10 year anniversary

**Subscribe now**

X

Get the highlights in your inbox every week.

Your email...

Your location...

Subscribe

[Privacy Statement](#)

**LOG IN**

Articles Resources Downloads About

Open Organization

# Ma... ed with shell

16 Jan 2017 | Seth Kenlon (Red Hat) (/users/seth) | 513 | 8 comments

old AT&T training video (https://youtu.be/XvDZLjaCJuw). In the video, Brian W. Kernighan (the "K" in **awk**) and Lorinda L. Cherry (co-author of **bc**) demonstrate how one of the founding principles of UNIX was to empower

es to create complex and customized tools.

s://youtu.be/tc4ROCJYbm0): "Think of the ly as [...] building blocks with which you can pipe-lining is the fundamental contribution of e a bunch of programs...and stick them data flows from the one on the left to the one elf looks after all the connections. The w anything about the connection; as far as alking to the terminal."

day users the ability to program.

The POSIX operating system is, figuratively, an API for itself. If you can figure out how to complete a task in a POSIX shell, then you can automate that task. That's programming, and the main vehicle for this everyday POSIX programming method is the shell script.

True to its name, the shell *script* is a line-by-line recipe for what you want your computer to do, just as you would have done it manually.

Because a shell script consists of common, everyday commands, familiarity with a UNIX or Linux (generically known as **POSIX**) shell is helpful. The more you practice using the shell, the easier it becomes to formulate new scripts. It's like learning a foreign language: the more vocabulary you internalize, the easier it is to form complex sentences.

and probably others. In a few sections, I do provide some bash-specific examples, but the final script abandons those, so you can either switch to bash for the lesson about setting variables, or do some simple syntax

t.org/unix-shells).

ıse **bash**. It's a good shell with lots of friendly ux, Cygwin, WSL, Mac, and an option on

llo world script from a terminal window. Mind ıd double have different effects.

```
                                sh
$ echo "echo 'hello world' " >> hello.sh
```

As you can see, writing a shell script consists, with the exception of the first line, of echoing or pasting commands into a text file.

To run the script as an application:

```
$ chmod +x hello.sh
$ ./hello.sh
hello world
```

And that's, more or less, all there is to it!

Now let's tackle something a little more useful.

If there's one thing that confuses the computer and human interaction, it's spaces in file names. You've seen it on the internet: URLs like **http://example.com/omg%2ccutest%20cat%20photo%21%211.jpg**. Or

ı up when running a simple command:

```
such file or directory
o such file or directory
```

space with a backslash, or quotation marks:

⸺now, but it gets inconvenient, so why not write a script to remove those annoying spaces from file names?

Create a file to hold the script, starting with a "shebang" (**#!**) to let your system know that the file should run in a shell:

```
$ echo '#!/bin/sh' > despace
```

Good code starts with documentation. Defining the purpose lets us know what to aim for. Here's a good README:

```
despace is a shell script for removing spaces from file names.

Usage:
$ despace "foo bar.txt"
```

Assuming you have a file called "foo bar.txt" in an otherwise empty directory, try this:

ınd output. In this case, the input has been a
y. The output is what you would expect: the
/.

the input of another command through a
ɔsite side of the pipe acts as a sort of filter.
ıigned especially to modify strings passed
**--delete** option to delete a character defined
ın quotes.

```
$ ls "foo bar.txt" | tr --delete ' '
foobar.txt
```

Now you have just the output you need.

In the BASH shell, you can store output as a **variable**. Think of a variable as an empty box into which you place information for storage:

```
$ NAME=foo
```

When you need the information back, you can look in the box by referencing a variable name preceded by a dollar sign (**$**).

To get the output of your despacing command and set it aside for later, use a variable. To place the *results* of a command into a variable, use backticks:

```
r -d ' '`
```

oal, you have a method to determine the
ource filename.

```
-d ' '`
```

The second part of the script must perform the renaming. You probably already now that command:

```
$ mv "foo bar.txt" foobar.txt
```

