Natural Shell: An Assistant for End-User Scripting

Xiao Liu, College of Information Sciences and Technology, Pennsylvania State University, University Park, PA, USA
Yufei Jiang, College of Information Sciences and Technology, Pennsylvania State University, University Park, PA, USA
Lawrence Wu, College of Information Sciences and Technology, Pennsylvania State University, University Park, PA, USA
Dinghao Wu, College of Information Sciences and Technology, Pennsylvania State University, University Park, PA, USA

ABSTRACT

Scripting is a widely-used way to automate the execution of tasks. Despite the popularity of scripting, it remains difficult to use for both beginners and experts: because of the cryptic commands for the first group, and incompatible syntaxes across different systems, for the latter group. The authors introduce Natural Shell, an assistant for enabling end-users to generate commands and scripts for various purposes. Natural Shell automatically synthesizes scripts for different shell systems based on natural language descriptions. By interacting with Natural Shell, new users can learn the basics of scripting languages without the obstacles from the incomprehensible syntaxes. On the other hand, the authors’ tool frees more advanced users from manuals when they switch shell systems. The authors have developed a prototype system and demonstrate its effectiveness with a benchmark of 50 examples of popular shell commands collected from online forums. In addition, the authors analyzed the usage of Natural Shell in a lab study that involves 10 participants with different scripting skill levels. Natural Shell effectively assists the users to generate commands in assigned syntaxes and greatly streamlines their learning and using experience.

KEYWORDS

Education, End-User Scripting, Natural Language Programming, Shell, WinBat

INTRODUCTION

The command line interface is essential for users and administrators, as it allows them to fully harness and customize the power of operating systems (Robbins & Beebe, 2005). Shell scripting significantly extends the utility of the command line by automating a batch of commands. However, despite the popularity that shell script enjoys, its syntax is cryptic for beginners and it requires quite a bit of practice before they can put down the manual.

Shell scripting is complicated, because of the rigid syntax and the inconsistency between shell systems. Most commands have “options”, which significantly change and extend the functions of commands. However, the use of options is excessively succinct and thus, makes the commands incomprehensible for newcomers to a particular shell. Besides, the large amount of options that each command accepts, adds more cognitive load during the learning process. Take the grep command as an
example, there are 47 options that `grep` accepts including popular ones like “-i” for case-insensitive, “-r” for recursive search, and “-w” for word only. It is difficult for novices to remember more than a few, and for experts to remember all the options. Moreover, this situation compounds when a user has to get accustomed to the syntax of some other shell as she switches to another shell system. If a Linux user wants to delete a directory in MS Windows, she may encounter difficulties, since MS Windows Command Prompt adopts `del` rather than `rm`, as the command to perform delete operations. Other superficial differences include argument position, case-sensitivity, options for commands instead of separate commands, etc. Even within the UNIX-like OS family, there are Ksh, Csh, Bash, and Zsh shells, each of which has its own features. Because of the difficulties engendered by shell scripting, there is a need for a “unified shell”.

Previous efforts have focused on whether to set up a reasoning system for UNIX Shells, or how to augment the UNIX shell with more powerful functions. Lee proposed a natural interface for shell scripting that can reason based on pre-defined cases (Lee & Lee, 1995). The reasoning system behind the interface generates scripts for seasoned users, but it overlooks the uninitiated users. Weaver et al. (Weaver & Smith, 2012) presented another UNIX text-processing tool that extends the original shell to support frequent manipulations on text files. However, their tool is platform-sensitive. We believe that a cross-platform system can handle more cases and be particularly useful for those who work across multiple operating systems.

Natural language is recognized as a more versatile method for beginners to work with programming tasks (Ballard & Biermann, 1979; Dijkstra, 1979; Brill, 1992). The way people perceive knowledge with natural language descriptions differs from that using strict and unintuitive scripting commands and their arcane outputs. Comparisons using different dimensions were discussed including simplicity, affordance, memorability, and common results, concluding that natural language has clear advantages.

In this paper, we propose a tool that interacts with end-users in natural language and transforms their descriptions into shell scripts, as shown in Table 1. The higher level of abstraction eliminates the syntactical differences between shell systems. Users can write scripts built on bare, pseudo-style logic. As such, we believe that the natural language shell makes shell scripting accessible to typical computer end-users. Our design, which we call Natural Shell is based on Weizenbaum’s ELIZA (Weizenbaum, 1966), a chatbot that uses a rule-based method to process English conversations. Natural Shell inherits the rule-based method from ELIZA but adopts regular expressions other than keywords to construct rules. In addition, a generic scripting language is created, called “Uni-Shell”, that functions as an intermediary between the natural language interpreter and the target shell commands. Compared to the alternative portages, e.g., Csh or Bash for UNIX and batch for Windows, Uni-Shell is more natural and universal. On the other hand, it is succinct enough such that frequent users may prefer it, compared to taking natural language as the input.

The whole script generation is an interactive process as can be seen in Figure 1. The system consists of three parts: ELIZA, SiE (Scripting in ELIZA) script and Shell Executor. These three processes form a pipeline. ELIZA module first processes natural languages descriptions to synthesize the commands in Uni-Shell and then responds in natural language to confirm. Then, SiE script takes in the Uni-Shell commands from the ELIZA module and synthesizes the target shell commands, which are then picked up and executed by the Shell Executor. Instead of typing with natural languages, commands in Uni-Shell are also provided for users as a second choice. Users can directly input a

<table>
<thead>
<tr>
<th>Natural Language Descriptions: Remove Directories that Contain “foo” in the Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Csh:</strong> if grep $name “foo” rm -r $name endif</td>
</tr>
<tr>
<td><strong>Bash:</strong> if grep $name “foo” rm -r $name fi</td>
</tr>
<tr>
<td><strong>Batch:</strong> if FINDSTR /r %name “foo” RD /s/q %name</td>
</tr>
</tbody>
</table>
Related Content

Developing Success Measure for Staff Portal Implementation
Dewi Rooslan Tojib (2009). Evolutionary Concepts in End User Productivity and Performance: Applications for Organizational Progress (pp. 78-94). www.igi-global.com/chapter/developing-success-measure-staff-portal/18646?camid=4v1a

An Architecture for Organizational Decision Support Systems that Utilizes a Model Coordination Subsystem
Impact of Personal Innovativeness on the Use of the Internet Among Employees at Work
[www.igi-global.com/chapter/impact-personal-innovativeness-use-internet/163887?camid=4v1a](www.igi-global.com/chapter/impact-personal-innovativeness-use-internet/163887?camid=4v1a)

Contingencies in the KMS Design: A Tentative Design Model
[www.igi-global.com/chapter/contingencies-kms-design/18647?camid=4v1a](www.igi-global.com/chapter/contingencies-kms-design/18647?camid=4v1a)