embedding short python scripts inside a bash script

45

28

I'd like to embed the text of short python scripts inside of a bash script, for use in say, my `.bash_profile`. What's the best way to go about doing such a thing?

The solution I have so far is to call the python interpreter with the `-c` option, and tell the interpreter to `exec` whatever it reads from `stdin`. From there, I can build simple tools like the following, allowing me to process text for use in my interactive prompt:

```
function pyexec() {
    echo "$(/usr/bin/python -c 'import sys; exec sys.stdin.read()')"
}

function traildirs() {
    pyexec <<END
trail=int('${1:-3}')
import os
home = os.path.abspath(os.environ['HOME'])
cwd = os.environ['PWD']
if cwd.startswith(home):
    cwd = cwd.replace(home, '~', 1)
parts = cwd.split('/')
joined = os.path.join(*parts[-trail:])
if len(parts) <= trail and not joined.startswith('~'):
    joined = '/'+joined
print joined
END
}

export PS1="\h [\$(traildirs 2)] % "
```

This approach smells slightly funny to me though, and I'm wondering what alternatives to doing it this way might be.

My bash scripting skills are pretty rudimentary, so I'm particularly interested to hear if I'm doing something silly from the bash interpreter's perspective.

python bash

can you say more clearly what is it you are actually trying to do? from what i see , Python is not really needed. you can do most things with the shell. – ghostdog74 Feb 3 '10 at 1:55

1 @ghostdog74: nothing really deeper than I was saying; I'm just a much better python programmer than a bash programmer, and IMO python is more powerful, in general, than bash. It might be handy to implement functionality used in a bash script in python, and sometimes not depend on external files when doing so. I'm finally making the switch from tcsh to bash (after 15 years), and I'm trying to bend the shell to my will/preferences. – Matt Anderson Feb 3 '10 at 3:15

-1: Why not simply create a .py module file? Why force the Python into a shell script when a better solution is (usually) to stop using the shell entirely? – S.Lott Feb 3 '10 at 3:19

2 @S.Lott: In my case, I am running bash scripts as Alfred commands but need the split functionality of Python. Creating a .py file adds unnecessary overhead. – Chris Redford Mar 25 '13 at 21:24

29

The python interpreter accepts `-` on the command line as a synonym for `stdin` so you can replace the calls to pyexec with:

```
python - <<END
```

See command line reference here.

1 Huh -- I had done a text search on the python man page, and the text `stdin` isn't in there. Looking more closely though, "standard input" is. Doh. – Matt Anderson Feb 3 '10 at 1:54

Agreed the heredoc approach can work. There are things to be careful about however: (1) if the heredoc marker (`END` in Ned's example) occurs in column 0 in the Python script, the heredoc will end early; (2) variations on heredoc marker (with or without leading `-`) affect whether tabs and spaces are preserved; and (3) syntactic overlap between Python and bash when variable expansion is turned on (heredoc marker not quoted), e.g. `${1}` could appear legitimately in a Python string format specifier, but would be replaced with a bash command line parameter if the heredoc marker isn't quoted. – Chris Johnson Mar 24 '14 at 3:09

1 Be careful how you name the heredoc marker. When it's unquoted as in your example, shell expansion will occur within the heredoc string. For example, if your python code consisted of just `print '$SHELL'` , the output would be `/bin/bash` or whatever your shell happens to be. If you change the first line to `python - <<'END'` , the output would be `$SHELL` . If you are copying and pasting existing code to embed into a bash script, you almost certainly don't want shell substitutions -- you want the code to run the same way it does when it's not embedded in a bash script, right? – Chris Johnson Aug 17 '15 at 19:01

A full example would be nice. – kev Jan 19 '18 at 7:14

39

Why should you need to use `-c` ? This works for me:

```
python << END
... code ...
END
```

without needing anything extra.

2 To collect output into a variable, I did `ABC=$(python << END` ... `END)` and it worked. – Evgeni Sergeev Feb 6 '14 at 5:53

What if we want to communicate to python a string parameter coming from bash script? – Augustin Riedinger Feb 4 '15 at 22:24

Then you need the `-` , as in Ned's answer. The arguments would follow the dash. – Ricardo Cárdenes Feb 4 '15 at 23:41

7 Be careful how you name the heredoc marker. When it's unquoted as in your example, shell expansion will occur within the heredoc string. For example, if your python code consisted of just `print '$SHELL'` , the output would be `/bin/bash` or whatever your shell happens to be. If you change the first line to `python - <<'END'` , the output would be `$SHELL` . If you are copying and pasting existing code to embed into a bash script, you almost certainly don't want shell substitutions -- you want the code to run the same way it does when it's not embedded in a bash script, right? – Chris Johnson Aug 17 '15 at 19:02

10

One problem with using bash `here document` is that the script is then passed to Python on `stdin` , so if you want to use the Python script as a filter, it becomes unwieldy. One alternative is to use the `bash` 's `process substitution` , something like this:

```
... | python <( echo '
code here
' ) | ...
```

If the script it too long, you could also use `here document` inside the paren, like this:

```
... | python <(
cat << "END"
code here
END
 ) | ...
```

Inside the script, you can read/write as you normally would from/to standard i/o (e.g., `sys.stdin.readlines` to gobble up all the input).