However, remember in the script that you're using a variable to hold the destination name. You do know how to reference variables:

```
#!/bin/sh

NAME=`ls "foo bar.txt" | tr -d ' '`
echo $NAME
mv "foo bar.txt" $NAME
```

whatever you use in your script).

```
$ touch "foo bar.txt"
```

y as your documentation describes. It's
/ works on a file called **foo\ bar.txt**, and

self as **$0** and to anything typed after it, sequentially, as **$1**, **$2**, **$3**, and so on. Your shell script counts as a POSIX command, so try swapping out **foo\ bar.txt** with **$1**.

```
#!/bin/sh

NAME=`ls $1 | tr -d ' '`
echo $NAME
mv $1 $NAME
```

Create a few new test files with spaces in the names:

```
$ touch "one two.txt"
$ touch "cat dog.txt"
```

```
ls: cannot access 'one': No such file or directory
ls: cannot access 'two.txt': No such file or directory
```

ıs such; everything is working as designed, . Your script is "expanding" the **$1** variable to and along with that comes that bothersome .

ıble in quotations the same way you wrap

```
echo $NAME
mv "$1" $NAME
```

Another test or two:

```
$ ./despace "one two.txt"
onetwo.txt
$ ./despace c*g.txt
catdog.txt
```

This script acts the same as any other POSIX command. You can use it in conjunction with other commands just as you would expect to be able to use any POSIX utility. You can use it in conjunction with a command:

```
$ for FILE in ~/test1/* ; do /path/to/despace $FILE ; done
```

and so on.

l, but technically it could be optimized, and it
vements.

lly not needed. The shell can calculate the
go.

operations. The same way you, in math,
s first, the shell resolves statements in
efore executing a command. Therefore, the
statement:

```
$ mv foo\ bar.txt `ls foo\ bar.txt | tr -d ' '`
```

gets transformed into:

```
$ mv foo\ bar.txt foobar.txt
```

and then the actual **mv** command is performed, leaving us with just **foobar.txt**.

Knowing this, you can condense the shell script into:

That looks disappointingly simple. You might think that reducing it to a one-liner makes the script unnecessary, but shell scripts don't have to have lots of lines to be useful. A script that saves typing even a simple command can

;, which is especially important when moving

e improvement. Additional testing reveals a running **despace** with no argument renders

```
h file or directory

 operand after ''
ormation.
```

These errors are confusing because they're for **ls** and **mv**, but as far as users know, it wasn't **ls** or **mv**, but **despace**, that they ran.

If you think about it, this little script shouldn't even attempt to rename a file if it didn't get a file as part of the command in the first place, so try using what you know about variables along with the **test** function.

## If and test

The **if** statement is what turns your little despace utility from a script into a program. This is serious code territory, but don't worry, it's also pretty easy to understand and use.

have to do is define for the computer what needs to be true or false and what to do as a result.

ue or False is the **test** utility. You don't call it this in a terminal:

```
yes, true, affirmative"; fi

 "yes, true, affirmative"; fi
```

e all manner of shorthand to choose from, option, which detects if the length of a string lea translates in your despace script as:

```
#!/bin/sh

if [ -z "$1" ]; then
   echo "Provide a \"file name\", using quotes to nullify the space."
   exit 1
fi

mv "$1" `ls "$1" | tr -d ' '`
```

The **if** statement is broken into separate lines for readability, but the concept remains: if the data inside the **$1** variable is empty (zero characters are present), then print an error statement.

Try it:

```
$ ./despace
```

Success!

Well, actually it was a failure, but it was a *pretty* failure, and more

is is a way for POSIX applications to send an
ncountered an error. This capability is
ther people who may want to use despace in
succeeding in order for everything else to

something to protect the user from
eally, you'd pass this option through to the
r the sake of simplicity, you'll hard code it.
permission before overwriting a file that
already exists:

```
#!/bin/sh

if [ -z "$1" ]; then
   echo "Provide a \"file name\", using quotes to nullify the space."
   exit 1
fi

mv -i "$1" `ls "$1" | tr -d ' '`
```

Now your shell script is helpful, useful, and friendly—and you're a programmer, so don't stop now. Learn new commands, use them in your terminal, take note of what you do, and then script it. Eventually, you'll put yourself out of a job, and the rest of your life will be spent relaxing while

Topics : **Getting started** (/tags/getting-started)

**Subscribe now**

Get the highlights in your inbox every week.

