Efficient Unix System Management through Custom Shell, AWK, and Sed Scripting

Sambasiva Rao Madamanchi
Unix/Linux Administrator
National Institutes of Health (Bethesda, MD)

Abstract

This article provides an in-depth exploration of how system administrators can harness the power of Shell, AWK, and Sed to streamline and automate essential operational tasks in Unix and Linux environments. These lightweight command-line tools have stood the test of time due to their efficiency, portability, and ability to handle complex data manipulation with minimal overhead. Through a series of practical examples, the article demonstrates how scripting can be used to simplify a wide array of system administration duties ranging from log file monitoring and filtering, user and group management, resource reporting, to scheduled maintenance and backup routines. By focusing on real-world applications, the article goes beyond textbook syntax and showcases how these tools can be meaningfully applied in production environments. It illustrates how Shell scripts can orchestrate tasks, how AWK can be leveraged for dynamic text processing and reporting, and how Sed can perform in-place edits and transformations either as standalone utilities or in combination for more powerful workflows. Additionally, the article emphasizes the importance of writing secure and maintainable scripts by incorporating error handling, proper logging mechanisms, modular functions, and adherence to POSIX standards for portability.

Keywords: Unix, Linux, Shell, AWK, Admin.

1.Introduction

The Power of Lightweight Automation

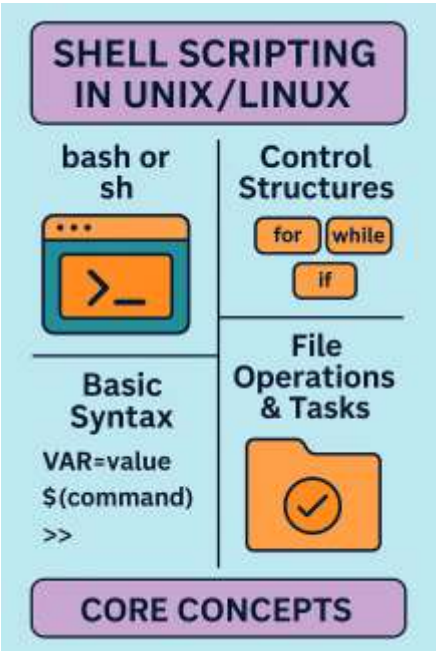
In the rapidly evolving world of IT infrastructure, system administrators are expected to handle a wide range of tasks efficiently and consistently. From managing users and monitoring system health to performing backups and automating deployments, the workload can be substantial. While modern tools like Ansible, Puppet, and Kubernetes provide comprehensive solutions for large-scale automation, the importance of traditional scripting tools Shell, AWK, and Sed remains undiminished. These tools form the foundational layer of Linux and Unix-based systems, providing a powerful and lightweight means to automate daily administrative chores without the overhead of more complex orchestration platforms (Naik 2018). The primary goal of this article is to bridge the gap between theoretical knowledge and practical application by providing a curated set of real-world examples that demonstrate how Shell, AWK, and Sed can be effectively used in everyday system administration tasks. Unlike generic tutorials, this article focuses on hands-on guidance, offering reusable script snippets and patterns that sysadmins can immediately apply or tailor to suit their environments.

Custom scripting with Shell, AWK, and Sed in Unix/Linux environments

Readers will learn how to automate user and permission management with Shell scripts, analyze and summarize logs using AWK, perform in-place file edits and batch text replacements with Sed, and combine all three tools to build powerful one-liners and automation scripts. Additionally, the article explores how to implement basic monitoring and alerting systems using these lightweight tools. Each example is accompanied by a clear explanation of its logic, common pitfalls to avoid, and suggestions for further customization. Beyond just presenting code, the article aims to cultivate a problem-solving mindset encouraging sysadmins to think resourcefully and work efficiently using the command-line tools already at their disposal (Parker 2011).

