UNIX/Linux системы в инфокуммуникациях

Программирование на Bash

В. Титов, О Садов
tit@linux-ink.ru

Университет Информационных Технологий, Механики и Оптики

9.11.2015

Почему Bash?

- Он всегда есть
- Вы уже используете его
- Все системные скрипты работают в bash
- Многие проекты, инструменты, системы в значительной части реализованы на bash

Почему Bash?

- Он всегда есть
- Вы уже используете его
- Все системные скрипты работают в bash
- Многие проекты, инструменты, системы в значительной части реализованы на bash

Почему Bash?

- Он всегда есть
- Вы уже используете его
- Все системные скрипты работают в bash
- Многие проекты, инструменты, системы в значительной части реализованы на bash

Почему Bash?

- Он всегда есть
- Вы уже используете его
- Все системные скрипты работают в bash
- Многие проекты, инструменты, системы в значительной части реализованы на bash

Переменные среды

Зарегистрировавшисть в системе (или войдя в терминал), мы оказываемся в оболочке bash.
Одно из самых удобных свойств переменных среды заключается в том, что они являются стандартной частью модели процессов UNIX.
Это значит, что переменные среды доступны не только скриптам оболочки, но их также можно использовать и в других программах. Хороший пример - команда 'vipw', которая обычно позволяет пользователю root редактировать системный файл паролей. Определив в переменной среды 'EDITOR' ваш любимый редактор, вы тем самым говорите 'vipw', что нужно использовать именно его. Другой пример: crtl-x ctrl-e.
Стандартный способ определить переменную среды в bash:

```
$ myvar='Это моя переменная среды!'
```

Переменные среды, 2

- с обеих сторон от знака "="нет никаких пробелов;
- кавычки необходимы, если значение переменной среды состоит из более чем одного слова;
- " vs '.

Попробуйте!

Переменные среды, 3

Использование переменных

```
$ echo $myvar
$ echo foo$myvarbar
$ echo foo$myvarbar
```

Переменные среды, 3

Использование переменных

```
$ echo $myvar
$ echo foo$myvarbar
$ echo foo$myvarbar
```

Переменные среды, 3

Использование переменных

```
$ echo $myvar
$ echo foo$myvarbar
$ echo foo$myvarbar
```

Переменные среды, 4

Если мы экспортируем переменную среды, она автоматически становится доступной в среде любого запущенного позже скрипта или программы.
Скрипты оболочки могут "добраться"до переменной среды с помощью встроенной в оболочку поддержки переменных среды, а программы C могут использовать вызов функции getenv().
Вот пример соответствующего кода C:

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  char *myenvvar=getenv("EDITOR");
  printf("Environment variable EDITOR is set to %s\n",
         myenvvar);
}
```

Скомпилируйте, проверьте.

Переменные среды, 5

Проверьте вашу программу с переменной, поиграйтесь с командами export и unset.

Разделение исходной строки на отдельные куски — из тех задач, которые ежедневно выполняются скриптами оболочки. Часто скриптам оболочки требуется взять полностью указанный путь и найти завершающий файл или директорию. Стандартная команда для этого basename:

```
$ basename /usr/local/share/doc/foo/foo.txt
foo.txt
$ basename /home/test
test
```

Переменные среды, 5

Проверьте вашу программу с переменной, поиграйтесь с командами export и unset.
Разделение исходной строки на отдельные куски — из тех задач, которые ежедневно выполняются скриптами оболочки.
Часто скриптам оболочки требуется взять полностью указанный путь и найти завершающий файл или директорию.
Стандартная команда для этого basename:

```
$ basename /usr/local/share/doc/foo/foo.txt
foo.txt
$ basename /home/test
test
```

Переменные среды, 5

Проверьте вашу программу с переменной, поиграйтесь с командами export и unset.
Разделение исходной строки на отдельные куски — из тех задач, которые ежедневно выполняются скриптами оболочки.
Часто скриптам оболочки требуется взять полностью указанный путь и найти завершающий файл или директорию.
Стандартная команда для этого basename:

```
$ basename /usr/local/share/doc/foo/foo.txt
foo.txt
$ basename /home/test
test
```

Переменные среды, 6

или dirname:

```
$ dirname /usr/local/share/doc/foo/foo.txt
/usr/local/share/doc/foo
$ dirname /home/test
/home
```

Прежде чем рассмотреть рассмотреть другие способы выделения подстрок, рассмотрим подставновку команд.

Командная строка

Горячие клавиши:

- Перемещение/удаление: ctrl-a, ctrl-e, ctrl-w, ctrl-u, ctrl-k, ctrl-l, alt-b, alt-f
- История: history, !$, !!, ctrl-rTab
- Редактирование alt-#, ctrl-t, ctrl-x ctrl-e
- alias

Автозавершение:
Tab

Подстановка команд

Как создать переменную среды, которая содержит результат выполнения команды:

```
$ MYDIR=$(dirname /etc/X11/xinit/xinitrc)
$ echo $MYDIR
/etc/X11/xinit
```

или

```
$ MYDIR=‘dirname /etc/X11/xinit/xinitrc‘
$ echo $MYDIR
/etc/X11/xinit
```

Можно использовать и конвейер

```
$ MYFILES=$(ls /etc | grep pam)
$ echo $MYFILES
pam.d
pam_smb.conf
```

Подстановка команд

Как создать переменную среды, которая содержит результат выполнения команды:

```
$ MYDIR=$(dirname /etc/X11/xinit/xinitrc)
$ echo $MYDIR
/etc/X11/xinit
```

или

```
$ MYDIR=‘dirname /etc/X11/xinit/xinitrc‘
$ echo $MYDIR
/etc/X11/xinit
```

Можно использовать и конвейер

```
$ MYFILES=$(ls /etc | grep pam)
$ echo $MYFILES
pam.d
pam_smb.conf
```

Подстановка команд

Как создать переменную среды, которая содержит результат выполнения команды:

```
$ MYDIR=$(dirname /etc/X11/xinit/xinitrc)
$ echo $MYDIR
/etc/X11/xinit
```

или

```
$ MYDIR=‘dirname /etc/X11/xinit/xinitrc‘
$ echo $MYDIR
/etc/X11/xinit
```

Можно использовать и конвейер

```
$ MYFILES=$(ls /etc | grep pam)
$ echo $MYFILES
pam.d
pam_smb.conf
```

Подстановка команд

Как создать переменную среды, которая содержит результат выполнения команды:

```
$ MYDIR=$(dirname /etc/X11/xinit/xinitrc)
$ echo $MYDIR
/etc/X11/xinit
```

или

```
$ MYDIR=‘dirname /etc/X11/xinit/xinitrc‘
$ echo $MYDIR
/etc/X11/xinit
```

Можно использовать и конвейер

```
$ MYFILES=$(ls /etc | grep pam)
$ echo $MYFILES
pam.d
pam_smb.conf
```

Раскрытие

- ${name:-word}
- ${name:?error message}
- ${1:?usage: $0 input_file}
- ${#name}
- {1..10}
- $(((i + 1) % 5))

Выделение строк

```
$ MYVAR=foodforthought.jpg
$ echo ${MYVAR##*fo}
rthought.jpg
$ echo ${MYVAR#*fo}
odforthought.jpg
```

Сравните с

```
$ MYFOO="chickensoup.tar.gz"
$ echo ${MYFOO%%.*}
chickensoup
$ echo ${MYFOO%.*}
chickensoup.tar
```

Выделение строк

```
$ MYVAR=foodforthought.jpg
$ echo ${MYVAR##*fo}
rthought.jpg
$ echo ${MYVAR#*fo}
odforthought.jpg
```

Сравните с

```
$ MYFOO="chickensoup.tar.gz"
$ echo ${MYFOO%%.*}
chickensoup
$ echo ${MYFOO%.*}
chickensoup.tar
```

Выделение строк, 3

```
$ EXCLAIM=cowabunga
$ echo ${EXCLAIM:0:3}
cow
$ echo ${EXCLAIM:3:7}
abunga
```

Пример:

```
#!/bin/bash

if [ "${1##*.}" = "tar" ]
then
        echo По-видимому, это архив tar.
else
        echo На первый взгляд это не кажется архивом tar.
fi
```

Выделение строк, 3

```
$ EXCLAIM=cowabunga
$ echo ${EXCLAIM:0:3}
cow
$ echo ${EXCLAIM:3:7}
abunga
```

Пример:

```
#!/bin/bash

if [ "${1##*.}" = "tar" ]
then
        echo По-видимому, это архив tar.
else
        echo На первый взгляд это не кажется архивом tar.
fi
```

Предложение if

