Препарировать плохой лайнер

```
*ls  $ .zip | пока  я читаю, я делаю  j=`echo $i|sed's/.zip//g"; mkdi
r  $ j;  cd  $ j;  распаковать ../$ i;   cd ..; готово
```

Это настоящий однострочник, о котором кто-то спрашивал `#bash` . **В нем есть несколько недостатков. Давайте разберем это!**

```
* ls  $ .zip | пока  я читаю; делаю ...; готово
```

(Пожалуйста, прочитайте http://mywiki .wooledge.org/ParsingLs (http://mywiki.wooledge.org/ParsingLs) .) Эта команда выполняется `ls` при расширении `*.zip` . Предполагая, что в текущем каталоге есть имена файлов, которые заканчиваются на '.zip', ls предоставит понятный для человека список этих имен. Вывод ls не предназначен для синтаксического анализа. Но, как в sh, так и в bash, мы можем безопасно перебирать сам глобус:

```
для $ i в  *.zip; do  j=`echo $i | sed' s/.zip//g"; mkdir  $j; cd  $j
; распаковать ../$i; cd ..; готово
```

Давайте разберем это еще немного!

```
j= `echo  $ i |  sed' s/.zip//g" # где $i - это имя, заканчивающееся
 на '.zip'
```

Цель здесь, похоже, получить имя файла без его `.zip` расширения. На самом деле, для этого есть команда, совместимая с POSIX®: `basename` Реализация здесь неоптимальна по нескольким причинам, но единственное, что действительно подвержено ошибкам, это " `echo $i` ". Повторение переменной без *кавычек* означает, что будет происходить разделение слов, поэтому любые пробелы в `$i` будут по существу нормализованы. Для `sh` достижения цели необходимо использовать внешнюю команду и подоболочку, но мы можем исключить канал (подоболочки, внешние команды и каналы несут дополнительные накладные расходы при запуске, поэтому они могут действительно снизить производительность в цикле). Просто для хорошей оценки, давайте использовать более читаемую, современную `$()` конструкцию вместо подсказок старого стиля:

```
sh $ для i в  *.zip; do  j=$(базовое  имя "$i"  ".zip"); mkdir  $j; c
d  $j; распаковать ../$ i; cd ..; готово
```

В Bash нам не нужна подоболочка или внешняя команда basename . См . раздел Удаление подстроки с расширением параметра:

```
bash $ для i в  * .zip; сделать  j="$ {i%.zip }"; mkdir $j; cd $j; pa
спаковать ../$i; cd ..; готово
```

Давайте продолжим:

```
$ j mkdir $ ; cd  $ j; ...; cd ..
```

Как программист, вы **никогда** не знаете, в какой ситуации будет работать ваша программа. Даже если вы это сделаете, следующая лучшая практика никогда не помешает: когда следующая команда зависит от успеха предыдущей команды (команд), проверьте успех! Вы можете сделать это с `&&` помощью соединения "", таким образом, если предыдущая команда завершится неудачей, bash не будет пытаться выполнить следующие команды. Это полностью POSIX®. О, и помните, что я говорил о разделении слов на предыдущем шаге? Ну, если вы не будете цитировать `$j` , разделение слов может произойти снова.

```
&& " $ j" cd &&  " $ j"mkdir  $ ... && cd ..
```

Это почти верно, но есть одна проблема – что произойдет, если `$j` содержит косую черту? Затем `cd ..` не вернется в исходный каталог. Это неправильно! `cd -` приводит к возврату компакт-диска в предыдущий рабочий каталог, так что это гораздо лучший выбор:

```
&& " $ j"cd &&  " $ j"mkdir  $ ... && cd -
```

(Если вам пришло в голову, что я забыл проверить успех после cd -, хорошая работа! Вы могли бы сделать это с `{ cd - || break; }` , но я собираюсь оставить это, потому что это многословно, и я думаю, что, вероятно, мы сможем вернуться к нашему исходному рабочему каталогу без проблем.)

Итак, теперь у нас есть:

```
sh $ для i в  *.zip; сделать  j=$(базовое  имя "$i"  ".zip"); mkdir "
$j" && cd "$ j" && распаковать ../$ i && cd -; готово
```

```
bash $ для i в  * .zip; сделать  j="$ {i%.zip }"; mkdir "$j" && cd "
$j" && распаковать ../$i && cd -; готово
```

Давайте добавим `unzip` команду обратно в микс:

```
распаковать  &&"$ j " cd && "$ j " mkdir ../$i && cd -
```

Ну, кроме разделения слов, в этом нет ничего ужасно плохого. Тем не менее, вам не приходило в голову, что unzip уже может быть нацелен на каталог? Для команды нет стандарта `unzip` , но все реализации, которые я видел, могут выполнять это с помощью флага -d. Таким образом, мы можем полностью отказаться от команд cd:

```
$ mkdir "$ j" && распаковать -d "$j" "$ i"
```

```
sh $ для i в  * .zip; do  j=$(базовое  имя "$i"  ".zip"); mkdir "$j"
&& unzip -d "$ j" "$ i"; готово
```

```
bash $ для i в *.zip; do  j="${i%.zip }"; mkdir "$j" && распаковать -
d "$j" "$i"; готово
```

Вот! Это настолько хорошо, насколько это возможно.

Обсуждение

Михаил Шигорин (http://www.altlinux.org), 2012/01/11 10:55 ()

Спасибо, хорошее пошаговое руководство :) Я бы только подчеркнул правильное цитирование немного больше, не вводя никаких расширений переменных без кавычек самостоятельно.

Eduardo Bustamante (http://dualbus.com/), 2012/06/18 01:04 ()

I'd like to address some issues I noticed in the dissection.

First, in the section that states: ``but the only thing that's genuinely error-prone with this is "echo $i". . I think ``sed 's/.zipg''' *is also error prone. Let me explain it. The dot is a RE meta-character, which matches *anything*. So, it will match things like azip, bzip & czip. Also, if the goal is to strip the extension, it will require some anchoring (i.e. s/\.zip$)*; or else, it will remove more than just the extension. If that anchor is used, there's no need for the `g' flag. I'm not stating that using sed is the way to go; I'm merely remarking on its usage in that one-liner. Also, echo has multiple incompatible implementations, and using it to print an arbitrary string is risky, since that string can take the form of an option (-e or -n for example). There's no way to avoid this, like using `–', since echo will just print it. Its replacements are printf (printf '%s\n' "$i") or using the here-string syntax in bash («< "$i"). The next thing to note is in the differentiation between sh and bash regarding the ${i%.zip} expansion. The ${name%foo} expansion is standardized in POSIX, so it's safe to use it in sh also. And it's clearly simpler, since you can do just ${i%.*}. And the last thing. It can be made to work with other casings of .zip, like .Zip, .ZIP, and all the possible permutations, using *.[Zz] [Ii][Pp] as the pattern, or just using shopt -s nocaseglob.