2. Overview of the Tools

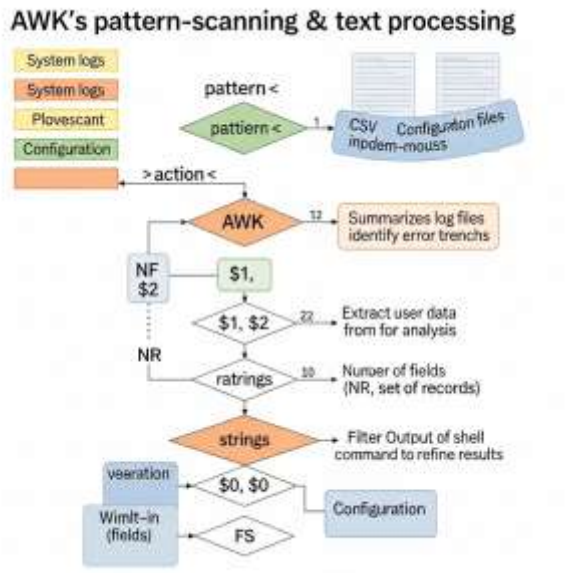
Shell Scripting (bash/sh)



Visual representation of shell scripting fundamentals

Shell scripting, particularly with bash or sh, is the cornerstone of Unix/Linux automation. It enables users to chain commands, use control structures like loops (for, while), conditional statements (if, case), and functions to create complex scripts. Basic syntax includes variable declarations (VAR=value), command substitution ($(command)), and input/output redirection (>, >>, <). Scripts can perform file operations, system calls, user management, backups, and more. Loops allow iteration over files or command outputs, making repetitive tasks easier. Conditionals help enforce logic and handle error checking, ensuring robust script behavior. Bash also supports arrays, arithmetic operations, and built-in utilities like read, echo, test, and trap. Shell scripts are portable, easy to debug, and can be scheduled with cron, making them ideal for routine sysadmin tasks. Whether checking disk usage or monitoring system services, a well-written shell script can dramatically reduce manual effort and improve consistency across system operations (Robbins and Beebe 2008).

AWK

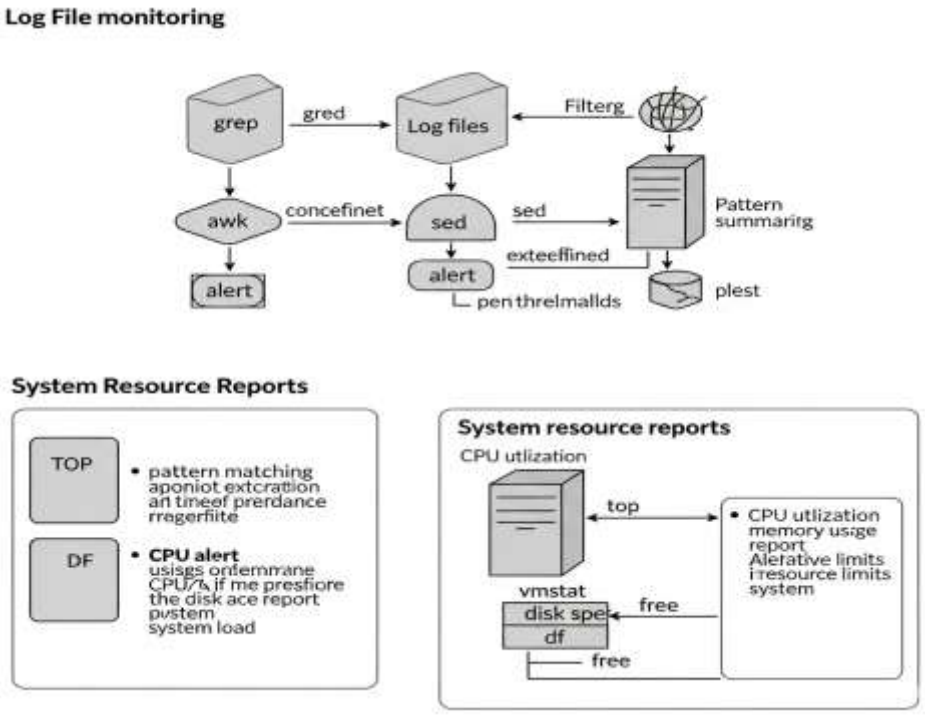


AWK is a powerful pattern-scanning and text-processing language designed for extracting and transforming data. It operates line-by-line on input files or streams, using a combination of patterns and actions (pattern { action }). AWK is especially useful for working with structured text files like logs, CSVs, and configuration files. Fields in a line can be accessed using $1, $2, etc., where $0 represents the entire line. It supports arithmetic operations, string manipulation, and built-in variables such as NF (number of fields), NR (record number), and FS (field separator). AWK scripts can be as simple as a one-liner to print specific columns or as complex as a multi-line program that aggregates statistics or formats output. It's an indispensable tool for sysadmins needing to summarize log files, extract user data from /etc/passwd, or filter command output. AWK's concise syntax and powerful built-in functions make it a go-to tool for efficient data handling (Aho et al., 1979).