```
if        [ условие ]
then
        действие
fi
```

или

```
if [ условие ]
then
        действие
elif [ условие2 ]
then
        действие2
        .
elif [ условие3 ]
then
        .
else
        действиеX
fi
```

Предложение if

```
if       [ условие ]
then
        действие
fi
```

или

```
if [ условие ]
then
        действие
elif [ условие2 ]
then
        действие2
        .
elif [ условие3 ]
then
        .
else
        действиеX
fi
```

Аргументы

Мы уже встретили $1. Как задать другие аргументы.
Вот пример:

```
#!/bin/bash

echo имя скрипта $0
echo первый аргумент $1
echo второй аргумент $2
echo семнадцатый аргумент $17
echo число аргументов $#
```

Можно сослаться на все аргументы командной строки сразу $@.

Аргументы

Мы уже встретили $1. Как задать другие аргументы.
Вот пример:

```
#!/bin/bash

echo имя скрипта $0
echo первый аргумент $1
echo второй аргумент $2
echo семнадцатый аргумент $17
echo число аргументов $#
```

Можно сослаться на все аргументы командной строки сразу $@.

Условия

Вот примеры

```
if [ -z "$myvar" ]
then
     echo "myvar не определена"
fi

if [ "$myvar" -eq 3 ]
then
     echo "myvar равна 3"
fi

if [ "$myvar" = "3" ]
then
     echo "myvar равна 3"
fi
```

Условия

Вот примеры

```
if [ -z "$myvar" ]
then
     echo "myvar не определена"
fi

if [ "$myvar" -eq 3 ]
then
     echo "myvar равна 3"
fi

if [ "$myvar" = "3" ]
then
     echo "myvar равна 3"
fi
```

Условия. Подводные камни

```
if [ $myvar = "foo bar oni" ]
then
     echo "да"
fi
```

```
[: too many arguments
```

```
[ foo bar oni = "foo bar oni" ]
```

Правильный вариант

```
if [ "$myvar" = "foo bar oni" ]
then
    echo "да"
fi
```

' vs ".

Условия. Подводные камни

```
if [ $myvar = "foo bar oni" ]
then
     echo "да"
fi
```

```
[: too many arguments
```

```
[ foo bar oni = "foo bar oni" ]
```

Правильный вариант

```
if [ "$myvar" = "foo bar oni" ]
then
    echo "да"
fi
```

' vs ".

Условия. Подводные камни

```
if [ $myvar = "foo bar oni" ]
then
     echo "да"
fi
```

```
[: too many arguments
```

```
[ foo bar oni = "foo bar oni" ]
```

Правильный вариант

```
if [ "$myvar" = "foo bar oni" ]
then
    echo "да"
fi
```

' vs ".

Условия. Подводные камни

```
if [ $myvar = "foo bar oni" ]
then
     echo "да"
fi
```

```
[: too many arguments
```

```
[ foo bar oni = "foo bar oni" ]
```

Правильный вариант

```
if [ "$myvar" = "foo bar oni" ]
then
    echo "да"
fi
```

' vs ".

Цикл for

```
#!/bin/bash

for x in one two three four
do
    echo number $x
done
```

Вывод:

```
number one
number two
number three
number four
```

Цикл for, 2

Вот более реальный пример:

```
#!/bin/bash

for myfile in /etc/r*
do
    if [ -d "$myfile" ]
    then
      echo "$myfile (dir)"
    else
      echo "$myfile"
    fi
done
```

Вывод:

```
/etc/rc.d (dir)
/etc/resolv.conf
/etc/resolv.conf~
/etc/rpc
```

Цикл for, 3

Можно и так

```
for x in /etc/r??? /var/lo* /home/test/* /tmp/${MYPATH}/*
do
    cp $x /mnt/mydira
done
```

и так

```
#!/bin/bash
for thing in "$@"
do
    echo вы набрали ${thing}.
done
```

```
$ allargs hello there you silly
```

Вывод:

```
вы набрали hello.
вы набрали there.
...
```

Цикл for, 3

Можно и так

```
for x in /etc/r??? /var/lo* /home/test/* /tmp/${MYPATH}/*
do
    cp $x /mnt/mydira
done
```

и так

```
#!/bin/bash
for thing in "$@"
do
    echo вы набрали ${thing}.
done
```

```
$ allargs hello there you silly
Вывод:
вы набрали hello.
вы набрали there.
...
```

Цикл for, 3

Можно и так

