xonsh

a superset of both shell and Python

Alexander Sosedkin

2019-05-30

xonsh’s modest official description

xonsh is a Python-powered, cross-platform,
Unix-gazing shell language and command prompt.
The language is a superset of Python 3.4+ with additional shell primitives.
xonsh (pronounced *conch*) is meant for the daily use of experts and novices alike.

`https://xon.sh`, follow the tutorial first.

Comparison table

	Bash	zsh	plumbum	fish	IPython	xonsh
Sane language			X	X	X	X
Easily scriptable	X	X	X	X		X
Native cross-platform			X	X	X	X
Meant as a shell	X	X		X		X
Tab completion	X	X		X	X	X
Man-page completion				X		X
Large standard library		X			X	X
Typed variables			X	X	X	X
Syntax highlighting				X	in notebook	w/ptk
Pun in name	X		X			X
Rich history						X

How I’d put it

	UNIX shells	xonsh	Other Python supersets
Usable as a shell	X	X	
Is Python		X	X

Overview

- Scripting
  - Take a script in shell
  - Port it to Python
  - Discuss more approaches to subprocess calling
  - Port it to xonsh
- Interactive usage
  - Marvel at the ergonomics
  - Frown at the guesswork
- Feature rundown
- Pitfalls and downsides

Scripting in shell

Example task

A researcher wants to plot Cython to Python rewrite progress.

Given a repository, create a stackplot of *source lines of code over time*, broken down by language (Python/Cython).

Tools: counting SLOC

We need a command to count SLOC at a specific revision…

```
cloc --git $commit_hash
```

```
-------------------------------------
Language  files  blank  comment  code
-------------------------------------
Python       51    771      703  2424
Cython        6    394      356  1662
Markdown      1      4        0    10
-------------------------------------
SUM:         58   1169     1059  4096
-------------------------------------
```

Tools: counting SLOC

We need a command to count SLOC at a specific revision…

```
cloc --git $commit_hash --json
```

```
{"header" : ...,
 "Python" :{
   "nFiles": 51,
   "blank": 771,
   "comment": 703,
   "code": 2424},
 "Cython" :{
   "nFiles": 6,
   "blank": 394,
   "comment": 365,
   "code": 1662},
 ...
```

Tools:

… and also one to listing the commits and their commit times.

```
git log --format='%aI %H' --since Nov | tac
```

```
2018-11-16T18:40:55+07:00 bbf6e7e3f0a9d56ebfaf0431f3c664f3eb54ff75
2018-11-16T18:41:56+07:00 2f68ff88a954678be562c92916add12f1a68f652
2018-11-16T18:42:31+07:00 5ff3b894132d58ab295d261ae89ea60964f7d94f
2018-11-25T19:19:58+07:00 fe2be0fcfa1775156bd7cde428c0044c264078bc
2018-11-25T19:22:08+07:00 146c24e14426b8c18c88c2d3c38b753f5fb120d5
2018-11-25T20:47:21+07:00 eae47992630dae23fc73d00ce774f0f98fe93f23
2018-11-26T11:45:16+07:00 93315b1e696a0224aa7397f60a1cebafd5f4569c
2018-11-25T21:56:49+07:00 550c9cbbd2c66f58e67d02cc5db349db0aa4df70
...
```

Shell script (BASH)

```
cd repo
echo -n > ../series

IFS=$'\n'
for line in $(git log --format='%cI %H' --since Nov | tac); do
    commit_date=${line%% *}
    commit_hash=${line#* }
    json=$(cloc --json --git $commit_hash)
    py_count=$(jq .Python.code//0 <<<$json)
    cy_count=$(jq .Cython.code//0 <<<$json)
    echo $commit_date $py_count $cy_count >> ../series
done
```

My feelings about it

It’s not even that bad, ${it just looks alien to a Pythonista%% *}.

Even my source highlighter gets confused on `${line#*}`.

Real shell gurus can write it better, but not much better.

Shell excels at spawning commands, and is hacky at everything else.

My feelings about it

It’s not even that bad, ${it just looks alien to a Pythonista%% *}.

Even my source highlighter gets confused on ${line#*}.

Real shell gurus can write it better, but not much better.

