Автоматизация задач с помощью *tcsh*

1. Использование *tcsh*

Изначально оболочка `tcsh` была разработана для операционной системы TENEX, которая использовалась в далеком прошлом на компьютерах DEC PDP-10. В Linux по умолчанию используется оболочка `bash`, а вот в FreeBSD по умолчанию используется именно `tcsh`. Поэтому данная глава будет особенно полезна для пользователей Linux, которые планируют перейти на FreeBSD, или же для пользователей FreeBSD, которые хотят больше узнать про свою любимую оболочку.

Вообще, выбор оболочки больше зависит от ваших предпочтений. В какой бы операционной системе вы ни работали, вы всегда сможете изменить оболочку по умолчанию: в FreeBSD вам никто не запретит использовать по умолчанию `bash`, равно как и в Linux использовать `tcsh`. В некоторых дистрибутивах, например в Ubuntu, `tcsh` по умолчанию не установлена. Для ее установки нужно ввести команду:

```
sudo apt-get install tcsh
```

Сейчас не хочется делать ни экскурс в историю, ни проводить сравнение с `csh`. Скажу только, что во всех свободных системах (Linux, FreeBSD) файл /bin/csh — это ссылка на /bin/tcsh.

Как и `bash`, `tcsh` позволяет создавать сценарии, а язык оболочки `tcsh` насчитывает 65 встроенных команд. Просмотреть список этих команд можно командой `builtins`. На рис. 1 приведен вывод этой команды.

Основные интерактивные возможности `tcsh` следующие:

- редактирование командной строки;
- дополнение слов — как команд, так и путей;
- хранение и возможность просмотра истории команд;
- управление задачами.

Редактирование командной строки осуществляется с помощью клавиш <Left> и <Right> (перемещение курсора по командной строке) и клавиш <Delete> и <Backspase> (удаление символов). Как видите, все просто.

Для автоматического дополнения слов, как и в `bash`, используется клавиша <Tab>, но правильнее использовать комбинации клавиш <Ctrl>+<I> и <Ctrl>+<D>.

Первая обеспечивает автодополнение слова, а вторая выводит список подходящих вариантов для автодополнения.

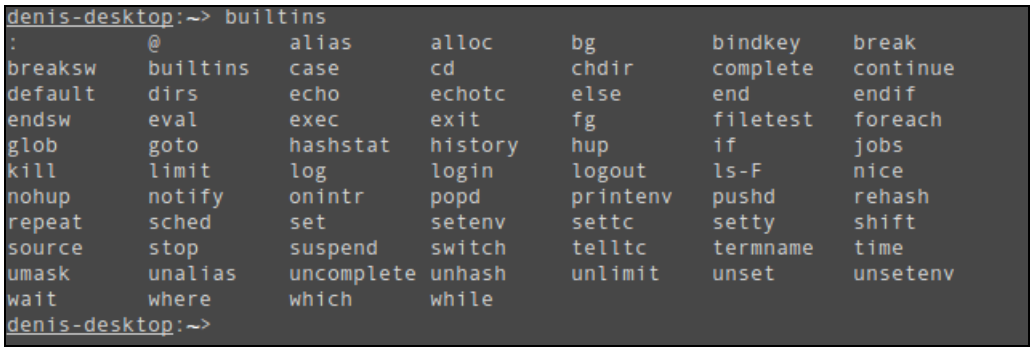

Рис. 1. Список встроенных команд оболочки `tcsh`

Просмотреть историю ранее введенных команд можно с помощью клавиш <Up> и <Down>. Если же эти клавиши почему-то не работают (меньше нужно играть в NFS!), то можно использовать команду `history`:

```
$ history
1   13:08  clear
2   13:09  builtins
3   13:19  history
```

Вызвать любую команду из истории можно с помощью конструкции `!#`, где `#` — это номер команды, например:

```
!1
```

Управление задачами осуществляется так же, как и в `bash`. Чтобы запустить команду в фоновом режиме, нужно добавить к ней символ `&`, например:

```
$ command &
```

Вывести список запущенных в фоновом режиме задач можно командой `jobs`. Команда `fg` переводит задачу в активный режим (foreground), нужно указать ее номер (полученный командой `jobs`), например:

```
$ fg 1
```

