The POSIX shell as a programming language

Michael Greenberg (Pomona College)

OBT 2017 — Paris, France
i love shell
shell is everywhere

• vital for managing systems
  • maintenance
  • deployment
• universal tool for sysadmins
• extremely powerful
POSIX shell

• Open Group Spec/IEEE Standard 1003.1
  • Intimately connected to POSIX
• Many implementations!
# figure out the absolute path to the script being run a bit
# non-obvious, the ${0%/*} pulls the path out of $0, cd's into the
# specified directory, then uses $PWD to figure out where that
directory lives - and all this in a subshell, so we don't affect
# $PWD
STEAMROOT="$(cd "${0%/*}" && echo $PWD)"

# Scary!
rm -rf "${STEAMROOT}"*

https://github.com/ValveSoftware/steam-for-linux/issues/3671

curl -k https://<master hostname>:8140/packages/current/install.bash | bash

https://puppetlabs.com/blog/simplified-agent-installation-puppet-enterprise-3.2
i love reasoning
hasn’t shell been ‘fixed’ already?

• scsh and shill?
  • not POSIX shells!

• tclsh
  • no formal attention, to my knowledge
  • and a bit out of date at this point
ShellCheck

- Linter for shell
- Catches bug in Steam script...
  - ...but not a trivial refactoring
NoFAQ

• Machine learning to correct console commands
  • No semantics insights
  • No guarantees
  • More about commands than about the shell

D’Antoni and Vaughn 2016
ABash

• Static analysis for number of arguments
  • Semantic understanding
  • Great start!

Mazurak and Zdancewic 2007
shell is unique

- unique evaluation model
- expansion, not evaluation, of args by default
- deploy and manage concurrency
- uniquely interactive programming model
- try before you buy
conventional evaluation

\[ e_1 \textbf{eval} v_1 \quad e_2 \textbf{eval} v_2 \quad \delta(\otimes, v_1, v_2) = v_3 \]

\[ e_1 \otimes e_2 \textbf{eval} v_3 \]
expansion by default

\[ e_1 \text{ expand } s_1 \quad e_2 \text{ expand } s_2 \]
\[ \text{unparse}(\delta(\otimes, \text{parse}(s_1), \text{parse}(s_2))) = s_3 \]
\[ e_1 \otimes e_2 \text{ eval } v_3 \]

\[ e \text{ eval } v \quad \text{unparse}(v) = s \]
\[ `e` \text{ expand } s \]
c ::= v=a ... a ... | c r
| c1|c2|c3|...|cn | c & | ( c )
| c1 && c2 | c1 || c2
| ! c | c1 ; c2 | if c1 c2 c3
| switch a ... { case a...) c } ...
| while c1 c2 | for x in a ... c
| defun v c
{ semantics

fixed behavior at compile time

read
tokenize
parse
expand
redirect
execute
wait}
expansion

- **read**
- **tokenize**
- **parse**
- **expand**
- **redirect**
- **execute**
- **wait**

---

echo ~ ➔ /Users/mgree

echo ${PWD} ➔ /Users/mgree/talks/obt

basename `pwd` ➔ obt

echo $((1+1)) ➔ 2

IFS="" ➔ [shows contents of ‘some file’]

cat `echo some file`

echo * ➔ abstract.txt posix.key some file

echo you can “” me ➔ you can me
backquoting

echo ~ ➝ /Users/mgree

echo ${PWD} ➝ /Users/mgree/talks/obt

basename `pwd` ➝ obt

echo $((1+1)) ➝ 2

IFS="" ➝ [shows contents of ‘some file’]

cat `echo some file`

echo * ➝ abstract.txt posix.key some file

echo you can “” me ➝ you can me
$ basename \`pwd\`

$ basename /Users/mgree/talks/obt

obt
essentials of the semantics
essentials of the semantics

```
$ x=${{y:=1}} ; echo $((x+=`echo 2`))
$ x=${1} ; echo $((x+=`echo 2`))
$ echo $((x+=`echo 2`))
$ echo $((x+=2))
$ echo 3
```

3

env

<table>
<thead>
<tr>
<th>PATH</th>
<th>/usr/bin:...</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>3</td>
</tr>
</tbody>
</table>

legend: expansion evaluation
what do I want to do?
support the programming model

• have script echo commands until script is just right
  • maybe running *some* commands

• set -x prints commands run…
  *but it still runs the commands!*

• can we do better?
other tools

- compile to other languages as a form of “gradual scripting”
- “cruft” inserter
  - hardens a shell script against, e.g., signals
  - uses weakest preconditions to guarantee good exit status of all commands
types!

- commands take a regular expression over args as input, produces certain patterns of system calls

- summarize sets of commands/system calls/outputs
  - e.g., this script will delete all files in ~/.foo/ except for ~/.foo/cache

- analyze curl-based installers!
design

$ ls
filename
spaces
filename with spaces
$ x="filename with spaces"
$ rm $x
rm: with: No such file or directory
$ ls
filename with spaces
filename with spaces
$ rm "$x"
$ ls
what else?

• theoretical ideas/angles i’m missing?

• suppose we’ve got a great model… what else should we do with it?

thanks to:
  Arjun Guha for early chats
  Calvin Aylward and Austin Blatt