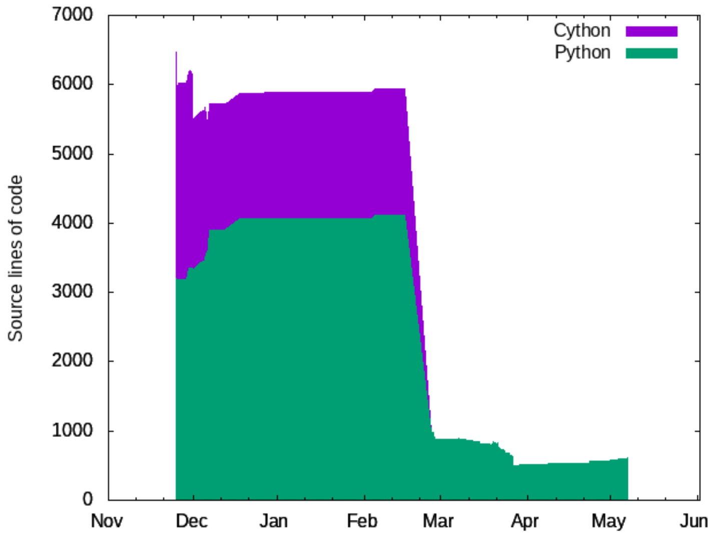
Shell excels at spawning commands, and is hacky at everything else.

It also needs some plotting code, like this.

Plotting

```
gnuplot > plot.png <<EOF

set terminal png
set xdata time
set timefmt '%Y-%m-%dT%H:%M:%S'
set format x "%b"
set tics front
set ylabel "Source lines of code"

plot "series" using 1:(\$2+\$3) with filledcurve x1 \
              t "Cython", \
     "series" using 1:2  with filledcurve x1 t "Python"
EOF
```

Plot


Cython
Python
Source lines of code
0
1000
2000
3000
4000
5000
6000
7000
Nov
Dec
Jan
Feb
Mar
Apr
May
Jun

Porting it to Python

Overall structure

```
#!/usr/bin/python

import json
import os
from datetime import datetime
from matplotlib import pyplot as plt
from matplotlib import dates as mdates

os.chdir('repo')

dates, python, cython = [], [], []
for line in {{{ THE OUTPUT OF git log ... | tac }}}:
    commit_date, commit_hash = line.split()
    jsn = json.loads( {{{ THE OUTPUT OF cloc --json }}} )
    dates.append(datetime.fromisoformat(commit_date))
    python.append(jsn['Python']['code'] if 'Python' in jsn else 0)
    cython.append(jsn['Cython']['code'] if 'Cython' in jsn else 0)

plt.stackplot(dates, python, cython,
              labels=['Python', 'Cython'])
plt.legend()
xaxis = plt.gca().xaxis
xaxis.set_major_locator(mdates.MonthLocator())
xaxis.set_major_formatter(mdates.DateFormatter('%b'))
plt.ylabel('Source lines of code')
plt.savefig('productivity_plot.png')
```

Zoom-in: imports

```
#!/usr/bin/env python

import json
import os
from datetime import datetime
from matplotlib import pyplot as plt
from matplotlib import dates as mdates
```

Zoom-in: core

```
os.chdir('repo')

dates, python, cython = [], [], []
for line in {{{ THE OUTPUT OF git log ... | tac }}}:
    commit_date, commit_hash = line.split()
    jsn = json.loads( {{{ THE OUTPUT OF cloc --json }}} )
    dates.append(datetime.fromisoformat(commit_date))
    python.append(jsn['Python']['code'] if 'Python' in jsn else 0)
    cython.append(jsn['Cython']['code'] if 'Cython' in jsn else 0)
```

Zoom-in: plotting

```
plt.stackplot(dates, python, cython,
              labels=['Python', 'Cython'])
plt.legend()
xaxis = plt.gca().xaxis
xaxis.set_major_locator(mdates.MonthLocator())
xaxis.set_major_formatter(mdates.DateFormatter('%b'))
plt.ylabel('Source lines of code')
plt.savefig('plot.png')
```

Zoom-in: core again

```
os.chdir('repo')

dates, python, cython = [], [], []
for line in {{{ THE OUTPUT OF git log ... | tac }}}:
    commit_date, commit_hash = line.split()
    jsn = json.loads( {{{ THE OUTPUT OF cloc --json }}} )
    dates.append(datetime.fromisoformat(commit_date))
    python.append(jsn['Python']['code'] if 'Python' in jsn else 0)
    cython.append(jsn['Cython']['code'] if 'Cython' in jsn else 0)
```

But how do we fill in the {{{ ... }}} gaps?

