How to implement common bash idioms in Python? [closed]

Asked 16 years, 2 months ago Modified 6 years, 8 months ago

Viewed 116k times

242

votes

Closed. This question is opinion-based. It is not currently accepting answers.

Closed 6 years ago.

Locked. This question and its answers are locked because the question is off-topic but has historical significance. It is not currently accepting new answers or interactions.

I currently do my textfile manipulation through a bunch of badly remembered AWK, sed, Bash and a tiny bit of Perl.

I've seen mentioned a few places that python is good for this kind of thing. How can I use Python to replace shell scripting, AWK, sed and friends?

python bash shell

Share

edited Apr 13, 2018 at 23:58

asked Oct 16, 2008 at 17:11

Chris Jefferson

7,157 ● 11 ● 46 ● 69

3 pythonpy is a good competitor for awk and sed using python syntax: github.com/Russell91/pythonpy – RussellStewart Sep 13, 2014 at 5:57

4 you can use shellpy that was designed with an idea in mind to replace bash/sh with python github.com/lamerman/shellpy – Alexander Ponomarev Feb 14, 2016 at 15:29

This is my question, I don't understand why it is opinion based. The top answer lists each of the main things a shell does, and tells you how to do them in python. That in my opinion answers the question in a non-opinion way. – Chris Jefferson Apr 9, 2018 at 10:03

This question, and it's closure, are being discussed on meta here – Erik A Apr 18, 2018 at 10:40

Comments disabled on deleted / locked posts / reviews

17 Answers

Sorted by: Highest score (default)

144

votes

Any shell has several sets of features.

- The Essential Linux/Unix commands. All of these are available through the subprocess library. This isn't always the best first choice for doing *all* external commands. Look also at shutil for some commands that are separate Linux commands, but you could probably implement directly in your Python scripts.

Another huge batch of Linux commands are in the os library; you can do these more simply in Python.

And -- bonus! -- more quickly. Each separate Linux command in the shell (with a few exceptions) forks a subprocess. By using Python `shutil` and `os` modules, you don't fork a subprocess.

- The shell environment features. This includes stuff that sets a command's environment (current directory and environment variables and what-not). You can easily manage this from Python directly.
- The shell programming features. This is all the process status code checking, the various logic commands (if, while, for, etc.) the test command and all of it's relatives. The function definition stuff. This is all much, much easier in Python. This is one of the huge victories in getting rid of bash and doing it in Python.
- Interaction features. This includes command history and what-not. You don't need this for writing shell scripts. This is only for human interaction, and not for script-writing.
- The shell file management features. This includes redirection and pipelines. This is trickier. Much of this can be done with subprocess. But some things that are easy in the shell are unpleasant in Python. Specifically stuff like `(a | b; c ) | something >result`. This runs two processes in parallel (with output of `a` as input to `b`), followed by a third process. The output from that sequence is run in

parallel with `something` and the output is collected into a file named `result`. That's just complex to express in any other language.

Specific programs (awk, sed, grep, etc.) can often be rewritten as Python modules. Don't go overboard. Replace what you need and evolve your "grep" module. Don't start out writing a Python module that replaces "grep".

The best thing is that you can do this in steps.

1. Replace AWK and PERL with Python. Leave everything else alone.
2. Look at replacing GREP with Python. This can be a bit more complex, but your version of GREP can be tailored to your processing needs.
3. Look at replacing FIND with Python loops that use `os.walk`. This is a big win because you don't spawn as many processes.
4. Look at replacing common shell logic (loops, decisions, etc.) with Python scripts.