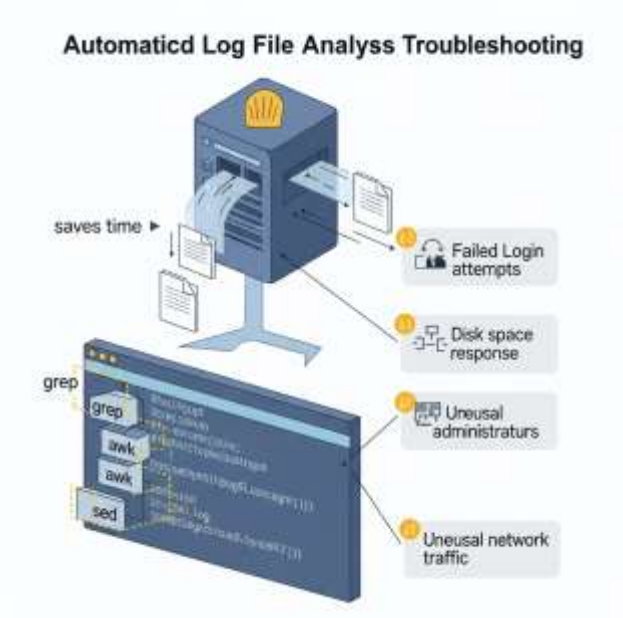
SED

Sed, short for "stream editor," is a non-interactive tool for parsing and transforming text directly from input streams or files. It reads input line by line, applies specified operations, and outputs the result without modifying the original file unless explicitly instructed to do so. The most common use of sed is substitution (s/old/new/), allowing for in-place editing of strings across multiple lines or files. Sed can also delete (d), insert (i), append (a), and replace lines based on patterns. It supports basic regular expressions and can be scripted for batch processing tasks. For example, sed can clean up configuration files, anonymize logs, or bulk-update file paths in scripts. Its ability to perform fast, automated edits without loading entire files into memory makes it ideal for processing large datasets or automating repetitive editing tasks. While sed's syntax can be terse, mastering it enables sysadmins to quickly manipulate text in powerful and efficient ways (Tharatipyakul et al., 2020).

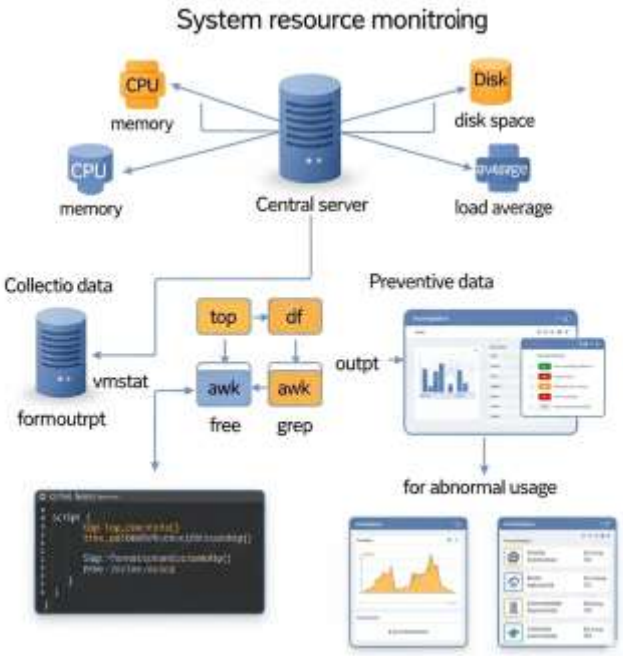
3. Common Sysadmin Tasks That Benefit from Scripting



Log File Monitoring and Filtering



Log files are critical for diagnosing issues and tracking activity on a system. Manually sifting through these files can be tedious, especially in high-volume environments. Scripting with tools like grep, awk, and sed allows sysadmins to automatically monitor logs for specific patterns such as failed logins, disk errors, or suspicious activity. Scripts can be scheduled to run at regular intervals, generate summaries, and even send alerts if certain thresholds are met. This proactive approach not only saves time but enhances security and response time to incidents by providing immediate visibility into potential system issues (Naik 2019).