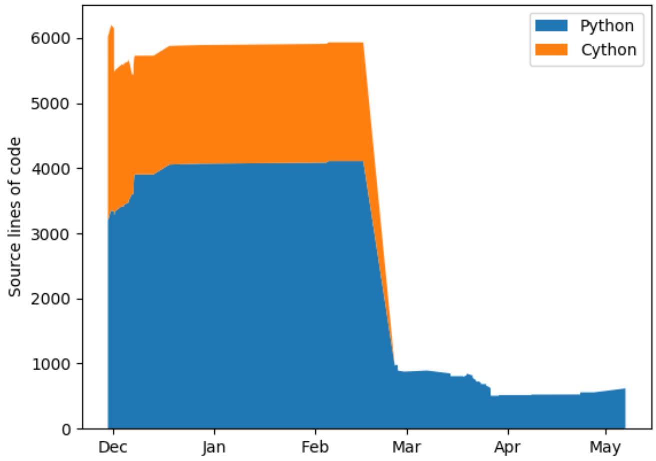
subprocess: main changes

```
os.chdir('repo')

dates, python, cython = [], [], []
git_cmd = "git log --format='%cI %H' --since Nov | tac"
for line in subprocess.check_output(git_cmd, shell=True).split(b'\n'):
    if not line:
        continue
    commit_date, commit_hash = line.decode().split()
    jsn = json.loads(subprocess.check_output(
        ['cloc', '--json', '--git', commit_hash]
    ).decode())
    dates.append(datetime.fromisoformat(commit_date))
    python.append(jsn['Python']['code'] if 'Python' in jsn else 0)
    cython.append(jsn['Cython']['code'] if 'Cython' in jsn else 0)
```

subprocess: minor gripes

```
os.chdir('repo')

dates, python, cython = [], [], []
git_cmd = "git log --format='%cI %H' --since Nov | tac"
for line in subprocess.check_output(git_cmd, shell=True).split(b'\n'):
    if not line:
        continue
    commit_date, commit_hash = line.decode().split()
    jsn = json.loads(subprocess.check_output(
        ['cloc', '--json', '--git', commit_hash]
    ).decode())
    dates.append(datetime.fromisoformat(commit_date))
    python.append(jsn['Python']['code'] if 'Python' in jsn else 0)
    cython.append(jsn['Cython']['code'] if 'Cython' in jsn else 0)
```

Plot



Python
Cython
Source lines of code
0
1000
2000
3000
4000
5000
6000
Dec
Jan
Feb
Mar
Apr
May

subprocess: shelling out

```
os.chdir('repo')

dates, python, cython = [], [], []
git_cmd = "git log --format='%cI %H' --since Nov | tac"
for line in subprocess.check_output(git_cmd, shell=True).split(b'\n'):
    if not line:
        continue
    commit_date, commit_hash = l.decode().split()
    jsn = json.loads(subprocess.check_output(
        ['cloc', '--json', '--git', commit_hash]
    ).decode())
    dates.append(datetime.fromisoformat(commit_date))
    python.append(jsn['Python']['code'] if 'Python' in jsn else 0)
    cython.append(jsn['Cython']['code'] if 'Cython' in jsn else 0)
```

Proper shell-free piping (I hope)

```
from subprocess import Popen, PIPE
p1 = Popen(["git", "log", "--format=%cI %H", "--since", "Nov"],
           stdout=PIPE)
p2 = Popen(["tac"], stdin=p1.stdout, stdout=PIPE)
p1.stdout.close()
output = p2.communicate()[0].decode()
```

Proper shell-free piping (I hope)

```
from subprocess import Popen, PIPE
p1 = Popen(["git", "log", "--format=%cI %H", "--since", "Nov"],
           stdout=PIPE)
p2 = Popen(["tac"], stdin=p1.stdout, stdout=PIPE)
p1.stdout.close()
output = p2.communicate()[0].decode()
```

That’s a long way to spell

```
git log '--format=%cI %H' --since Nov | tac
```

plumbum

Piping with plumbum

```
from plumbum.cmd import git, tac
pl = git["log", "--format=%cI %H", "--since", "Nov"] | tac
output = pl()
```

That’s shorter, but introduces a new, alien syntax.