Share

edited Jun 20, 2020 at 9:12

Community Bot

1 ● 1

answered Oct 16, 2008 at 17:41

S.Lott

391k ● 82 ● 517 ● 788

6 wrote: "Interaction features. This includes command history and what-not. You don't need this." I'm afraid no one could tell what a person really needs or not. Perhaps he does. Besides, these facilities make a lot of sense in an interactive shell, taking as an example the difference between Idle and IPython. – heltonbiker Mar 21, 2011 at 16:44

47 I sincerely wish people would ditch shell scripting altogether. I understand that hacking is practically a religion in the *nix world but I get really tired of trying to interpret all the hackish workarounds implanted in the OS. The novelty of microtools (awk, sed, top, base, etc) wore off the day everybody decided to roll their own version. I cringe when I imagine the amount of man-hours wasted on crappy little tools that could easy be replaced by a couple well designed Python modules. ::sigh:: – Evan Plaice Feb 21, 2012 at 10:07

40 I disagree @EvanPlaice because the python version of several `find` scripts I have is ugly and long and unmaintainable in comparison. Many things *should be* shell scripts, many others *should not*. Not everything needs to be only one of Python or BASH (or anything else). – mikebabcock Oct 2, 2012 at 4:04

8 @mikebabcock Ideally there would be a complete library that implements all of the micro-tools that are made available by the basic *nix stack. Functions like find() and last() would be included and instead of pipes, a combination of currying and lazy-loading would handle gluing it all together. Wouldn't it be nice to have a POSIX scripting environment that works in a standard way across all distros? Nothing like that exists, yet... – Evan Plaice Oct 3, 2012 at 16:59

2 The point about shell pipelines (such as `(a | b; c ) | something >result`) is somewhat mitigated by it being trivially easy to pass shell pipelines to `subprocess` methods using `shell=True` – iruvar May 24, 2013 at 1:17

103

votes

Yes, of course :)

Take a look at these libraries which help you ***Never write shell scripts again*** (Plumbum's motto).

- Plumbum
- Sarge
- sh

Also, if you want to replace awk, sed and grep with something Python based then I recommend pyp -

> "The Pyed Piper", or pyp, is a linux command line text manipulation tool similar to awk or sed, but which uses standard python string and list methods as well as custom functions evolved to generate fast results in an intense production environment.

Share

edited Mar 10, 2013 at 10:55

answered Oct 16, 2012 at 13:37

Piotr Dobrogost

42.3k ● 45 ● 241 ● 373

Also have a look at Envoy, which is an alternative to sh github.com/kennethreitz/envoy – AllanLRH Jun 10, 2015 at 13:37

57 votes

I just discovered how to combine the best parts of bash and ipython. Up to now this seems more comfortable to me than using subprocess and so on. You can easily copy big parts of existing bash scripts and e.g. add error handling in the python way :) And here is my result:

```
#!/usr/bin/env ipython3

# *** How to have the most comfort scripting experience
# ######################################################
#
# … by using ipython for scripting combined with subcomm
#
# 1. echo "#!/usr/bin/env ipython3" > scriptname.ipy
#
# 2. chmod +x scriptname.ipy
#
# 3. starting with line 2, write normal python or do som
#    the ! magic of ipython, so that you can use unix cc
#    within python and even assign their output to a var
#    var = !cmd1 | cmd2 | cmd3
#
# 4. run via ./scriptname.ipy - if it fails with recogni
#    but parses raw python fine, please check again for

# ugly example, please go and find more in the wild
files = !ls *.* | grep "y"
for file in files:
  !echo $file | grep "p"
# sorry for this nonsense example ;)
```

See IPython docs on system shell commands and using it as a system shell.

Share

edited May 22, 2017 at 10:15

anatoly techtonik
20.5k ● 13 ● 130 ● 144

answered Mar 29, 2013 at 22:49

TheK
571 ● 4 ● 2

11 Upvoted because for some bizarre reason, no-one else has mentioned !-commands in IPython, which are absolutely key; especially since you can also assign their output to a variable (list of lines) as in `filelines = ! cat myfile` – kampu May 24, 2013 at 1:04

And you can use python variables as `$var` in a shell command? Wow. This should be the accepted answer. – chtenb Mar 17, 2016 at 12:08

And you can also use it from within the jupyter notebooks – Yuval Atzmon Mar 9, 2017 at 10:01

44

votes

As of 2015 and Python 3.4's release, there's now a reasonably complete user-interactive shell available at: http://xon.sh/ or https://github.com/scopatz/xonsh

The demonstration video does not show pipes being used, but they ARE supported when in the default shell mode.

