

Разговор о переносимости

Язык программирования сценариев BASH основан на синтаксисе оболочки Bourne с некоторыми расширениями и производными.

Если скрипты должны быть переносимыми, следует избегать некоторых синтаксических элементов, специфичных для BASH. Для всех сценариев следует избегать других, например, если имеется соответствующий синтаксис, совместимый с POSIX® (см. Устаревший и устаревший синтаксис).

Некоторые синтаксические элементы имеют специфичный для BASH и переносимый ¹⁾ кулон. В этих случаях следует предпочесть переносимый синтаксис.


построить	портативный эквивалент	Описание	Переносим
source FILE	. FILE	включить файл сценария	Оболочка Bourne (bash, ksh, POSIX zsh, ...)
declare ключевое слово	typeset ключевое слово	определение локальных переменных (или переменных со специальными атрибутами)	ksh, zsh, ... POSIX!
command <<< WORD	command <<MARKER WORD MARKER	строка here, специальная форма документа here, избегайте ее в переносимых скриптах!	POSIX®
export VAR=VALUE	VAR=VALUE export VAR	Хотя POSIX® позволяет это, некоторые оболочки не хотят присваивать и экспортировать в одной команде	POSIX®, zsh, ksh, ...

((МATH)) построить	: \$((МATH)) портативный эквивалент	POSIX® не Описание определяет команду арифметического вычисления, многие оболочки этого не знают. Использование псевдокоманды : и арифметического расширения \$(()) - это своего рода обходной путь. Внимание: не все оболочки поддерживают назначение, подобное \$((a = 1 + 1)) ! Также см. Ниже для, вероятно, более переносимого решения.	все оболоч совместим POSIX® Переноси
[[EXPRESSION]] [EXPRESSION] или test EXPRESSION		Ключевое слово Bashish test зарезервировано POSIX®, но не определено. Используйте старомодный способ с test помощью команды. См. классическую тестовую команду	POSIX® и ,
COMMAND < <(... INPUTCOMMANDS > TEMPFILE INPUTCOMMANDS...) COMMAND < TEMPFILE		Замена процесса (здесь используется с перенаправлением); используйте старомодный способ (временные файлы)	POSIX® и ,

построить	портативный эквивалент	Описание	Переносим
<code>((echo X);(echo Y))</code>	<code>((echo X); (echo Y))</code>	Вложенные подоболочки (отделите внутренние <code>()</code> от внешних <code>()</code> пробелами, чтобы не путать оболочку с операторами арифметического управления)	POSIX® и

Обоснование переносимости

Вот некоторая информация о переносимости. Примите это как небольшое руководство, чтобы сделать ваши скрипты немного более переносимыми. Оно неполное (и никогда не будет полным!) и не очень подробное (например, вы не найдете информации о том, какая оболочка технически разветвляется от какой подоболочки). Это всего лишь небольшой набор рекомендаций по переносимости. - *Thebonsai*

 **Fix Me!** Гуру оболочки UNIX, пожалуйста, будьте терпеливы с таким новичком, как я, и давайте комментарии и подсказки вместо пламени.

Переменные среды (экспортируемые)

Когда **существующей переменной среды** присваивается новое значение, есть две возможности:

Новое значение просматривается последующими программами

- без каких-либо специальных действий (например, Bash)
- только после явного экспорта с `export VARIABLE` помощью (например, Sun's /bin/sh)

Поскольку дополнительный `export` не повредит, самый безопасный и переносимый способ - всегда (повторно) экспортировать измененную переменную, если вы хотите, чтобы ее видели последующие процессы.

Арифметика

В Bash есть специальная составная команда для выполнения арифметических операций без расширения. Однако в POSIX такой команды нет. В таблице вверху : `$((MATH))` конструкция упоминается как возможная альтернатива. Что касается кода выхода, 100% эквивалентная конструкция будет:

```
# Bash (или другие) составная команда
if ((МАТЕМАТИКА)); затем
...

# переносимая эквивалентная команда
, если [ "$ ((МАТН))" - не 0 ]; тогда
...
```

Кавычки вокруг арифметического расширения `$((МАТН))` не должны быть обязательными в соответствии с POSIX, но Bash и AT & T-KSH выполняют разделение слов при арифметических расширениях, поэтому наиболее переносимыми являются *кавычки*.

команда echo

Общая проблема `echo` заключается в том, что существует 2 (может быть, больше) основных варианта. Единственный случай, когда вы можете безопасно использовать `an echo` во всех системах, это: повторение непереносимых аргументов, которые не начинаются с `-` (тире) и не содержат `\` (обратной косой черты).

Почему? (список известных поведений)

- может или не может автоматически интерпретировать коды `escrаре` с обратной косой чертой в строках
- может или не может автоматически интерпретировать переключатели (например `-n`)
- может игнорировать или не игнорировать тег "end of options" (`--`)
- `echo -n` и `echo -e` не являются ни переносимыми, ни стандартными (**даже в пределах одной оболочки**, в зависимости от версии или переменных среды или параметров сборки, особенно KSH93 и Bash)

По этим и, возможно, другим причинам POSIX (SUS) стандартизировал существование команды `"printf"`.