Команда `bg` переводит активную задачу в фоновый режим (background), как и для команды `fg`, нужно указать номер задачи:

```
$ bg 1
```

2. Конфигурационные файлы *tcsh*

Основные глобальные файлы конфигурации `tcsh` — /etc/csh.cshrc, /etc/csh.login, /etc/csh.logout. Первый файл считывается при запуске каждого экземпляра `tcsh` в интерактивном режиме, второй — только если `tcsh` является оболочкой по умол-

чанию для запустившего ее пользователя. Третий файл считывается при выходе из tcsh при условии, что tcsh является оболочкой по умолчанию для этого пользователя (является так называемым login shell).

При регистрации пользователя в домашнем каталоге создаются файлы ~/.cshrc, ~/.login и ~/.logout, которые являются пользовательскими аналогами /etc/csh.cshrc и /etc/csh.login. Содержимое этих файлов копируется из каталога /etc/skel (или /usr/share/skel/ — в некоторых системах).

Порядок считывания глобальных и пользовательских конфигурационных файлов задается при компиляции tcsh и может отличаться в разных системах. Обычно сначала считываются глобальные файлы, а затем пользовательские.

Также в пользовательском каталоге есть файл ~/.history, содержащий историю введенных команд.

В конфигурационных файлах устанавливаются псевдонимы команд (команда alias), переменные окружения (команда setenv), служебные переменные tcsh (команда set), влияющие на поведение оболочки. Определять переменные окружения имеет смысл ~/.login, который считывается только один раз — при входе пользователя в систему, а файл ~/.cshrc перечитывается при запуске каждого экземпляра.

Рассмотрим пример определения псевдонимов команд:

```
alias hist  history 25
alias rm rm -i
```

Первая команда создает псевдоним hist, выводящий последние 25 команд истории. Вторая команда создает псевдоним для команды rm, который называется так же — rm, а параметр -i означает, что при удалении файлов будет сделан запрос.

Вот пример использования команды set, которая устанавливает значение переменной path (путь поиска программ):

```
set path = (/bin /usr/bin /usr/local/bin /usr/X11R6/bin)
```

Переменные окружения устанавливаются командой setenv (в файле ~/.login), например:

```
setenv  EDITOR  nano
```

Здесь мы установили переменную окружения EDITOR, задающую текстовый редактор по умолчанию.

3. Создание сценариев на *tcsh*

3.1. Переменные, массивы и выражения

Переменные, как уже было показано, устанавливаются командой set:

```
set name = denis
```

Здесь мы установили переменную name (значение — denis). Вывести значение переменной можно так:

```
echo $name
```

Если вы введете просто команду set (без параметров), то вы увидите список уже установленных переменных, в том числе и служебных (рис. 16.2).