Monitoring system resources such as CPU, memory, disk usage, and load average—is essential for maintaining optimal performance. Scripts can collect this data using commands like top, vmstat, df, and free, and then process the output with awk or grep to extract key metrics. These scripts can generate daily or weekly reports, highlight abnormal usage trends, and alert administrators if resources exceed predefined limits. Automating resource checks ensures continuous awareness of system health and helps prevent outages or performance degradation by addressing issues before they escalate (Kalkhanda 2018).

Automated Backups and File Archiving

Regular backups are a foundational aspect of system reliability. Using shell scripts, sysadmins can automate the process of creating file and database backups, compressing them with tools like tar or gzip, and transferring them to secure locations such as remote servers or cloud storage. Scripts can include logic to timestamp backups, rotate old archives, and verify integrity. This reduces the risk of human error and ensures that critical data is consistently protected (Lennert et al., 2004).

User and Group Management

Managing users and groups is a repetitive but necessary task in multi-user systems. Scripts can simplify and automate tasks like creating users, setting passwords, assigning group memberships, and updating permissions. For example, a script can read a list of users from a CSV file and create accounts in bulk, applying consistent configurations across all new users. Likewise, scripts can monitor /etc/passwd and /etc/group for unauthorized changes, enforce password policies, and clean up inactive accounts. This improves consistency, saves time, and reduces the risk of configuration errors in user management (Parsons et al., 2020).

Scheduled Maintenance and Cleanup

Systems accumulate clutter over time—temporary files, unused packages, old logs that can degrade performance. Scripts can be written to automate routine cleanup tasks such as clearing cache directories, truncating log files, and uninstalling unused software. These tasks can be scheduled via cron or systemd timers to run during off-peak hours, minimizing disruption. Automating cleanup not only keeps systems running smoothly but also ensures compliance with storage policies and security practices by eliminating sensitive or outdated data that could pose a risk if left unmanaged (Alfares 2022).

Network Health and Connectivity Checks

Maintaining network reliability is crucial, especially for servers that depend on external services or need to be accessible to users. Scripts can automate the process of checking network connectivity using commands like ping, curl, or netstat. They can verify that critical services are reachable, detect changes in IP routes, and log network performance data over time. More advanced scripts can also test port availability or latency to specific endpoints. When used in conjunction with alerting tools, these scripts help sysadmins quickly detect and respond to connectivity issues before they affect users or business operations (Bente et al., 2010).