```
for x in /etc/r??? /var/lo* /home/test/* /tmp/${MYPATH}/*
do
    cp $x /mnt/mydira
done
```

и так

```
#!/bin/bash
for thing in "$@"
do
    echo вы набрали ${thing}.
done
```

```
$ allargs hello there you silly
```

Вывод:

```
вы набрали hello.
вы набрали there.
...
```

Арифметика bash

```
$ echo $(( 100 / 3 ))
33
$ myvar="56"
$ echo $(( $myvar + 12 ))
68
$ echo $(( $myvar - $myvar ))
0
$ myvar=$(( $myvar + 1 ))
$ echo $myvar
57
```

Циклы while

```
while [ условие ]
do
    предложения
done
```

Пример:

```
myvar=0
while [ $myvar -ne 10 ]
do
    echo $myvar
    myvar=$(( $myvar + 1 ))
done
```

Циклы while

```
while [ условие ]
do
    предложения
done
```

Пример:

```
myvar=0
while [ $myvar -ne 10 ]
do
    echo $myvar
    myvar=$(( $myvar + 1 ))
done
```

Циклы until

```
myvar=0
until [ $myvar -eq 10 ]
do
    echo $myvar
    myvar=$(( $myvar + 1 ))
done
```

Предложения case

Пример:

```
case "${x##*.}" in
      gz)
            gzunpack ${SROOT}/${x}
            ;;
      bz2)
            bz2unpack ${SROOT}/${x}
            ;;
      *)
            echo "Архивный формат не распознан."
            exit
            ;;
esac
```

Функции bash

Вот пример определения функции в bash:

```
tarview() {
    echo -n "Displaying contents of $1 "
    if [ ${1##*.} = tar ]
    then
        echo "(несжатый tar)"
        tar tvf $1
    elif [ ${1##*.} = gz ]
    then
        echo "(tar, сжатый gzip)"
        tar tzvf $1
    elif [ ${1##*.} = bz2 ]
    then
        echo "(tar, сжатый bzip2)"
        cat $1 | bzip2 -d | tar tvf -
    fi
}
```

Попробуйте то же самое сделать с помощью конструкции case.

Функции bash

Вот пример определения функции в bash:

```
tarview() {
    echo -n "Displaying contents of $1 "
    if [ ${1##*.} = tar ]
    then
        echo "(несжатый tar)"
        tar tvf $1
    elif [ ${1##*.} = gz ]
    then
        echo "(tar, сжатый gzip)"
        tar tzvf $1
    elif [ ${1##*.} = bz2 ]
    then
        echo "(tar, сжатый bzip2)"
        cat $1 | bzip2 -d | tar tvf -
    fi
}
```

Попробуйте то же самое сделать с помощью конструкции case.

Функции bash, 2

Если в оболочке мы уже определили функцию, то ее можно использовать:

```
$ tarview shorten.tar.gz
Вывод содержимого shorten.tar.gz (tar, сжатый gzip)
drwxr-xr-x ajr/abbot          0 1999-02-27 16:17 shorten-2.3a
-rw-r-r- ajr/abbot         1143 1997-09-04 04:06 shorten-2.3a/M
-rw-r-r- ajr/abbot         1199 1996-02-04 12:24 shorten-2.3a/I
-rw-r-r- ajr/abbot          839 1996-05-29 00:19 shorten-2.3a/I
....
```

Механизм передачи параметров функции ($1,...$9, а также числа параметров $#) тот же, что и при вызове скриптов bash. А вот $0 даст либо строку "bash либо имя скрипта, из которого вызывается функция.

Если вы запишете определение функции в ~/.bashrc или в ~/.bash_profile, вы сможете использовать ее всякий раз, когда окажетесь в bash.

Функции bash, 2

Если в оболочке мы уже определили функцию, то ее можно использовать:

```
$ tarview shorten.tar.gz
Вывод содержимого shorten.tar.gz (tar, сжатый gzip)
drwxr-xr-x ajr/abbot          0 1999-02-27 16:17 shorten-2.3a
-rw-r-r- ajr/abbot         1143 1997-09-04 04:06 shorten-2.3a/M
-rw-r-r- ajr/abbot         1199 1996-02-04 12:24 shorten-2.3a/I
-rw-r-r- ajr/abbot          839 1996-05-29 00:19 shorten-2.3a/I
....
```

Механизм передачи параметров функции ($1,...$9, а также числа параметров $#) тот же, что и при вызове скриптов bash. А вот $0 даст либо строку "bash либо имя скрипта, из которого вызывается функция.

Если вы запишете определение функции в ~/.bashrc или в ~/.bash_profile, вы сможете использовать ее всякий раз, когда окажетесь в bash.

Функции bash, 2

Если в оболочке мы уже определили функцию, то ее можно использовать:

```
$ tarview shorten.tar.gz
Вывод содержимого shorten.tar.gz (tar, сжатый gzip)
drwxr-xr-x ajr/abbot          0 1999-02-27 16:17 shorten-2.3a
-rw-r-r- ajr/abbot         1143 1997-09-04 04:06 shorten-2.3a/M
-rw-r-r- ajr/abbot         1199 1996-02-04 12:24 shorten-2.3a/I
-rw-r-r- ajr/abbot          839 1996-05-29 00:19 shorten-2.3a/I
....
```

Механизм передачи параметров функции ($1,...$9, а также числа параметров $#) тот же, что и при вызове скриптов bash. А вот $0 даст либо строку "bash либо имя скрипта, из которого вызывается функция.
Если вы запишете определение функции в ~/.bashrc или в ~/.bash_profile, вы сможете использовать ее всякий раз, когда окажетесь в bash.

Пространство имен

Если в bash вы создали внутри функции переменную среды, она будет добавлена в глобальное пространство имен. Это означает, что она перекроет любую глобальную переменную вне функции и будет продолжать существовать даже после выхода из функции:

```
#!/bin/bash
myvar="привет"
myfunc() {
    myvar="один два три"
    for x in $myvar
    do
        echo $x
    done
}

myfunc
echo $myvar $x
```

Пространство имен, 2

Будучи запущенным этот скрипт выдаст "один два три три". Здесь это легко поправить, изменив имя переменной, но лучший путь решения этой проблемы использовать локальные переменные.

```
#!/bin/bash
myvar="привет"
myfunc() {
    local x
    local myvar="один два три"
    for x in $myvar
    do
        echo $x
    done
}

myfunc
echo $myvar $x
```

Разное

- ., ../, ~/
- ()
- конвейеры
- && и ||
- раскрытие имен файлов
- { }
- <, <<, >, >>

```
diff file1 <(sort file1)
```

```
cat <<EOF .....EOF
```

Разное

- ., ../, ~/
- ()
- конвейеры
- && и ||
- раскрытие имен файлов
- { }
- <, <<, >, >>

```
diff file1 <(sort file1)
```

```
cat <<EOF .....EOF
```

Разное

- ., ../, ~/
- ()
- конвейеры
- && и ||
- раскрытие имен файлов
- { }
- <, <<, >, >>

```
diff file1 <(sort file1)
```

```
cat <<EOF .....EOF
```

Разное

- ., ../, ~/
- ()
- конвейеры
- && и ||
- раскрытие имен файлов
- { }
- <, <<, >, >>

```
diff file1 <(sort file1)
```

```
cat <<EOF .....EOF
```

Разное

- ., ../, ~/
- ()
- конвейеры
- && и ||
- раскрытие имен файлов
- { }
- <, <<, >, >>

```
diff file1 <(sort file1)
```

```
cat <<EOF ......EOF
```

Разное

- ., ../, ~/
- ()
- конвейеры
- && и ||
- раскрытие имен файлов
- { }
- <, <<, >, >>

```
diff file1 <(sort file1)
```

```
cat <<EOF .....EOF
```

Разное

- ., ../, ~/
- ()
- конвейеры
- && и ||
- раскрытие имен файлов
- { }
- <, <<, >, >>

```
diff file1 <(sort file1)
```

```
cat <<EOF ......EOF
```

Разное

- ., ../, ~/
- ()
- конвейеры
- && и ||
- раскрытие имен файлов
- { }
- <, <<, >, >>

```
diff file1 <(sort file1)
```

```
cat <<EOF ......EOF
```

Разное

- ., ../, ~/
- ()
- конвейеры
- && и ||
- раскрытие имен файлов
- { }
- <, <<, >, >>

```
diff file1 <(sort file1)
```

```
cat <<EOF ......EOF
```