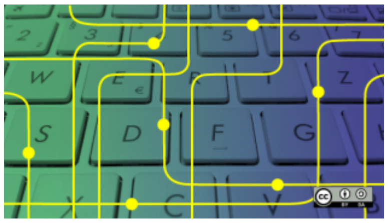
X

uthor

Kenlon is an independent multimedia artist, free culture geek. He has worked in the film ).com/name/nm1244992) and computing dhat.com/skenlon) industry, often at the same time. ntainers of the Slackware-based multimedia production rmedia.info (http://slackermedia.info)

ısers/seth)

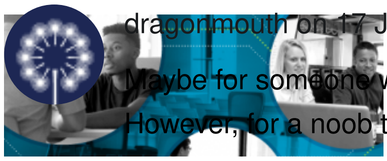
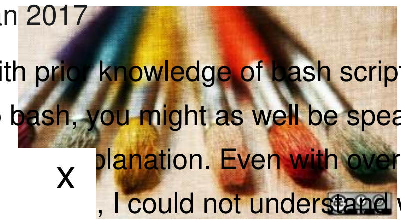
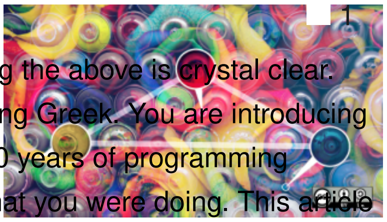
## Recommended reading



**Learn how to program in Python by building a simple dice game** (/article/17/10/python-101?utm_campaign=intrel)

**8 tips to help non-techies move to Linux** (/article/18/12/help-non-techies?utm_campaign=intrel)

**Did your first pull request get accepted?** (/article/18/10/pull-request-poll?utm_campaign=intr

dragonmouth on 17 Jan 2017 1

Maybe for someone with prior knowledge of bash scripting the above is crystal clear. However, for a noob to bash, you might as well be speaking Greek. You are introducing ... planation. Even with over 30 years of programming ... , I could not understand what you were doing. This article ... e to get started in shell scripting.

**Subscribe now** X

Get the highlights in your inbox every week.

**introduction to ... cape for absolute ... eginners** (/article ... 18/1/inkscape-absolute-... ners?utm_campaign=intrel)

**Getting started with ImageMagick** (/article /17/8/imagemagick?utm

... seth) on 17 Jan 2017 1

... s to do. If you're keen to learn, open up a terminal and ... sson. It's not meant as a copy/paste solution for anything, ... the code samples. If you do that, I think you'll find yourself ... lessons.

... . I've used this lesson in classes with great success, but ... styles differ. Luckily, there are many great shell scripting ... iding a great one right here on opensource.com by Wicked Cool Shell Scripts author Dave Taylor: https://opensource.com/article/16/12/calcshell-interactive-linux-command... (https://opensource.com/article/16/12/calcshell-interactive-linux-command-line-calculator)

JJ (/users/wavesailor) on 27 Jan 2017 1

Thanks Set as I'm about to write my first Bash script.

Seth Kenlon (/users/seth) on 29 Jan 2017 0

Good luck! There are several great resources online, of course. Just remember to test your script in a safe environment *before* using it on real data that actually matters. Trust me.

X

1

Awesome article, thanks!

I will just add a bit more stuff, in case anyone wants to know more.

**Subscribe now**

X

Get the highlights in your inbox every week.

ace will only work if it is in your current directory. For
ing the full pathname.

g more, check out more articles here or on tldp.org.

seth) on 29 Jan 2017 0

referencing tldp.org in an article I wrote! Why? because
of the shell scripting I know from tldp.org! I agree, it's a
ting tips.

peacecop kalmer: on 02 Mar 2017 0

./despace: 2: ./despace: NAME: not found

mv: 'foo bar.txt' järel puudub sihtfail
Lisainfo saamiseks proovige 'mv --help'.

Seth Kenlon (/users/seth) on 05 Mar 2017 0

This could be down to a few problems. Hard to diagnose (and probably not terribly efficient to try) here in the comments section. I suggest you open an account on http://linuxquestions.org (http://linuxquestions.org), if you haven't already got one, and paste in the contents of your shell script as it is now, and the exact command you are trying. We'll try to debug over there.

Subscribe to our weekly newsletter

**Subscribe now**

X

Get the highlights in your inbox every week.

address...

ıtry or region

Subscribe

hts in your inbox every week.

Find us:

Privacy Policy | Terms of Use | Contact | Meet the Team | Visit opensource.org