Лучшие практики написания bash-скриптов

Скрипт — это небольшая программа. Bash оперирует практически теми же концепциями, что и языки программирования. У нас есть циклы, условный оператор, переменные, функции.

В любой программе критично важна читаемость кода. Инженеры эксплуатации практически всегда работают в командах, современные системы нередко слишком велики и слишком сложны для одного человека.

Когда мы работаем командой, важно иметь общие инструменты, общие наработки, чтобы мы могли не изобретать велосипед, а воспользоваться трудом коллеги для решения определённой задачи.

Чтобы скриптом могли воспользоваться другие люди, он должен быть удобен для чтения, чтобы найти в нём ошибку или чтобы добавить в него какие-то новые функции.

Что же повышает читаемость скрипта?
Во-первых, комментарии. Строка, содержащая комментарий, должна начинаться с символа решётки (хеш). Мы можем добавлять краткие пояснения, что делает та или иная конструкция; при помощи символов комментария временно отключать какие-то фрагменты скрипта (и добавлять пояснения, почему это было сделано). Ещё комментарием иногда пишут todo, вещи, которые нужно добавить или исправить.
Во-вторых, понятнее скрипт делают отступы. Они не несут функциональной нагрузки.

В случае с bash в принципе оба способа записи будут работать:

```
if [ $num -eq 0 ]
then
    echo "true"
else
    echo "false"
fi

if [ $num -eq 0 ]
then
echo "true"
else
echo "false"
fi
```

Если добавить вложенный условный оператор, разница станет гораздо нагляднее.

Если сперва мы хорошо видим каждый блок кода, то во втором они сливаются.

```
if [ "$username" = Anna ]
then
    if [ "$password" = 123 ]
  then
echo "Welcome!"
        else
                        echo "Wrong password"
fi
else
    echo "Wrong username"
fi

if [ "$username" = Anna ]
then
if [ "$password" = 123 ]
then
echo "Welcome!"
else
echo "Wrong password"
fi
else
echo "Wrong username"
fi
```

В-третьих, важно именование переменных. Вот вещи, которых лучше избегать:

```
read -p "Username: " $a
read -p "Password: " $b
if [ "$a" = Anna ]
then
    if [ "$b" = 123 ]
  then
echo "Welcome!"
        else
                        echo "Wrong password"
fi
else
    echo "Wrong username"
fi
```

Не используйте:

- имена переменных, состоящие из одной буквы;
- акронимы и сокращения (исключением можно считать какие-то очень распространённые аббревиатуры — LED, CD, HDMA и так далее);
- в качестве имён переменных произвольные слова.

Если в скрипте вам требуется сохранить все запущенные на данный момент процессы, файл лучше не называть таким образом:

```
cat =  process_list.txt
harvest = process_list.txt
darthvader = process_list.txt
```

Лучше назвать его, например, файлом вывода.

```
output_file = report.txt
```

Чтобы не обнаружилась неочевидная конструкция наподобие:

```
if [ -f $darthvader ]
then
  ...
```

И не пришлось перечитывать скрипт с начала.

Это же относится к именам функций. Они должны быть читаемыми и отражать назначение функции. Конструкция подобного вида может изрядно озадачить:

```
badguy $haha $lol
```

Функции сами по себе значительно повышают понятность скрипта, и упорядочивают код, и позволяют переиспользовать единожды написанный код.

Ещё один приём, который повышает читаемость скрипта, — вынос объявления важных переменных в начало.

Кроме объявления переменных, важно обращать внимание и на их использование в дальнейшем коде. Не забывайте заключать переменные в кавычки, особенно это важно, если в переменные вы записываете имена файлов.

Отладка скриптов

Например, у нас есть такой скрипт:

```
n=0
while [ $n -lt 10 ]
do
echo "Hello world"
done
```

Цикл становится бесконечным, никакой ошибки он нам не возвращает.

Нужно, чтобы скрипт выводил каждый шаг своих действий.
Сделать мы это можем двумя способами.
Во-первых, при запуске скрипта передать необходимый ключ bash:

```
bash -x script1.sh
```

Или включить эту настройку в самом скрипте:

```
set -x
```

Теперь мы легко сможем увидеть, какие значения записывались в переменную n и поймём, что мы забыли о том, что увеличивать счётчик нужно в каждой итерации.

```
n=$(($n+1))
```

Действительно, так всё отлично работает.