4. Practical Shell Scripting Use Cases

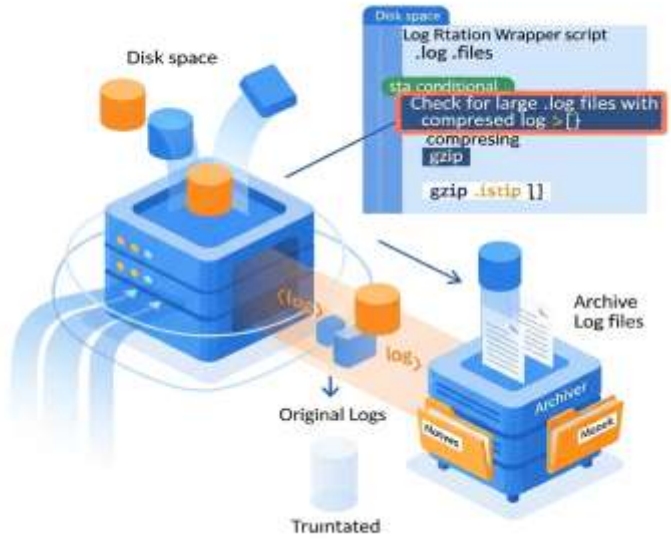
Disk Usage Alerts

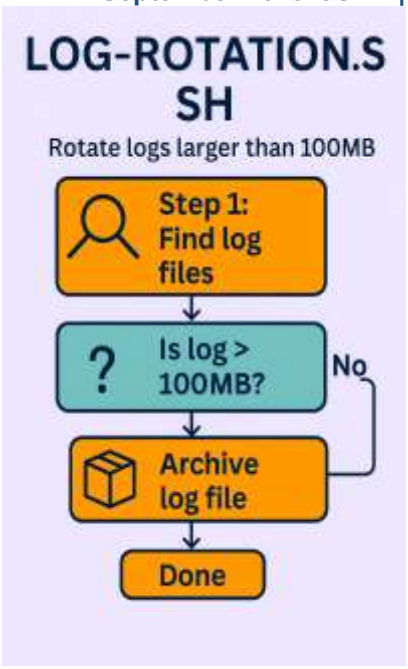
```
#!/bin/bash
# disk_usage_alert.sh - Alerts if disk usage is over 80%

echo "🔍 Checking disk usage..."
df -h | awk '$5+0 > 80 { print "⚠️ High Disk Usage:", $0 }'
```

Monitoring disk space is essential to prevent outages or system crashes due to full partitions. A common use case for Shell scripting is to check the current disk usage and identify partitions that are nearing capacity, typically over a set threshold like 80%. A script can automatically scan the output of system commands such as df and flag any partitions that exceed this limit. This type of script is lightweight, easy to schedule with cron, and can be enhanced to send alerts via email, log the output, or trigger automated cleanup actions, ensuring that administrators are warned before disk issues escalate (Dakic and Redzepagic 2022).

Log Rotation Wrapper

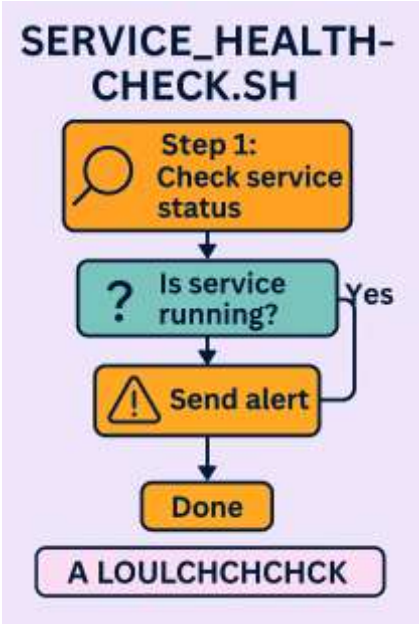
System and application logs can grow rapidly, consuming large amounts of disk space and impacting system performance. A Shell script can be used to automate the rotation of logs—archiving, compressing, and clearing them once they exceed a certain size or age. This not only helps conserve disk space but also keeps logs manageable for review or audit purposes. Such a wrapper script can check for large .log files in a directory, compress them using standard tools like gzip, store them in an archive directory, and truncate the original logs to maintain continuous logging without interruption (Ebrahim and Mallett 2018).





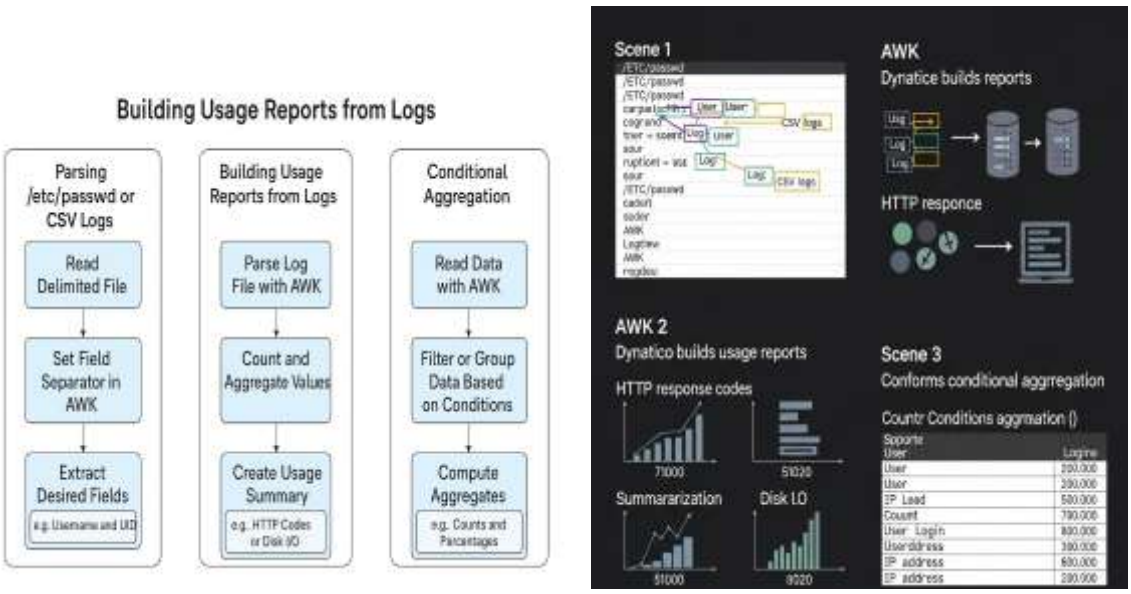
Service Health Check

```
#!/bin/bash
# service_health_check.sh - Checks if nginx is running

SERVICE="nginx"
EMAIL="admin@example.com"

if ! systemctl is-active --quiet "$SERVICE"; then
    echo "$SERVICE service is down on $(hostname) at $(date)" | \
    mail -s "ALERT: $SERVICE is DOWN" "$EMAIL"
fi
```



