Automation and Self-Sufficiency through Scripting

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

William Lambley

Spring, 2022

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Briana Morrison, Department of Computer Science

Automation and Self-Sufficiency through Scripting

CS4991 Capstone Report, 2022
William Lambley
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
wel2vh@virginia.edu

Abstract

An accelerator facility in Virginia found its system-info database to be out-of-date, with many of its systems being out-of-service, leading to problems with accelerator measurements, unresponsive SSH requests and confusion on which systems required support. With hundreds of systems spread across the campus and no engineers with the free time to check them individually, an automated solution was required. I created this solution in the form of a user-friendly Bash script utilizing several Secure Socket Shell connections across 200+ systems. The script would check the information gained from each SSH and cross-examine it with the information present in the database in the database. This script presented several options to the user in how and where data was collected to promote its usability and convenience. The project was successful and, with some minor tweaks, was integrated into the workflow of several system engineers on site as well as the database manager, creating a much more accurate and robust systems database, allowing the appropriate out-of-service systems to be replaced. However, this script was only utilized by the Linux system management. Expanding the script to encapsulate the Windows system database could yield similarly positive results on system/database management workflow.

1. Introduction

The process of software engineering (and to an extent, all engineering) can be simplified into two steps: the assessment of a problem, and the creation of a function to solve that problem. The latter step involves taking input, running the potential solution on that input, and using the output. But what happens when the input is wrong or missing? No function can be made, so no solution can be found, leaving the assessed problem unsolved.

This is the reason accurate information is so important to any engineering process. The sheer amount of information present in enterprise-level workflows, useful or not, can also create many issues when trying to manage data for a project. For this massive quantity of information to be used efficiently, some sort of manager must be assigned, either human or (more likely) an automated script or program.

The problem I faced at the accelerator facility was an extension of this issue. With an out-of-date database for hundreds of systems across a campus that stretched over several square miles, it simply was not reasonable to have an engineer check the information of each individual system and update the database manually. Even if someone had the time to check these systems, they are updated frequently for security reasons, so any changes made would become outdated quickly. The volume of information, inaccessibility of the systems, and tedium of the work meant that an automated solution involving information-pulls from over the network made the most sense.

2. Literature Review

Many organizations have taken notice of the impending impact that automation will have on virtually every industry. Manyika, et. al. (2022) did extensive research on the topic for McKinsey & Company. Their research focuses on the impact of automation on enterprise-level

workflows and the impending consequences this will have on jobs, wages, and desired skillsets. Citing several statistics collected by the McKinsey organization, the article shows the full extent of the impact that automation could have on many industries into the future, stating that "…50% of current work activities are technically automatable" [1].

While this article focuses on the consequences of this transition, my project was more concerned with why companies desire so much automation and how it is being implemented in this case, including ways implementation could be generalized to many other uses. An integral resource that guided me through the process of creating my script was Shotts' textbook, *The Linux Command Line* (2019). The textbook is formatted to be much less formal and robotic than a traditional textbook, illustrated by the opening line, "I want to tell you a story" [2]. This formatting made it much easier to follow as a source of information and was my main source of information regarding both BASH scripting and the Linux terminal as I was starting to develop my project. Shotts features illustrated examples, in-depth but concise explanations of scripting concepts and a very pragmatic structure that is both easy to read and very good at conveying important info. While the book contains information on the entire Linux terminal, I focused on the introductory and scripting sections, which were integral to my swift completion of this project.

3. Process Design

In order to provide a deeper understanding of how automation can be used to aid in enterprise scenarios, this section will focus on the problem and solution design centered around my previous internship. The former being the structure of the database and its connected systems, and the latter being the automated script I created.

3.1 Database and System Architecture

The database that required clean-up and management was formatted as a single JSON file, with each entry being a single on-site Linux system. These systems were broken into groups based on use, location, hardware, etc. The groups were denominated by names, and systems within a given group were denominated by numbers, e.g. DEV115 for the 115th development system. Within the "block" of information for each system were lines containing the systems hardware, operating system name and version, location, and other info. The systems these blocks corresponded to had JSON files that were formatted similarly but NOT the same as the database. The files were updated automatically by startup scripts as the system's info changed over the course of its usage.

3.2 Script Design

The two main goals for the bash script I developed were to be as fast and convenient as possible. This was a response to the fact that, while the database COULD be updated without the script, no one was willing to do so because of the tedium and time requirement of such a task. The script had an intentionally simplistic design to run as quickly and efficiently as possible. First, it would load in a local version of the database supplied by the user to refer to throughout execution. Second, it would present the user with a simple ASCII menu, a recreation of which can be seen in Figure 1.

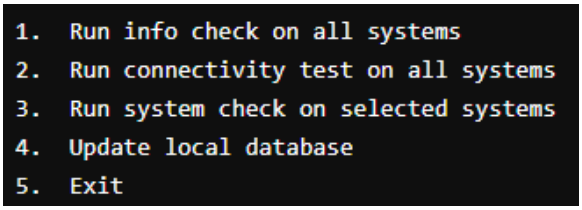

Figure 1: Simple ASCII Menu

The options presented were selected to keep the script simple but give the user as much control over the script as possible. The user would then input the number that corresponded to their desired choice. Furthermore, the script could be called with several command-line arguments that changed the behavior of the script slightly. For instance, adding "-v" to the list of arguments would list the information of the system that did not match the database if such a discrepancy was found. The first choice from the main menu would SSH into every system held in the local

database, using the name of each system to run a SSH "systemName" command, creating a secure tunnel across the internal network, as shown in Figure 2.

