

Re: SHELLdorado Newsletter 1/2005 – April 30th, 2005

Source: <http://unix.derkeiler.com/Newsgroups/comp.unix.shell/2005-05/0091.html>

From: Stephane CHAZELAS (*this.address_at_is.invalid*)

Date: 05/02/05

Date: Mon, 2 May 2005 19:55:00 +0100

2005-04-30, 17:43(+02), Heiner Steven:

> *SHELLdorado Newsletter 1/2005 – April 30th, 2005*

You may prefer to cross post, instead of multi posting (to cus and cuq).

[...]

```
> file=/etc/motd
> for line in `cat $file` # WRONG
> do
> echo "$line"
> done
>
> ...is no solution, because the variable "line" will in turn
> contain each (whitespace-delimited) *word* of the file, not
> each line.
```

Not to speak of filename generation.

```
> The "while" command is a better candidate for
> this job:
>
> file=/etc/motd
> while read line
> do
> echo "$line"
> done < "$file"
>
> Note that the "read" command automatically processes its
> input: it removes leading whitespace from each line, and
> concatenates a line ending with "\n" with the one following.
> The following commands suppress this behaviour:
>
> file=/etc/motd
> OIFS=$IFS; IFS= # Change input field separator
> while read -r line
```

```
> do
> echo "$line"
> done < "$file"
> IFS=$OIFS # Restore old value
```

This doesn't restore the previous value if IFS was previously unset.

Also note that commands in the loop have their stdin affected.

As other pointed out:

```
while IFS= read -r line <&3; do
  printf '%s\n' "$line" # echo should be banished
done 3< "$file"
```

```
> There still is one disadvantage to this loop: it's slow. If
> the processing consists of string manipulations, consider
> replacing the loop completely e.g. with an AWK script.
>
> Portability:
> "read -r" is available with ksh, ksh93, bash, zsh,
> POSIX, but not with older Bourne Shells (sh).
```

I'd rather say "but not with older sh's (as the Bourne shell)".
Actually, the Bourne Shell from SVR4.2 supports read -r (according to Sven Mascheck, but I've never come across such a shell). ash doesn't support -r but newer so called POSIX conformant shells based on it do (such as dash or newer BSD shells).

Note that in the Bourne Shell, IFS= read line, sets IFS not only for the duration of "read", but also in the Bourne shell, redirected loops are run in a subshell (use exec 3< "\$file" to prevent that).

For shells that don't support read -r:

```
sed 's/\&&/g' < "$file" | while IFS= read line
```

But I think all that is in the Unix FAQ.

```
>
-----
> >> Shell Tip: Print a line from a file given its line number
>
-----
[...]
```

```
> lineno=5
> sed -n "${lineno}{p;q;}"
```

Or sed "\$lineno!d;q"

>

> >> *Shell Tip: How to convert upper–case file names to lower–case*

>

>

> *Admit it: you sometimes copy files from an operating system
> with a name ending in *indows. A frequent annoyance are file
> names IN ALL UPPER CASE.*

The obvious solution is to use dedicated tools such as zsh's zmv:

```
autoload -U zmv # in ~/.zshrc
zmv '*[[:upper:]]*' '${(L)f}' # rename all files
```

```
zmv '^*[[:lower:]]*' '${(L)f}' # rename only the all ucase ones
```

>

> *The following command renames them to contain only lower
> case characters:*

>

```
> for file in *
> do
> lcase=`echo "$file" | tr '[A-Z]' '[a-z]`
```

```
printf '%s\n' "$file" | ...
```

>

```
> # Does the target file exist already? Do not
> # overwrite it:
> [ -f "$lcase" ] && continue
```

"[-f" is for exists *and is regular file*, why not renaming the other types of files?

>

```
> # Are old and new name different?
> [ x"$file" = x"$lcase" ] && continue # no change
>
> mv "$file" "$lcase"
```

```
mv -- "$file...
```

[...]

```
> because the former code may need many "gzip" processes for a
> task the latter command accomplishes with only one external
> process. But how could we build a command line like the one
> above when the input files come from a file, or even
> standard input? A naive approach could be
```

>
> *gzip `cat textfiles.list archivefiles.list`*
>
> *but this command can easily run into an "Argument list too*
> *long" error, and doesn't work with file names containing*
> *embedded whitespace characters.*

That depends how what value you gave to IFS.

With
IFS=
'; set -f

There shouldn't be problems with filenames with blanks (there should be with filenames with newlines though).

> *A better solution is using*
> *"xargs":*

Which works only if you use quoting properly (and various implementations of xargs understand different quotings), it doesn't work if textfiles.list contains one filename per line (unless filenames are known not to contain any blanks or quotes or backslashes).

zsh's xargs would be a better solution. Recent versions of ksh93 have command -x.

[...]
> *cat textfiles.list archivefiles.list | xargs gzip*
>
> *The "xargs" command reads its input line by line, and build*
> *a command line by appending each line to its arguments*
> *(here: "gzip"). Therefore the input*

No, not /each line/.

Also note the problem fixed with xargs -r in some xargs implementations (gzip called even if the file list is empty).

>

> >> *Shell Tip: How to avoid "Argument list too long" errors*
>

>
> *Oh no, there it is again: the system's spool directory is*
> *almost full (4018 files); old files need to be removed, and*
> *all useful commands only print the dreaded "Argument list*
> *too long":*
>

```
> $ cd /var/spool/data
> $ ls *
> ls: Argument list too long
> $ rm *
> rm: Argument list too long
```

```
ls /* rm /*
or
ls -- * rm -- *
```

```
[...]
> $ echo *
> file1 file2 [...]
>
> $ for file in *; do rm "$file"; done # be careful!
```

```
rm -- "$file"
```

```
>
> o Use "xargs"
>
> $ ls | xargs rm # careful!
>
> $ find . -type f -size +100000 -print | xargs ...
>
> o Limit the number of arguments for a command:
>
> $ ls [a-l]*
> $ ls [m-z]*
```

and zsh's zargs:

```
autoload -U zargs # in ~/.zshrc
zargs /* -- rm
```

```
or ksh93 command -x:
command -x rm /*
```

Or use zsh's builtin rm:

```
zmodload zsh/files
rm /*
```

```
--
Stéphane
```