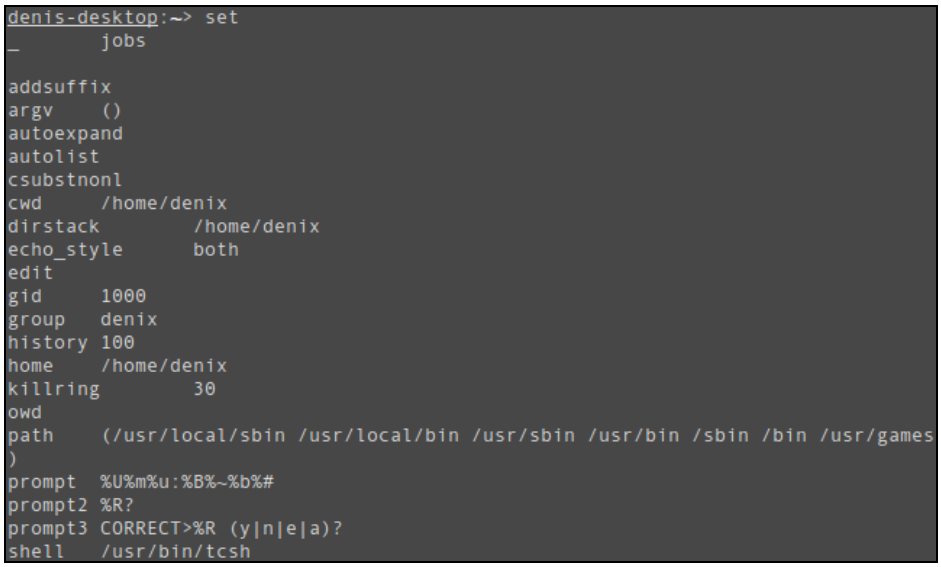

Рис. 2. Команда `set`

Удалить переменную можно командой `unset`:

```
unset name
```

Тогда при обращении к переменной вы увидите сообщение:

```
name: Undefined variable.
```

Переменные окружения принято устанавливать командой `setenv` так:

```
setenv переменная значение
```

Обратите внимание, что между именем переменной и значением нет знака равенства. Пример:

```
setenv  EDITOR  nano
```

В `tcsh` также можно создавать массивы. Вот пример создания и использования массива:

```
$ set nums = (one two three four five)

$ echo $nums

one two three four five

$ echo $nums[3]

three

$ echo $nums[1-3]

   one two three
```

Как видите, мы можем вывести сразу весь массив, конкретный элемент и диапазон элементов, что очень удобно. Нумерация элементов массива начинается с 1.

Отдельного разговора заслуживают числовые переменные. Чтобы присвоить переменной число или результат арифметического выражения, нужно использовать символ `@`, при этом символ `$` перед именем переменной указывать не нужно.

Рассмотрим несколько примеров:

```
$ @ num = 0

$ echo $num

0

$ @ num = ( 2 + 2 ) * 2

$ echo $num

8

$ @ num += 5

$ echo $num

13

$ @ num++

$ echo $num

14

$ @ num2 = 5

$ @ num = $num2 + 5

$ echo $num

10

$ @ num = 1

echo $nums[$num]

one
```

Когда вы используете переменные в выражениях, тогда нужно указывать символ `$`, а после символа `@` указывать `$` не нужно. После `@` обязательно должен быть пробел.

Общий синтаксис при работе с числовыми переменными выглядит так:

```
@ переменная оператор выражение
```

Оператор — это один из операторов присваивания C: =, +=, – =, *=, /= или %=. Выражение — это арифметическое выражение, которое тоже строится по тем же правилам, что и в языке C. Ничего удивительного: `tcsh` предназначена для тех, кто знаком с языком C, и должна была облегчить создание сценариев C-программистам.

Теперь рассмотрим пример работы с массивами чисел:

```
$ set a = (0 0 0 0 0)

$ @ a[1] = 10

$ @ a[3] = ($ages[1] + 5)

$ echo $a[3]

15
```

3.2. Чтение ввода пользователя

Для чтения ввода пользователя используется конструкция "$<", например:

```
echo -n "Введите строку: "

set line = "$<"
```

3.3. Переменные оболочки *tcsh*

При создании сценариев на tcsh вы можете использовать переменные оболочки, представленные в табл. 1.

Таблица 1. *Переменные оболочки* `tcsh`

Переменная	Описание
argv	Массив содержит аргументы командной строки (параметры, переданные сценарию). argv[1] — первый параметр, а argv[0] — это имя самого сценария. Обратиться к параметрам можно через конструкцию argv[n] или $n, например argv[1] или $1. Элемент массива argv[*] содержит все параметры вместе
autolist	Контролирует завершение команд и переменных. См. man tcsh
autologout	Включает возможность автоматического выхода. По умолчанию — 60 минут, если вы суперпользователь. Данная переменная не устанавливается (по умолчанию) для обычных пользователей
cdpath	Влияет на команду cd, задает путь поиска команд, обычно устанавливается в файле ~/.login, например: set cdpath = (/home/denix /home/denix/bin)
cwd	Содержит имя текущего каталога
fignore	Содержит массив суффиксов, которые tcsh будет игнорировать при завершении имен файлов
gid	Содержит идентификатор группы
histfile	Переменная содержит полное имя файла, в котором находится история команд. По умолчанию ~/.history
history	Размер списка истории
home или HOME	Имя домашнего каталога пользователя

