аналогичную приведенной ниже, где команде `sed` передаются три, а не два аргумента.

```
sed /Stromboli Pizza/d phonebook > /tmp/phonebook
```

## Конструкция `${n}`

Если предоставить программе больше девяти аргументов, то десятый и последующие аргументы (`$10`, `$11` и т.д.) окажутся недоступными. И если попытаться получить доступ к десятому аргументу по ссылке `$10`, то оболочка фактически подставит значение аргумента `$1`, а затем значение `0`. Вместо этого следует воспользоваться конструкцией `${n}`. Так, чтобы получить непосредственный доступ к десятому аргументу, следует указать ссылку `${10}` в своей программе и так далее для одиннадцатого, двенадцатого и остальных аргументов.

## Команда `shift`

Эта команда позволяет, по существу, *сместить влево* позиционные параметры. Если выполнить команду `shift`, предыдущее значение позиционного параметра `$2` будет присвоено позиционному параметру `$1`, а предыдущее значение позиционного параметра `$3` — позиционному параметру `$2` и т.д. В то же время прежнее значение позиционного параметра `$1` будет безвозвратно утрачено.

При выполнении этой команды значение переменной `$#`, содержащей количество аргументов, также автоматически уменьшается на единицу, как показано ниже.

```
$ cat tshift
```

*Программа для проверки смещения  
позиционных параметров*

```
echo $# $*
shift
echo $# $*
shift
echo $# $*
shift
echo $# $*
shift
echo $# $*
shift
echo $# $*
$ tshift a b c d e
5 a b c d e
4 b c d e
3 c d e
2 d e
1 e
0
$
```

Если попытаться выполнить команду `shift` в отсутствие переменных для смещения (т.е. когда значение переменной `$#` уже равно нулю), то оболочка выдает следующее сообщение об ошибке, хотя его содержимое зависит от конкретной версии оболочки:

```
prog: shift: bad number
```

где `prog` — имя программы, в которой была выполнена команда `shift`, а `bad number` — неверное количество аргументов.

Смещение может быть произведено сразу на несколько позиций, если указать их количество при вызове команды `shift`. Так, выполнение следующей команды:

```
shift 3
```

дает такой же результат, как и выполнение подряд трех команд `shift` без аргументов:

```
shift  
shift  
shift
```

Команда `shift` удобна для обработки переменного количества аргументов. Ее практическое применение будет продемонстрировано в главе 8, когда речь пойдет о циклах. А до тех пор достаточно запомнить, что позиционные параметры можно продвигать по цепочке, используя команду `shift`.