Xonsh ('conch') tries very hard to emulate bash, so things you've already gained muscle memory for, like

```
env | uniq | sort -r | grep PATH
```

or

```
my-web-server 2>&1 | my-log-sorter
```

will still work fine.

The tutorial is quite lengthy and seems to cover a significant amount of the functionality someone would generally expect at a ash or bash prompt:

- Compiles, Evaluates, & Executes!
- Command History and Tab Completion
- Help & Superhelp with `?` & `??`
- Aliases & Customized Prompts
- Executes Commands and/or `*.xsh` Scripts which can also be imported
- Environment Variables including Lookup with `${}`
- Input/Output Redirection and Combining
- Background Jobs & Job Control
- Nesting Subprocesses, Pipes, and Coprocesses
- Subprocess-mode when a command exists, Python-mode otherwise
- Captured Subprocess with `$()`, Uncaptured Subprocess with `$[]`, Python Evaluation with `@()`

- Filename Globbing with `*` or Regular Expression Filename Globbing with Backticks

Share

edited Mar 21, 2018 at 14:37

Michael come lately
9,273 ● 7 ● 66 ● 92

answered Jun 3, 2015 at 10:08

Kamilion
545 ● 4 ● 8

But why does it seem like all of these answers are just reinventing the wheel *for people who don't know bash*? I've gotten moderately comfortable with bash and every one of these answers looks like it will end up being more work for little benefit. These answers are all aimed at python people who are afraid of (or don't want to spend time learning) bash, am I right? – Buttle Butkus Jul 13, 2017 at 7:34

Seems it has some disadvantages like requirement to use `.xsh` extension for files with the xonsh code: github.com/xonsh/xonsh/issues/2478. Otherwise you have to use `evalx` to call it directly from the `.py` files. – Andry Aug 14, 2017 at 11:52

31

votes

- If you want to use Python as a shell, why not have a look at IPython ? It is also good to learn interactively the language.
- If you do a lot of text manipulation, and if you use Vim as a text editor, you can also directly write plugins for Vim in python. just type ":help python" in Vim and follow the instructions or have a look at this

presentation. It is so easy and powerfull to write functions that you will use directly in your editor!

Share

edited Oct 31, 2011 at 17:15

Catskul
19k ● 16 ● 88 ● 117

answered Oct 16, 2008 at 18:16

Mapad
8,527 ● 6 ● 44 ● 40

8 there's an ipython profile called 'sh' that makes the interpreter *very* much like a shell. – Autoplectic Feb 24, 2009 at 20:22

3 The ipython 'sh' profile has been removed for some time now. – gdw2 May 20, 2013 at 19:40

>>>result = !dmesg | grep -i 'usb' #the ! operator does it all – Permafacture Nov 9, 2013 at 17:14

16 votes

In the beginning there was sh, sed, and awk (and find, and grep, and...). It was good. But awk can be an odd little beast and hard to remember if you don't use it often. Then the great camel created Perl. Perl was a system administrator's dream. It was like shell scripting on steroids. Text processing, including regular expressions were just part of the language. Then it got ugly... People tried to make big applications with Perl. Now, don't get me wrong, Perl can be an application, but it can (can!) look like a mess if you're not really careful. Then there is all this flat data business. It's enough to drive a programmer nuts.

Enter Python, Ruby, et al. These are really very good general purpose languages. They support text processing, and do it well (though perhaps not as tightly entwined in the basic core of the language). But they also scale up very well, and still have nice looking code at the end of the day. They also have developed pretty hefty communities with plenty of libraries for most anything.

