Ошибки начинающих

Вот несколько типичных ловушек:

Выполнение скрипта

Ваш идеальный скрипт Bash выполняется с синтаксическими ошибками

Если вы пишете Bash-скрипты с синтаксисом и функциями, специфичными для Bash, запустите их с помощью Bash и запустите их с помощью Bash в собственном режиме.

Неправильно:

- без шуток
  - используемый интерпретатор зависит от реализации ОС () и текущей оболочки
  - **может** быть запущен путем вызова bash с именем скрипта в качестве аргумента, например `bash myscript`
- `#!/bin/sh` дело
  - зависит от того `/bin/sh` , что есть на самом деле, для Bash это означает режим совместимости, а **не** собственный режим

Смотрите также:

- Режим запуска Bash: режим SH
- Режим запуска Bash: режим POSIX

Ваш скрипт с именем "test" не выполняется

Дайте ему другое название. Исполняемый `test` файл уже существует.

В Bash это встроено. В других оболочках это может быть исполняемый файл. В любом случае, это плохой выбор имени!

Обходной путь: вы можете вызвать его, используя путь:

```
/home/user/bin/test
```

Глобализация

Расширение скобки не глобализируется

Следующая командная строка не связана с глобализацией (расширением имени файла):

```
# ВЫ ОЖИДАЕТЕ
# -i1.vob -i2.vob -i3.vob ....

echo -i{*.vob,}

# ВЫ ПОЛУЧАЕТЕ
# -i*.vob -i
```

Почему?Расширение фигурных скобок - это простая замена текста. Генерируется весь возможный текст, образованный префиксом, постфиксом и самими фигурными скобками. В примере их всего две: `-i*.vob` и `-i` . **После** этого происходит расширение имени файла, поэтому есть вероятность, что `-i*.vob` оно будет расширено до имени файла - если у вас есть такие файлы, как `-ihello.vob` . Но это определенно не делает того, чего вы ожидали.

Пожалуйста, посмотрите:

- Расширение скобки

Тест-команда

- `if [ $foo ]` …
- `if [-d $dir]` …
- …

Пожалуйста, посмотрите:

- Классическая тестовая команда - подводные камни

Переменные

Настройка переменных

Знак доллара

Нет `$` (знака доллара), когда вы ссылаетесь на **имя** переменной! Bash - это не PHP!

```
# ЭТО НЕПРАВИЛЬНО!
$myvar="Привет, мир!"
```

Имя переменной, перед которым стоит знак доллара, всегда означает, что переменная **расширяется**. В приведенном выше примере он может превратиться в ничто (поскольку он не был установлен), что фактически приведет к…

```
= "Привет, мир!"
```

... что **определенно неправильно**!

Когда вам нужно **имя** переменной, вы пишете **только имя**, например

- (как показано выше) для установки переменных: `picture=/usr/share/images/foo.png`
- чтобы назвать переменные, которые будут использоваться `read` встроенной командой: `read picture`
- для именования переменных, которые будут отключены: `unset picture`

Когда вам нужно **содержимое** переменной, вы добавляете к ее имени **знак доллара**, например

- echo "Используемое изображение: $ picture"

Пробелы

При присвоении значения переменной `=` не **удастся поставить пробелы по одну или обе стороны от знака равенства (**).

```
# НЕВЕРНО 1
пример = Привет

# НЕВЕРНО 2
пример= Привет

# НЕВЕРНО 3
пример = Привет
```

Единственная допустимая форма - **отсутствие пробелов между именем переменной и присвоенным значением**:

```
# ИСПРАВИТЬ 1
пример= Привет

# ИСПРАВИТЬ 2
пример="Привет"
```

Расширение (использование) переменных

Типичная ловушка новичка - цитирование.

Как отмечалось выше, когда вы хотите **расширить** переменную, то есть "получить содержимое", имя переменной должно иметь префикс со знаком доллара. Но, поскольку Bash знает различные способы цитирования и использует разделение слов, результат не всегда один и тот же.

