# 6
# 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 laserl-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 с 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: arg3 = :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          Действует по-прежнему
$ lu "Susan T"                     А теперь попробуем ввести
                                   тот же самый аргумент снова
Susan Topple    212-555-4932
$
```

## 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.

```
$ rem '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
$ rem 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, *смещать 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 выдаст 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`.