Keeping critical services like web servers or databases running is vital for system reliability. A practical Shell scripting use case involves regularly checking whether a specific service—such as nginx or mysql—is active. If the service is found to be inactive or has crashed, the script can notify the system administrator via email, a messaging platform, or even restart the service automatically. This kind of health check script can be scheduled to run every few minutes, ensuring rapid detection of failures and reducing downtime by allowing for swift corrective action (Flynt et al., 2017).

5. AWK for Data Extraction and Reporting



AWK excels at extracting structured data and generating concise, meaningful reports—making it a valuable tool for sysadmins and analysts alike. Its field-based processing model allows it to quickly parse files like /etc/passwd, CSVs, or logs, and extract targeted information without the need for more complex scripting languages.

Parsing /etc/passwd or CSV Logs: AWK makes it easy to extract specific fields from delimited files. For example, parsing /etc/passwd using a colon (:) as the field separator allows sysadmins to display usernames and UIDs with a simple one-liner. Similarly, it can parse CSV logs to extract timestamps, IP addresses, or status codes. This is ideal for automating audits or user data reports.

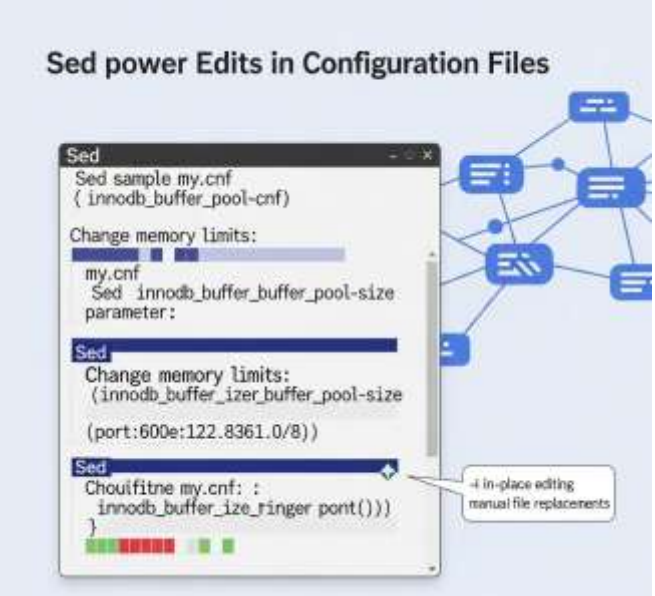
Building Usage Reports from Logs: AWK can analyze large log files to count HTTP response codes (e.g., 200s, 404s), summarize disk I/O from system stats, or aggregate SSH login attempts. It supports arithmetic and string operations, making it efficient for building performance summaries or traffic analysis dashboards directly from raw logs.

Conditional Aggregation: AWK can also perform grouping and aggregation based on conditions—such as counting occurrences by user or IP and calculating percentages. This enables sysadmins to produce distribution reports, usage statistics, or anomaly detection without needing a full database or spreadsheet tool (Robbins 2015)

6. Sed for Real-Time Text Manipulation

Sed (Stream Editor) is a lightweight, efficient tool for making real-time edits to text files and streams. Its power lies in the ability to perform inline editing, substitution, and pattern-based transformations with just a single command. Sed is especially useful for system administrators dealing with configuration files, large log files, or needing to automate repetitive text modifications.