Давайте определим пример переменной, содержащей текст с пробелами:

```
пример="Привет, мир"
```

Используемая форма	Результат	количество слов
`$example`	`Hello world`	2
`"$example"`	`Hello world`	1
`\$example`	`$example`	1
`'$example'`	`$example`	1

Если вы используете расширение параметров, вы **должны** использовать **имя** (`PATH`) переменных / параметров, на которые ссылаются ссылки. т.е. **not** (`$PATH`):

```
# НЕПРАВИЛЬНО!
echo "Первый символ ПУТИ - ${$PATH:0:1}"

# ИСПРАВЬТЕ
эхо "Первый символ ПУТИ - ${PATH:0:1}"
```

Обратите внимание, что если вы используете переменные в арифметических выражениях, тогда допускается простое **имя**:

```
((a = $ a +7)) # Добавить 7 к a
((a = a + 7)) # Добавить 7 к a. Идентично предыдущей команде.
((a += 7)) # Добавить 7 к a. Идентично предыдущей команде.

a = $ ((a + 7)) # POSIX-совместимая версия предыдущего кода.
```

Пожалуйста, посмотрите:

- Слова...
- Цитаты и экранирование
- Разделение слов
- Расширение параметров

Экспорт

Экспорт переменной означает предоставление **вновь созданным** (дочерним) процессам копии этой переменной. Он **не** копирует переменную, созданную в дочернем процессе, обратно в родительский процесс. Следующий пример **не** работает, поскольку переменная `hello` установлена в дочернем процессе (процессе, который вы выполняете для запуска этого скрипта `./script.sh`):

```
$ cat script.sh
экспорт привет=мир

$ ./script.sh
$ echo $привет
$
```

Экспорт односторонний. Направление - от родительского процесса к дочернему процессу, а не наоборот. Приведенный выше пример **будет** работать, если вы не выполняете скрипт, но включаете ("исходный код") его:

```
$ источник ./script.sh
$ echo $привет
, мир
$
```

В этом случае команда export бесполезна.

Пожалуйста, посмотрите:

- Bash и дерево процессов

Коды выхода

Реагирование на коды выхода

Если вы просто хотите отреагировать на код выхода, независимо от его конкретного значения, вам **не нужно** использовать `$?` в тестовой команде, подобной этой:

```
grep ^root: /etc/passwd >/dev/null 2>&1

 if [ $? -ne  0 ]; затем
   echo "корень не найден - проверьте паб на углу"
fi
```

Это можно упростить до:

```
grep ! если ^root: /etc/passwd >/dev/null 2>&1; затем
   повторите  "root не найден - проверьте паб на углу"
fi
```

Или, еще проще:

```
grep ^root: /etc/passwd >/dev/null 2> &1 || echo "корень не найден -
 проверьте паб на углу"
```

Если вам нужно конкретное значение `$?` , другого выбора нет. Но если вам нужна только индикация выхода "true / false", в этом нет необходимости `$?` .

Смотрите также:

- Коды выхода

Вывод против Возвращаемое значение

Важно помнить о различных способах запуска дочерней команды и о том, нужны ли вам выходные данные, возвращаемое значение или ни то, ни другое.

Когда вы хотите запустить команду (или конвейер) и сохранить (или распечатать) **результат**, будь то строка или массив, вы используете `$(command)` синтаксис Bash:

```
$(ls -l / tmp)
newvariable=$(printf "foo")
```

Если вы хотите использовать **возвращаемое значение** команды, просто используйте команду или add ( ) для запуска команды или конвейера в подоболочке:

```
если grep someuser /etc/passwd ; затем
 # сделай что-нибудь
, фи

если ( с grep someuser | grep sqlplus ); затем
 # кто-то из пользователей вошел в систему и запустил sqlplus
fi
```

Убедитесь, что вы используете форму, которую вы намеревались:

```
# НЕПРАВИЛЬНО!
если $(ОШИБКА grep / var/log/messages); тогда
 # отправлять оповещения
fi
```

Пожалуйста, посмотрите:

- Составные команды Bash
- Замена команд
- Группировка команд в подоболочке

Обсуждение

У.Ликерт, 2015/09/24 19:37 ()

Реагирование на коды выхода

Вы можете использовать конкретное значение и сделать гораздо больше, заключив группу команд в { } . (и все же 1-строка не скрывает остальную часть кода)

Обратите внимание на ';' после последней команды, **необходимо**

```
grep ^root: /etc/passwd >/dev/null 2> &1 || { rc= $?; echo "Э
тот поиск вернул мне '$ rc'. Может быть, пора в паб."; return
$ rc; }
```

scripting/newbie_traps.txt Последнее редактирование: 2020/05/28 12:34 автор: fgrouse

Bash Hackers Wiki