IPython

IPython bangs and other magics

```
%cd repo

dates, python, cython = [], [], []
out = !git log '--format=%cI %H' --since Nov | tac
for line in out:
    commit_date, commit_hash = line.split()
    cloc_output = !cloc --json --git $commit_hash
    jsn = json.loads(cloc_output.n)  # joining it back with \n
    dates.append(datetime.fromisoformat(commit_date))
    python.append(jsn['Python']['code'] if 'Python' in jsn else 0)
    cython.append(jsn['Cython']['code'] if 'Cython' in jsn else 0)
```

IPython as an interactive shell

With the introduction of `%cd` and `!cmd`,
some proclaimed that Python is ready to become their login shell.

```
In [1]: !ls | wc -l
10

In [2]: 3 + 3
Out[2]: 6
```

IPython as an interactive shell

With the introduction of `%cd` and `!cmd`,
some proclaimed that Python is ready to become their login shell.

```
In [1]: !ls | wc -l
10
In [2]: 3 + 3
Out[2]: 6
```

But, trust me on that, banging that `Shift-1` before each and every command makes you wanna bang your head against something hard and switch back to bash.

It may be just an issue of ergonomics, but ergonomics matters.

Even more than all the other missing shell features.

Enter xonsh

xonsh port of our script

```
#!/usr/bin/env xonsh

import json
from datetime import datetime
from matplotlib import pyplot as plt
from matplotlib import dates as mdates

cd repo

dates, python, cython = [], [], []
for line in !(git log '--format=%cI %H' --since Nov | tac):
    commit_date, commit_hash = line.split()
    jsn = json.loads($(cloc --json --git @(commit_hash)))
    dates.append(datetime.fromisoformat(commit_date))
    python.append(jsn['Python']['code'] if 'Python' in jsn else 0)
    cython.append(jsn['Cython']['code'] if 'Cython' in jsn else 0)

plt.stackplot(dates, python, cython,
              labels=['Python', 'Cython'])
plt.legend()
xaxis = plt.gca().xaxis
xaxis.set_major_locator(mdates.MonthLocator())
xaxis.set_major_formatter(mdates.DateFormatter('%b'))
plt.ylabel('Source lines of code')
plt.savefig('productivity_plot.png')
```

Zoom-in: core

```
cd repo

dates, python, cython = [], [], []
for line in !(git log '--format=%cI %H' --since Nov | tac):
    commit_date, commit_hash = line.split()
    jsn = json.loads($(cloc --json --git @(commit_hash)))
    dates.append(datetime.fromisoformat(commit_date))
    python.append(jsn['Python']['code'] if 'Python' in jsn else 0)
    cython.append(jsn['Cython']['code'] if 'Cython' in jsn else 0)
```

No need for separate lines, powerful nesting, and look at the `cd`!

A seamless blend

```
def stashup():
    git status
    clean = !(git diff --quiet)
    git fetch
    git --no-pager diff master...origin/master
    try:
        if not clean:
            git stash
        git pull --ff-only --rebase
        git push
    finally:
        if not clean:
            git stash pop
```

Interactive xonsh usage

Look ma, no bangs!

```
$ 2 * 2
4
$ cd sample_code
$ ls -l
-rwxr-xr-x 1 monk users  946 May 25 20:14 plot_productivity.ipy
-rwxr-xr-x 1 monk users 1181 May 25 19:50 plot_productivity.py
-rwxr-xr-x 1 monk users  406 May 25 14:03 plot_productivity.sh
-rwxr-xr-x 1 monk users 1093 May 25 19:49 plot_productivity.xsh
$ import random
$ random.choice(['ipy', 'py', 'sh', 'xsh'])
'.ipy'
```

Look ma, no bangs!

```
$ 2 * 2
4
$ cd sample_code
$ ls -l
-rwxr-xr-x 1 monk users  946 May 25 20:14 plot_productivity.ipy
-rwxr-xr-x 1 monk users 1181 May 25 19:50 plot_productivity.py
-rwxr-xr-x 1 monk users  406 May 25 14:03 plot_productivity.sh
-rwxr-xr-x 1 monk users 1093 May 25 19:49 plot_productivity.xsh
$ import random
$ random.choice(['ipy', 'py', 'sh', 'xsh'])
'.ipy'
```

Preserving both muscle memory sets is extremely important.