Figure 2: SSH Diagram [3]

Once a connection was established, the script would navigate to the directory of the JSON file held in the system and pull information out of that file as a single block of ASCII text. Once this block was obtained, the script would parse the information out of the block by accessing specific lines from the block and storing that into a local variable. Since these files were all formatted the same, the line numbers that the script took information form were hard-coded.

The local variables holding the information from the SSH were then checked against the information held in the local database in conditional statements, every time information did not match, the user was notified with a message that read, "SYSTEM_NAME's info does not match its database entry!" and at the end of execution, the user would be prompted to notify the database manager. If the user chose to do this, the script would send an email to the manager with a listing of systems that did not match their database entries and where exactly the entries differ. The second choice will be covered in the 'Challenges' section, the third choice is the same as the first, but the user inputs the name of the system (or group of systems) that they wish to check against the database.

This process is the same as when the script checks the SSH'd info against the database; but instead of an exact match, we search whether any system names in the database contain the substring entered by the user. If there is a match, then the list of systems that match the substring are listed and the user is given three choices: to edit the list by either adding or removing certain systems by name; to execute the script on the list of systems given; or to cancel. If executed, the process is the same as option one, but the script would only SSH and compare info to the systems specified by the user. The fourth option would SSH into the server that held the production database and replace the current local database with the file found on the server. The last option stops the execution of the script.

3.3 Challenges and Solutions

The most challenging part of creating this script was reducing the large number of SSH calls that resulted in a large amount of network traffic. In an earlier, more naïve implementation, each system required several SSH calls, pulling one piece of info off the machines at a time. This was very slow, requiring several minutes for the execution of the script to complete. The solution to this issue was to reduce the number of SSH calls to a single call that pulled all the required information at once, reducing network utilization to the extent that the script only took a few seconds to complete (assuming there were no hangs) when run on all systems.

The second problem was that certain systems that were not used frequently were either in hibernation, turned off or no longer connected to the network. This would cause any SSH calls to those systems to hang, freezing the script while it waited for a response that was never going to come. The first solution was to implement a simple timeout, after 30 seconds of attempting to access a system via SSH, the system is skipped. While this solution prevented hanging, it still slowed down the script considerably, as many on-site systems were asleep at any given time. So the second solution was to include an additional option, selection 2 on the main menu, to "touch" each system via SSH and immediately move on to the next. This would wake up systems to be ready to provide information to the script when it asked after this wake-up routine was finished running. The combination of these implementations sped up the script considerably, making it extremely quick, and therefore convenient, to use.

4. Results

With the completion of the project, the database manager is no longer forced to take dozens of

hours out of their schedule to individually check each system's local info and check it against the database, giving them more time to focus on other responsibilities. Now, all they must do is run the script and after a few seconds, every discrepancy present within the database will be available to them with as much or as little detail as they desire.

The upkeep of this database was enough work to warrant a part-time position but is now relegated to a simple command-line BASH script, saving the facility the costs of a part-time employee. The system will be reliable until the database or local filesystem format is changed, though even this would only require a small adjustment to the code of the script. As long as there are systems to be checked and a database to check them against, this script should be of great use to the facility and the manager of these systems.

5. Conclusion

As companies continue to amass more information, their databases will continue to bloat and their workflows will continue to slow. The solution I have outlined here is just one that has been implemented to curb this ever-expanding issue, and automation's impact on our industries and economies will only continue to grow as more solutions are put in place. While this adoption of automation will no doubt be a boon to workers, the full extent of automation's impact is yet to be seen.

6. Future Work

The next step for my project is to extend the script to handle on-site Windows machines, allowing the relevant database manager to quickly check their information. It is also possible that it could be extended beyond computer information, updating information from the accelerator itself, as well, although the process of creating such a program would be much more involved. Beyond this, however, the next logical step of automation is artificial intelligence, a computer that can not only manage information, but make decisions with that information, as well. This technology has the potential to create not only autonomous programs, but fully independent systems that do not require a human manager. As this field continues to grow and mature, more of our everyday lives will be managed by autonomous systems, allowing for a more productive society.

7. Acknowledgments

A warm thank you to my colleagues and managers at my workplace. Without them I would have not been able to teach myself what was necessary for this project. Their guidance and friendship was immensely helpful, and I will not forget what they have taught me.

References

[1] Manyika J., Lund S., Chui M., Bughin J., Woetzel J., Batra P., Ko R., Sanghvi S. 2022. Jobs Lost, jobs gained: What the future of work will mean for jobs, skills, and wages. (April 2022). Retrieved September 19, 2022 from https://www.mckinsey.com/featured-insights/future-of-work/jobs-lost-jobs-gained-what-the-future-of-work-will-mean-for-jobs-skills-and-wages

[2] Shotts, W. 2019. *The Linux Command Line: A complete introduction*, San Francisco, California: No Starch Press.

[3] Anon. *SSH Tunnel Diagram*, University of California Berkeley. Retrieved October 19, 2022 from https://security.berkeley.edu/sites/default/files/styles/panopoly_image_original/public/sshtunnel1.png?itok=y6Yw7XaY&timestamp=1438209756