Таблица 1 *(окончание)*

Переменная	Описание
`mail`	Содержит имя файла для проверки почты. TC Shell каждые 10 минут проверяет почту
`owd`	Предыдущий рабочий каталог
`path` или `PATH`	Путь поиска программ, обычно устанавливается в конфигурационных файлах `tcsh`: `set path = ( /usr/bin /bin /usr/local/bin /usr/bin/X11 ~/bin . )`
`prompt`	Содержит формат приглашения командной строки, аналогична переменной `PS1` в `bash`. Модификаторы формата описаны в табл. 16.2. Значение по умолчанию: `set prompt = '! $ '`
`prompt2`	Содержит формат приглашения командной строки для управляющих структур `while` и `foreach`
`prompt3`	Содержит формат приглашения командной строки при проверке правописания
`savehist`	Количество команд, которые будут сохранены в файл истории
`shell`	Полный путь к исполнимому файлу оболочки
`status`	Код завершения последней выполненной команды
`tcsh`	Версия `tcsh`
`time`	Может содержать только число или же буквы и цифры. Если переменная содержит только число, то это количество секунд процессорного времени. Если выполнение какой-то команды заняло больше времени, чем указано в `time`, то после выполнения команды будет выведено, сколько времени она занимала. Если `time = 0`, то после каждой команды будет выведено время выполнения. Если `time` содержит буквы и цифры (табл. 16.3), то это просто формат времени
`user`	Имя пользователя

Таблица 2. *Формат командной строки*

Модификатор	Описание
`%/`	Текущий каталог
`%~`	То же, что и `%/`, но заменяет путь к домашнему каталогу пользователя тильдой
`%!` или `%h` или `!`	Текущий номер события
`%m`	Имя узла без домена

***Таблица 2** (окончание)*

Модификатор	Описание
%M	Полное имя узла, с доменом
%n	Имя пользователя
%t	Время дня
%p	Время дня (с секундами)
%d	День недели
%D	День месяца
%W	Месяц, формат мм
%y	Год, формат гг
%Y	Год, формат гггг
%#	Решетка (#) для суперпользователя или знак больше (>) для обычного пользователя
%?	Код завершения последней команды

***Таблица 3.** Формат времени*

Модификатор	Описание
%U	Время, проведенное командой в пользовательском режиме (в процессорных секундах)
%S	Время, проведенное командой в режиме ядра (в процессорных секундах)
%W	Сколько раз процесс команды был выгружен на диск
%X	Средний размер сегмента кода программы, в килобайтах
%D	Средний объем памяти, используемый командой, в килобайтах
%K	Общий размер памяти, занятый командой (считается как %X+%D), в килобайтах
%M	Максимальный объем памяти, занятый командой, в килобайтах
%F	Количество ошибок страниц памяти
%I	Количество операций ввода
%O	Количество операций вывода

Перед тем как приступить к рассмотрению управляющих структур, рассмотрим применение скобок в tcsh. Предположим есть переменная:

```
$ set aa=abra
```

Потом нам нужно вывести ее в составе строки, для этого мы будем использовать фигурные скобки:

```
$ echo ${aa}cadabra
abracadabra
```

3.4. Управляющие структуры

Условный оператор *if*

Синтаксис оператора `if` очень прост:

```
if (выражение) команда
```

Команда будет выполнена, если выражение истинно. Выражения формируются так же, как в языке C. В листинге 1 представлен небольшой сценарий, проверяющий количество аргументов, переданных ему.

Листинг 1. Первый сценарий на `tcsh`

```
#!/bin/tcsh

if ( $#argv == 0 ) echo "Аргументы не заданы"
```

Также можно использовать следующее выражение:

```
-n имя_файла
```

В данном случае возможные варианты n представлены в табл. 4.