Feature rundown

Implicit mixing

```
$ 2 * 2
4
$ cd sample_code
$ ls -l
-rwxr-xr-x 1 monk users  946 May 25 20:14 plot_productivity.ipy
-rwxr-xr-x 1 monk users 1181 May 25 19:50 plot_productivity.py
-rwxr-xr-x 1 monk users  406 May 25 14:03 plot_productivity.sh
-rwxr-xr-x 1 monk users 1093 May 25 19:49 plot_productivity.xsh
```

Implicit mixing

```
$ 2 * 2
4
$ cd sample_code
$ ls -l
-rwxr-xr-x 1 monk users  946 May 25 20:14 plot_productivity.ipy
-rwxr-xr-x 1 monk users 1181 May 25 19:50 plot_productivity.py
-rwxr-xr-x 1 monk users  406 May 25 14:03 plot_productivity.sh
-rwxr-xr-x 1 monk users 1093 May 25 19:49 plot_productivity.xsh
```

Wait a bit, `ls -l` is valid Python! How do I subtract `l` from `ls` then?

Implicit mixing

```
$ 2 * 2
4
$ cd sample_code
$ ls -l
-rwxr-xr-x 1 monk users  946 May 25 20:14 plot_productivity.ipy
-rwxr-xr-x 1 monk users 1181 May 25 19:50 plot_productivity.py
-rwxr-xr-x 1 monk users  406 May 25 14:03 plot_productivity.sh
-rwxr-xr-x 1 monk users 1093 May 25 19:49 plot_productivity.xsh
```

Wait a bit, `ls -l` is valid Python! How do I subtract `l` from `ls` then?

```
$ ls, l = 3, 2
$ ls -l
1
```

Explicit mixing

```
$ ext = random.choice(['ipy', 'py', 'sh', 'xsh'])
$ ext
'.ipy'
$ wc -l @('plot_productivity.' + ext)
30 plot_productivity.ipy
$ wc -l @(f'plot_productivity.{ext}')
30 plot_productivity.ipy
$ $(wc -l @(f'plot_productivity.{ext}'))
'30 plot_productivity.ipy'
$ $(wc -l @(f'plot_productivity.{ext}')).split()
['30', 'plot_productivity.ipy']
```

More ways to spawn commands

`$(command)` captures and returns output as a string

`!(command)` makes a really smart thing:

- gives access to both `.out` and `.err`
- return-code-aware (and truthy if `0`)
- iterable
- can raise exceptions with `.itercheck()`

There are also `$[]`, `![]` and even `@$[]`.

All the Python stuff…

It’s all there, up to Python 3.7.

- built-ins and imports
- control flow
- functions, named and anonymous
- classes
- decorators
- format strings
- …

… and most of the shell stuff

- no BASH control flow
- different, Python-inspired quoting rules
- pipelines and IO redirection: `|`, <, >, >>, 2>…
- job control: &, `fg`, bg…
- basic `*` globbing, and regex globbing
- aliases (which can be xonsh functions)
- environment variables (with types!): `${...}`

… and most of the shell stuff

- no BASH control flow
- different, Python-inspired quoting rules
- pipelines and IO redirection: `|`, <, >, >>, 2>…
- job control: &, `fg`, bg…
- basic `*` globbing, and regex globbing
- aliases (which can be xonsh functions)
- environment variables (with types!): `${...}`
- tab completion
- syntax highlighting
- left, right and bottom prompts
- rich history
- macros

Windows support

xonsh has first-class Windows support and
a strong following of UNIX guys stranded on Windows machines.

Pitfalls and downsides

I’m using xonsh as my main interactive shell for about 3 years.

On the Python support side, I have only no issues.

On the shell support side, I have exactly two:

- quoting works differently. better, but differently.
- `$ todo.sh add buy sugar and tea`

Pitfalls and downsides

I’m using xonsh as my main interactive shell for about 3 years.

On the Python support side, I have only no issues.

On the shell support side, I have exactly two:

- quoting works differently. better, but differently.
- `$ todo.sh add buy sugar and tea`

Other than functional aspects, three more:

- it’s slow
- the internals are complicated
- it’s in beta

Pitfalls and downsides

I’m using xonsh as my main interactive shell for about 3 years.

On the Python support side, I have only no issues.

On the shell support side, I have exactly two:

- quoting works differently. better, but differently.
- `$ todo.sh add buy sugar and tea`

Other than functional aspects, three more:

- it’s slow
- the internals are complicated
- it’s in beta

But the ability to mix Python and shell freely is unbeatable.

Try xonsh today!