Beyond Bash

Shell scripting in a statically-typed, object-oriented language

Li Haoyi
haoyi.sg@gmail.com

Abstract

The Bash shell has been a staple of software engineers for decades, and contains many anachronisms that are out of place in 2015. And yet it perseveres: no other scripting language such as Python or Ruby comes close to replacing Bash as the primary command-line, while others which improve upon it such as Zsh or Fish are conservative and incremental. We present a radically different approach to command-line scripting, using Scala as a host language for a concise, type-safe command-line DSL. This DSL, known as Ammonite, considerably improves upon doing shell-scripting in traditional scripting languages with conciseness that approaches that of Bash, while maintaining a level of safety far beyond Bash and its derivatives.

Categories and Subject Descriptors D.4.9 [**Operating Systems**]: Systems Programs and Utilities – Command and control languages

General Terms Languages

Keywords *bash;shell;scala;languages*

1. Bash's Problems

A large portion of Bash's problems arise from a small number of attributes:

1. Ambiguous parsing & escaping rules
2. Limited mechanisms for composition
3. Widespread use of implicit, global state
4. Use of Strings as the primary datatype

To take the first point as an example, let's look at a common problem: finding and removing a set of files

```
find ~ -name test3* -ok rm {}\;
```

The odd suffix at the end of the line

```
\;
```

Illustrates the problem Bash has with escaping. It is basically impossible to deduce the fact that you need the trailing semicolon here which you do not need anywhere else, and even more=so that you have to escape them! Finally, it's unclear that the block `rm {}\;` is being passed only to the `-ok` argument, and not as separate arguments to `find`. This sort of problem with unclear parsing and escaping is pervasive in the Bash shell.

SIGPLAN'05 June 12–15, 2005, Location, State, Country.

2. What about modern scripting languages?

Modern languages ameliorate some of the problems with Bash. To take Python as an example, you get:

- Straightforward parsing
- Widespread use of proper data-structures (Arrays, Dictionaries, Objects) in addition to Strings
- Good tools for keeping things neat: modules, classes, functions

All this is great. However, these languages have limitations which prevent them from being used as a serious filesystem API. For example, the added verbosity of

```
shutil.copytree(
  "src/path", "destination/%s" % dest
)
```

over

```
cp src/path destination/$dest
```

Is significant: twice as many (58 vs 29) characters. Although the core abstractions are more modular and less prone to error, this increase is unacceptable for your primary interface to the operating system.

3. Beyond Bash

In this presentation, I'll present a new approach to shell-scripting that makes use of the Scala programming language. Although this is a statically typed, object-oriented/functional language traditionally used for building compilers and distributed systems, I'll show how Scala can be used to make type-safe, intuitive operations concise enough to write by hand as part of an interactive shell:

```
val dest = 'folder
mkdir! dest
cp('src/'path, dest/'destination)
```

Mimicking the concise, piping nature of Bash

```
val lineCount =
  ls.rec! wd
    |? (_.ext == "java")
    | read.lines
    | (_.size)
    sum
```

All in a powerful, high-level language that scales to writing large applications. Together with a modified Scala REPL, I will demonstrate that this provides a credible replacement for the age-old bash-prompt that we all know.