Now, much of the negativeness towards Perl is a matter of opinion, and certainly some people can write very clean Perl, but with this many people complaining about it being too easy to create obfuscated code, you know some grain of truth is there. The question really becomes then, are you ever going to use this language for more than simple bash script replacements. If not, learn some more Perl.. it is absolutely fantastic for that. If, on the other hand, you want a language that will grow with you as you want to do more, may I suggest Python or Ruby.

Either way, good luck!

Share

answered Oct 16, 2008 at 20:58

MattG

1,322 ● 10 ● 7

9

votes

I suggest the awesome online book *Dive Into Python*. It's how I learned the language originally.

Beyond teaching you the basic structure of the language, and a whole lot of useful data structures, it has a good

chapter on file handling and subsequent chapters on regular expressions and more.

Share

edited Mar 19, 2018 at 21:09

answered Oct 16, 2008 at 17:40

Dan Lenski

79.6k ● 13 ● 84 ● 127

1 ... main issue of link-only answers. – Jean-François Fabre ♦ Mar 26, 2018 at 19:58

---

7 votes

Adding to previous answers: check the pexpect module for dealing with interactive commands (adduser, passwd etc.)

Share

answered Oct 16, 2008 at 22:05

Federico A. Ramponi

47k ● 30 ● 111 ● 133

---

7 votes

One reason I love Python is that it is much better standardized than the POSIX tools. I have to double and triple check that each bit is compatible with other operating systems. A program written on a Linux system might not work the same on a BSD system of OSX. With Python, I just have to check that the target system has a sufficiently modern version of Python.

Even better, a program written in standard Python will even run on Windows!

Share

answered May 24, 2013 at 1:23

Hal Canary
2,240 ● 18 ● 17

1 "a program written in standard Python will even run on Windows": no kidding? – Jean-François Fabre ♦ Mar 26, 2018 at 20:07

6 votes

I will give here my opinion based on experience:

For shell:

- shell can very easily spawn read-only code. Write it and when you come back to it, you will never figure out what you did again. It's very easy to accomplish this.
- shell can do A LOT of text processing, splitting, etc in one line with pipes.
- it is the best glue language when it comes to integrate the call of programs in different programming languages.

For python:

- if you want portability to windows included, use python.
- python can be better when you must manipulate just more than text, such as collections of numbers. For this, I recommend python.

I usually choose bash for most of the things, but when I have something that must cross windows boundaries, I just use python.

Share

edited Dec 1, 2013 at 15:00

answered Dec 1, 2013 at 14:40

Germán Diago

7,664 ● 1 ● 38 ● 66

---

4 votes

[pythonpy](#) is a tool that provides easy access to many of the features from awk and sed, but using python syntax:

```
$ echo me2 | py -x 're.sub("me", "you", x)'
you2
```

Share

edited Mar 27, 2015 at 20:17

answered Sep 13, 2014 at 5:55

RussellStewart

5,413 ● 3 ● 28 ● 25

---

3 votes

I have built semi-long shell scripts (300-500 lines) and Python code which does similar functionality. When many external commands are being executed, I find the shell is easier to use. Perl is also a good option when there is lots of text manipulation.

Share

answered Oct 16, 2008 at 21:49

Mike Davis

1,441 ● 1 ● 8 ● 4

3

votes

While researching this topic, I found [this proof-of-concept code] (via a comment at http://jlebar.com/2010/2/1/Replacing_Bash.html) that lets you "write shell-like pipelines in Python using a terse syntax, and leveraging existing system tools where they make sense":

```
for line in sh("cat /tmp/junk2") | cut(d=',',f=1) | 'sor
    sys.stdout.write(line)
```

Share

answered Oct 3, 2010 at 20:21

Nickolay

31.9k ● 13 ● 110 ● 194

2

votes

Your best bet is a tool that is specifically geared towards your problem. If it's processing text files, then Sed, Awk and Perl are the top contenders. Python is a general-purpose *dynamic* language. As with any general purpose language, there's support for file-manipulation, but that isn't what it's core purpose is. I would consider Python or Ruby if I had a requirement for a dynamic language in particular.