Расширения параметров

- `${var:x:x}` специфичен ли KSH93 / Bash
- `${var/.../...}` и `${var//.../...}` специфичны ли KSH93 / Bash
- `var=$*` и `var=$@` не обрабатываются одинаково во всех оболочках, если первый символ IFS не является " " (пробел). `var="$*"` должно работать (за исключением того, что оболочка Bourne всегда соединяет расширения с пробелом)

Специальные переменные

PWD

PWD - это POSIX, но не Bourne. Большинство оболочек *не являются POSIX* в том смысле, что они не игнорируют значение переменной PWD среды. Обходной путь для исправления значения PWD в начале вашего скрипта:

```
pwd -P > dev/ null
```

Случайный

RANDOM - это переменная, специфичная для Bash / KSH / ZSH, которая даст вам случайное число до 32767 ($2^{15}-1$). Среди многих других доступных внешних опций вы можете использовать awk для генерации случайного числа. Существует несколько реализаций awk, и от того, какую версию использует ваша система, будет зависеть. Большинство современных систем будут вызывать "gawk" (то есть GNU awk) или "nawk". 'oawk' (т.е. Оригинальный / Старый awk) не имеет функций rand() или srand(), поэтому их лучше избегать.

```
# 'gawk' может генерировать случайные числа с помощью srand(). В это
# м примере 10 целых чисел от 1 до 500:
randpm=$(gawk -v min=1 -v max=500 -v nNum=10 'BEGIN { srand(systime()
+ PROCINFO["pid"]); для (i = 0; i < nNum; ++i) {print int(min + rand
() *(макс - мин)) } }')

# 'nawk' и 'mawk' делают то же самое, но для их функции rand() требуе
# тся начальное значение. В этом примере мы используем $(дата)
randpm=$(mawk -v min= 1 -v max= 500 -v nNum = 10 -v seed="$(дата +%Y%
M% d%H%M%S)" 'BEGIN { srand(начальное значение); для (i = 0; i < nNu
m; ++i) {print int(min + rand() * (max - min)) } }')
```

Да, я не awk эксперт, поэтому, пожалуйста, исправьте это, а не жалуйтесь на возможный глупый код!

```
# Ну, видя, как это // есть // BASH-hackers.org Я вроде как пропустил
bash-способ сделать вышеописанное ;- )
# выведите число от 0 до 500 :- )
printf $(( 500 * СЛУЧАЙНЫЙ / 32767 ))

# Или выведите 30 случайных чисел от 0 до 10 ;)
X=0; в то время как (( X ++ < 30 )); сделать echo $(( 10 * RANDOM /
32767 )); готово
```

СЕКУНДЫ

СЕКУНДЫ зависят от KSH / ZSH / Bash. Избегайте этого. Найдите другой метод.

Проверьте наличие команды в PATH

Переменная PATH представляет собой список имен каталогов, разделенных двоеточием, поэтому в принципе возможно запустить цикл и проверить каждый PATH компонент на наличие команды, которую вы ищете, и на возможность выполнения.

Однако этот метод выглядит не очень хорошо. Есть и другие способы сделать это, используя команды, которые *не имеют прямого* отношения к этой задаче.

ХЭШ

hash Команда используется для того, чтобы заставить оболочку сохранять полный путь к команде в справочной таблице (чтобы избежать повторного сканирования PATH при каждой попытке выполнения команды). Поскольку он должен выполнять PATH поиск, его можно использовать для этой проверки.

Например, чтобы проверить, доступна ли команда `ls` в месте, доступном PATH :

```
если хэш ls> /dev/null 2> &1;, то
echo "ls доступен"
fi
```

Что-то вроде массовой проверки:

```
для имени в ls grep sed awk; сделай
, если ! хэш "$name"> / dev / null 2> & 1; затем
повторите "СБОЙ: отсутствует команда '$name'"
выход 1
fi
выполнен
```

Здесь (bash 3) hash также учитываются встроенные команды. Я не знаю, работает ли это везде, но это кажется логичным.

команда

command Команда используется для явного вызова внешней команды, а не встроенной с тем же именем. Именно по этой причине он должен выполнять PATH поиск и может использоваться для этой проверки.

Например, чтобы проверить, доступна ли команда `sed` в месте, доступном PATH :

```
если command -v sed>/dev/null 2> &1;, то
echo "sed доступен"
fi
```

¹⁾ "переносимый" не обязательно означает, что это POSIX, это также может означать, что он "широко используется и принят" и, следовательно, может быть более переносимым, чем POSIX®

Обсуждение

Этот сайт поддерживается Performing Databases - вашими экспертами по администрированию баз данных

Bash Hackers Wiki



За исключением случаев, когда указано иное, контент на этой вики лицензируется по следующей лицензии:
Лицензия на бесплатную документацию GNU 1.3