Configuration File Edits



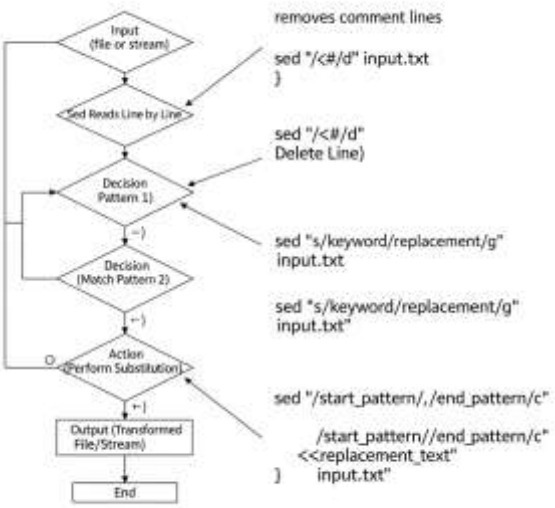
One of the most common uses of Sed is to perform automated edits in configuration files. For example, changing parameter values such as memory limits, port numbers, or service directives can be done without opening the file manually. Sed's -i flag allows in-place editing, making it ideal for scripting changes across multiple systems or containers. This reduces manual error and allows sysadmins to update configs during deployments or maintenance windows rapidly (Hanakawa and Obana 2019).

Cleanup of Log Entries

Sed is also perfect for cleaning or anonymizing log data. It can remove or mask sensitive information like IP addresses, usernames, or timestamps before sharing logs with third parties or archiving them. For example, you might use Sed to strip out login timestamps, redact email addresses, or remove noise from debug logs. This helps with compliance, privacy, and focusing only on the critical parts of a log during analysis (Beard 2016).

Multi-line and Pattern-based Replacements

While Sed primarily works line-by-line, it can be combined or chained to handle more complex edits. You can delete comment lines, replace all instances of a keyword, or even apply conditional replacements based on matching patterns. For instance, removing all lines starting with # (comments) and replacing the word “ERROR” with “Warning” improves the readability of logs and config files. These chained Sed commands can be embedded in scripts to handle bulk file transformations during system updates or migrations (Dougherty and Robbins 2008).