In short, learn Sed and Awk really well, plus all the other goodies that come with your flavour of *nix (All the Bash

built-ins, grep, tr and so forth). If it's text file processing you're interested in, you're already using the right stuff.

Share

answered Oct 16, 2008 at 18:14

Eric Smith
5,382 ● 2 ● 35 ● 51

2

votes

You can use python instead of bash with the ShellPy library.

Here is an example that downloads avatar of Python user from Github:

```
import json
import os
import tempfile

# get the api answer with curl
answer = `curl https://api.github.com/users/python
# syntactic sugar for checking returncode of executed pr
if answer:
    answer_json = json.loads(answer.stdout)
    avatar_url = answer_json['avatar_url']

    destination = os.path.join(tempfile.gettempdir(), 'p

    # execute curl once again, this time to get the imag
    result = `curl {avatar_url} > {destination}
    if result:
        # if there were no problems show the file
        p`ls -l {destination}
    else:
        print('Failed to download avatar')

    print('Avatar downloaded')
else:
    print('Failed to access github api')
```

As you can see, all expressions inside of grave accent (`) symbol are executed in shell. And in Python code, you can capture results of this execution and perform actions on it. For example:

```
log = `git log --pretty=oneline --grep='Create'
```

This line will first execute `git log --pretty=oneline --grep='Create'` in shell and then assign the result to the log variable. The result has the following properties:

stdout the whole text from stdout of the executed process

stderr the whole text from stderr of the executed process

returncode returncode of the execution

This is general overview of the library, more detailed description with examples can be found here.

Share

edited Mar 9, 2016 at 1:11

Jason

2,306 ● 2 ● 18 ● 25

answered Feb 16, 2016 at 21:23

Alexander Ponomarev

2,728 ● 5 ● 25 ● 32

1

vote

If your textfile manipulation usually is one-time, possibly done on the shell-prompt, you will not get anything better from python.

On the other hand, if you usually have to do the same (or similar) task over and over, and you have to write your scripts for doing that, then python is great - and you can easily create your own libraries (you can do that with shell scripts too, but it's more cumbersome).

A very simple example to get a feeling.

```
import popen2
stdout_text, stdin_text=popen2.popen2("your-shell-commanc
for line in stdout_text:
  if line.startswith("#"):
    pass
  else
    jobID=int(line.split(",")[0].split()[1].lstrip("<").r
    # do something with jobID
```

Check also sys and getopt module, they are the first you will need.

Share

answered Oct 16, 2008 at 17:42

Davide

17.7k ● 11 ● 55 ● 69

1

vote

I have published a package on PyPI: ez.
Use `pip install ez` to install it.

It has packed common commands in shell and nicely my lib uses basically the same syntax as shell. e.g., cp(source, destination) can handle both file and folder! (wrapper of shutil.copy shutil.copytree and it decides when to use which one). Even more nicely, it can support vectorization like R!

Another example: no os.walk, use fls(path, regex) to recursively find files and filter with regular expression and it returns a list of files with or without fullpath

Final example: you can combine them to write very simply scripts:

```
files = fls('.','py$'); cp(files, myDir)
```

Definitely check it out! It has cost me hundreds of hours to write/improve it!

Share

edited Mar 9, 2016 at 1:31

Jason

2,306 ● 2 ● 18 ● 25

answered Jan 10, 2015 at 21:14

Jerry T

1,690 ● 1 ● 20 ● 18

1 Looks interesting, but I can't break through the unformatted docs at pypi.python.org/pypi/ez , sorry... – Greg Dubicki Oct 13, 2016 at 17:40