Таблица 4. *Значения n*

n	Описание
b	Файл является блочным устройством (обмен данными с устройством осуществляется блоками данных)
c	Файл является символьным устройством (обмен данными с устройством осуществляется посимвольно)
d	Файл является каталогом
e	Файл существует
g	Для файла установлен бит SGID *(см. главу 4)*
k	Для файла установлен "липкий" бит
l	Файл является символической ссылкой
o	Файл принадлежит текущему пользователю
p	Файл является именованным потоком (FIFO)
r	У пользователя есть право чтения файла
s	Файл не пустой (ненулевой размер)
S	Файл является сокетом
t	Дескриптор файла открыт и подключен к экрану
u	Для файла установлен SUID *(см. главу 4)*
w	У пользователя есть право записи файла

***Таблица 4** (окончание)*

n	Описание
x	У пользователя есть право выполнения файла
X	Файл является встроенной командой или его исполнимый файл найден при поиске в каталогах, указанных в `$path`
z	Файл пуст (нулевой размер)

Рассмотрим пример использования данного условия:

```
if -e $1 echo "Файл существует"
```

Условный оператор *if..then..else*

Условный оператор `if..then..else` похож на `if`, только добавляется блок `else` (иначе), который выполняет команды в случае ложности условия. Сокращенная версия оператора выглядит так:

```
if (выражение) then

    команды, которые будут выполнены в случае истинности выражения

endif
```

Полная версия оператора выглядит так:

```
if (выражение) then

    команды, которые будут выполнены в случае истинности выражения

else

    команды, которые будут выполнены в противном случае (когда выраже-
ние = false)

endif
```

Существует еще одна форма этого оператора, точнее, это отдельный оператор `if..then..elif`:

```
if (выражение1) then

     команды (если выражение1 = true)

else if (выражение2) then

    команды (если выражение2 = true)

. . .

else

     команды (если ни одно из выражений не равно true)

endif
```

Рассмотрим небольшой пример использования условного оператора (листинг 2).

Листинг 2. Пример использования условного оператора

```
#!/bin/tcsh

# Получаем число из командной строки
set num = $argv[1]

set flag
#

if ($num < 0) then

    @ flag = 1

else if (0 <= $num && $num < 50) then

    @ flag = 2

else if (50 <= $num && $num < 1000) then

    @ flag = 3

else

    @ flag = 4

endif

#

echo "Flag: ${flag}."
```

Оператор *foreach*

Оператор `foreach` удобно использовать для перебора массивов, его синтаксис:

```
foreach индекс-цикла (список-аргументов)
     команды
end
```

Также цикл `foreach` удобно использовать при работе с файлами, например:

```
foreach f ( *.txt )
   echo $f
end
```

Здесь мы в цикле выводим имена всех текстовых файлов в текущем каталоге. Конечно, для этой цели проще вызвать команду `ls`, но здесь мы продемонстрировали использование `foreach`.

Оператор *while*

Оператор `while` — это еще один вариант цикла. Команды, находящиеся в теле цикла, выполняются до тех пор, пока выражение истинно:

```
while (выражение)
    команды
end
```

Рассмотрим пример сценария, вычисляющего факториал числа n, при этом n задается в командной строке (листинг 3).

Листинг 3. Пример использования `while`

```
#!/bin/tcsh

set n = $argv[1]

set i = 1

set fact = 1

#

while ($i <= $n)

    @ fact *= $i

    @ i++

end

#

echo "Факториал числа $n равен $fact"
```

В циклах вы можете использовать операторы `break` и `continue`. Оператор `break` прерывает цикл и передает управление оператору, следующему за `end`. Оператор `continue` прерывает только текущую итерацию цикла и передает управление оператору `end`, который после получения управления начнет следующую итерацию цикла.

Оператор *switch*

В ряде случаев использовать оператор `switch` намного удобнее, чем серию условных операторов. Синтаксис `switch` следующий:

```
switch (строка)

    case образец1:

        команды1

    breaksw


    case образец2:

        команды2

    breaksw

    ...

    default:

        команды по умолчанию

    breaksw

endsw
```

Вот пример использования данного оператора:

```
set string = test

switch (string)
    case test:
          echo "Строка: test"
    breaksw
    case text:
          echo "Строка: text"
    breaksw
    default:
          echo "Строка не опознана"
    breaksw
endsw
```

3.5. Встроенные команды *tcsh*

Оболочка `tcsh` обладает встроенными командами (как, впрочем, и любая другая оболочка). В табл. 5 приведены самые полезные встроенные команды `tcsh`.