Also, `python -c` can be used as mentioned in other answers, but here is how I like to do it to format nicely, while still respecting Python's indentation rules (credits):

```
read -r -d '' script <<-"EOF"
    code goes here prefixed by hard tab
EOF
python -c $script
```

Just make sure that the first character of each line inside here document is a hard tab. If you have to put this inside a function, then I use the below trick that I saw somewhere to make it look aligned:

```
function somefunc() {
    read -r -d '' script <<-"----EOF"
        code goes here prefixed by hard tab
----EOF
    python -c $script
}
```

2 Why 0 upvotes? It it the only correct answer here, if you want to use sys.stdin in your program! – Igor Chubin Jun 16 '16 at 11:22

1 This is the best solution, upvote it! – user1171968 Jun 26 '17 at 16:47

This is by far the best solution to date! I would also add that you probably want to quote the END in the second example: `cat <<'END'` to avoid having to escape $ chars and other special chars in the HERE document. – Michael Goldshteyn Aug 23 '18 at 14:14

@MichaelGoldshteyn Done! – haridsv Aug 23 '18 at 16:11

7

Sometimes it is not a good idea to use here document. Another alternative is to use python -c:

```
py_script="
import xml.etree.cElementTree as ET,sys
...
"

python -c "$py_script" arg1 arg2 ...
```

Nice and simple, and lets you move the python code away from the middle of the rest of the bash code. Also lets your python code read from stdin. – dbort Jan 13 '16 at 1:35

7

If you need to use the output of the python in the bash script you can do something like this:

```
#!/bin/bash
```

```
ASDF="it didn't work"

ASDF=`python <<END
ASDF = 'it worked'
print ASDF
END`

echo $ASDF
```

1 Be careful how you name the heredoc marker. When it's unquoted as in your example, shell expansion will occur within the heredoc string. For example, if your python code consisted of just `print '$SHELL'`, the output would be `/bin/bash` or whatever your shell happens to be. If you change the first line to `python -<<'END'`, the output would be `$SHELL`. If you are copying and pasting existing code to embed into a bash script, you almost certainly don't want shell substitutions -- you want the code to run the same way it does when it's not embedded in a bash script, right? – Chris Johnson Aug 17 '15 at 19:02

You are right. Thank you Chris! – desgua Aug 22 '16 at 11:59

5

Ready to copy-paste example with input:

```
input="hello"
output=`python <<END
print "$input world";
END`

echo $output
```

2

Interesting... I want an answer now too ;-)

He is not asking how to execute python code within a bash script but actually have the python set environment variables.

Put this in a bash script and try to get it to say "it worked".

```
export ASDF="it didn't work"

python <<END
import os
os.environ['ASDF'] = 'it worked'
END

echo $ASDF
```

The problem is that the python gets executed in a copy of the environment. Any changes to that environment aren't seen after python exits.

If there is a solution for this, I'd love to see it too.

1 That isn't really what I was getting at necessarily, and I'm pretty sure you can't get environment variables from a child process (directly) or alter a parent's environment variables (directly). You *can* however `eval` the output of the python script in the bash script, thus executing arbitrary statements, setting environment variables or doing anything else you might like. – Matt Anderson Feb 3 '10 at 3:12

1 @Eric: If you really want an answer, post your own question. – Ned Deily Feb 3 '10 at 3:29

Matt, sorry... I briefly looked at your python script and saw you were doing stuff with os.environ and assumed that was the kind of thing you were doing. My mistake. – eric.frederich Feb 3 '10 at 3:44

You can manage it by combining a bash variable and the python output. See my answer bellow. – desgua Dec 24 '13 at 2:25

1

Try this :

```
#!/bin/bash
a="test"
python -c "print  '$a this is a test'.title()"
```

0

If you are using zsh you can embed Python inside the script using zsh's join and python stdin option (`python -`):

```
py_cmd=${(j:\n:)"${(f)$(
    # You can also add comments, as long as you balance quotes
    <<<'if var == "quoted text":'
    <<<'  print "great!"')}"}

catch_output=$(echo $py_cmd | python -)
```

You can indent the Python snippet, so it looks nicer than the `EOF` or `<<END` solutions when inside functions.