Таблица 5. *Самые полезные встроенные команды* `tcsh`

Команда	**Описание**
`@`	Вычисляет арифметическое выражение. Данная команда была подробно описана ранее при написании сценариев
`alias`	Создает псевдоним для команд оболочки, эту команду мы тоже рассмотрели ранее
`alloc`	Отображает отчет о количестве свободной и использованной памяти
`bg`	Перемещает приостановленную задачу в фоновый режим
`builtins`	Отображает список встроенных команд
`cd или chdir`	Изменяет текущий каталог
`dirs`	Отображает стек каталогов
`echo`	Отображает свои аргументы. Обычно используется при написании сценариев
`exec`	Запускает другую программу в этой же оболочке
`exit`	Осуществляет выход из `tcsh`
`fg`	Перемещает задачу на "передний план", т. е. делает ее активной
`filetest`	В основном используется при написании сценариев. Позволяет проверить тип файла (устройство, каталог, файл), существование файла и осуществить ряд других полезных проверок (см. табл. 16.4)
`glob`	Похожа на `echo`, но не отображает пробелы между аргументами и не выводит символ новой строки после своего вывода
`hashstat`	Выводит статистику механизма хэширования `tcsh`
`history`	Отображает введенные ранее команды
`jobs`	Отображает список задач (приостановленные команды и те, которые выполняются в фоновом режиме)
`kill`	Завершает задачу или процесс
`limit`	Позволяет ограничить ресурсы текущего процесса и всех процессов, которые он будет создавать
`login`	Используется для входа пользователя. Сопровождается именем пользователя
`logout`	Завершает сессию, если вы используете `tcsh` в качестве оболочки по умолчанию
`ls-F`	Похожа на `ls -F`, но выполняется быстрее
`nice`	Позволяет изменить приоритет процесса *(см. разд. 5.3)*
`nohup`	Позволяет вам выйти из системы без завершения процессов, выполняемых в фоновом режиме
`notify`	Уведомляет вас при изменении статуса одной из ваших задач

***Таблица 5** (окончание)*

Команда	Описание
popd	Удаляет каталог из стека каталогов
printenv	Отображает список всех переменных окружения
pushd	Изменяет рабочий каталог и помещает новый каталог на вершину стека каталогов
rehash	Пересоздает внутренние таблицы, используемые механизмом хэширования. Допустим, вы создали свой сценарий и поместили его в /usr/bin. Оболочка `tcsh` не увидит этот файл до тех пор, пока вы не обновите внутренние таблицы
repeat	Команде нужно передать два аргумента — количество повторений и простую команду (без потоков и списка команд), и повторяет эту команду указанное количество раз
sched	Выполняет команду в указанное время, например: `$ sched 10:00 echo "Уже 10 часов!"`
set	Объявляет и инициализирует локальную переменную
setenv	Объявляет и инициализирует переменную окружения
source	Запускает сценарий оболочки, указанный в качестве аргумента. Данная команда не порождает новый процесс, а просто обрабатывает сценарий. Данную команду можно использовать как `include` в обычных языках программирования
stop	Останавливает задачу или процесс (которая или который выполняется в фоновом режиме)
suspend	Приостанавливает текущую оболочку и помещает ее в фоновый режим
time	Запускает команду, указанную в качестве параметра. После выполнения команды отображает информацию о ее выполнении (время выполнения)
umask	Изменяет маску прав доступа по умолчанию (см. `man umask`)
unalias	Удаляет псевдоним команды
unhash	Выключает механизм хэширования. См. также `hashstat` и `rehash`
unlimit	Удаляет лимиты текущего процесса
unset	Удаляет объявление переменной
unsetenv	Удаляет объявление переменной окружения
where	В качестве аргумента нужно передать команду, после чего получите информацию о том, является ли команда встроенной или исполнимым файлом и где находится этот файл. Похожа на команду `which`
which	Подобна стандартной утилите `which`, но работает быстрее и владеет информацией обо всех командах и псевдонимах. Выводит местонахождение исполнимого файла, если он вообще существует. Поиск производится в каталогах, указанных в переменной окружения `path`