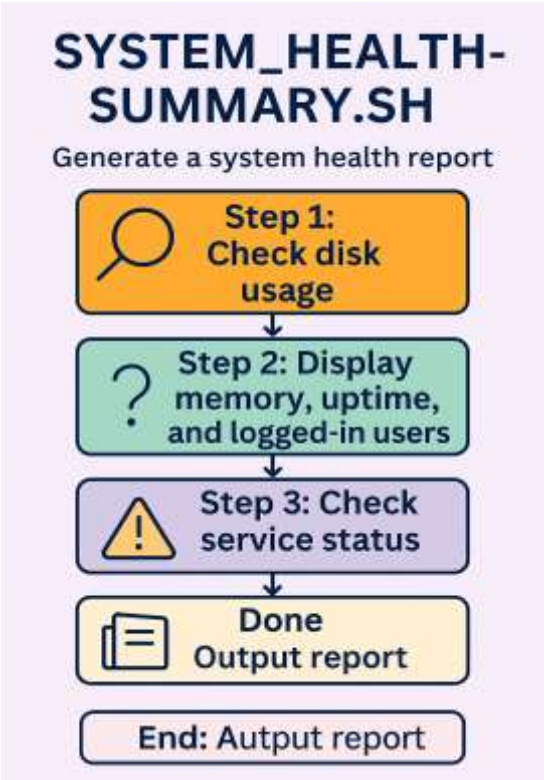


7. Combined Workflow Examples

System Health Summary Script

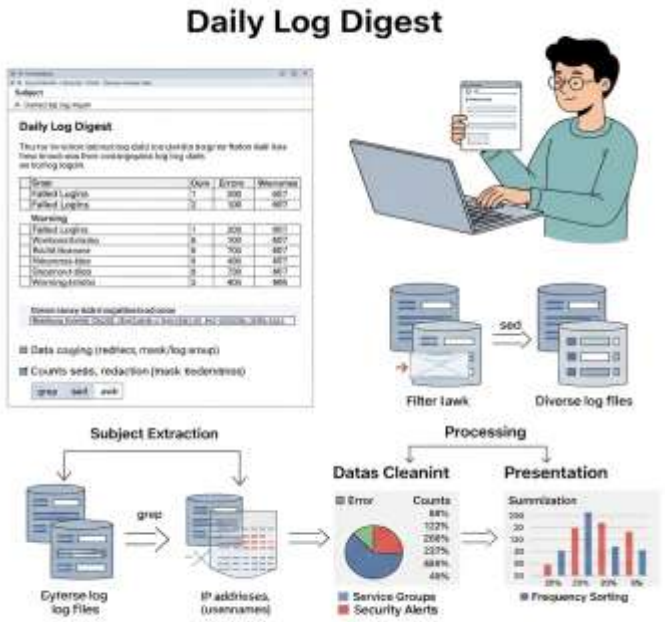
A System Health Summary Script brings together various system metrics to provide a comprehensive snapshot of the server's current state. This includes disk usage, memory status, system uptime, list of active users, and the status of critical services. A typical script would use df to report disk space, free or /proc/meminfo for memory, uptime for load averages, and who or users for logged-in users. systemctl or service commands can be included to verify that services like SSH, web servers, or databases are active. The results are formatted neatly, often with the help of AWK for aligning columns or summarizing values. This kind of script is especially useful as a daily report or a scheduled diagnostic tool that can alert admins to potential issues before they escalate (Kalkhanda 2018).

```
#hi/bcn.bash
echo "=== SYSTEM HEALTH SUMMARY ====
echo "=
echo >> Disk Usage (Partitions over
70%):
df -h\ | "awk $5+0 > 70 [print s0)"
echo "=
echo > Memory Status:
free -h
echo >> Uptime and Load Averages:
uptime
echo >> Currently Logged-in Users:
who
echo >> Service Status:
for for servece
on [systemctl is-active $service
2>>dev/null)
echo $sservice:= $status:-not found)
done
echo ====
======END OF OF REPORT ====

eccc
```



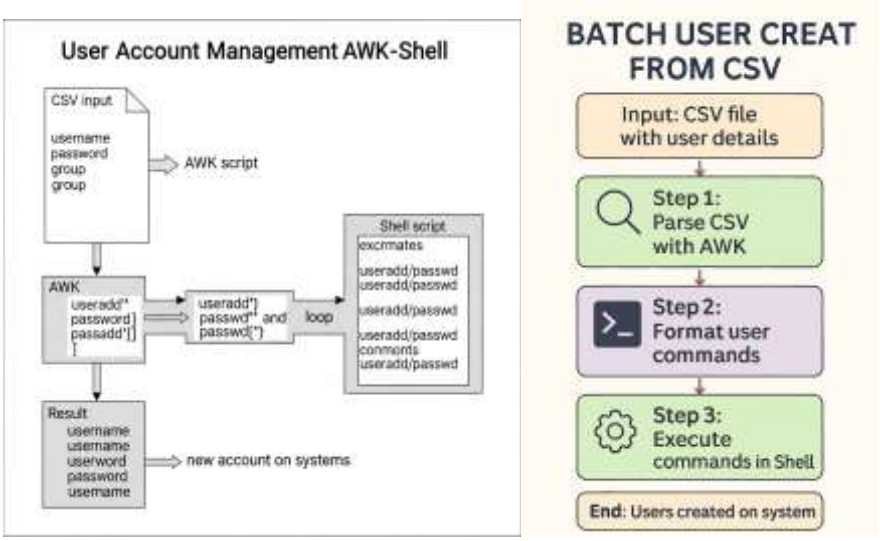
Daily Log Digest Email

Log monitoring becomes more effective when noise is filtered and insights are delivered in a digestible format. A daily log digest script typically chains together multiple Unix tools. It starts by using grep to extract relevant log entries (e.g., failed logins, errors, warnings), then sed may be used to clean up or redact sensitive data, such as IP addresses or usernames. AWK is then used to aggregate and summarize entries—counting error types, grouping by service, or sorting by frequency. Finally, the output is formatted into a readable report and sent via email using mailx, sendmail, or similar tools. This script helps administrators stay informed without manually parsing through massive log files every day (Topham 1990).



Batch User Creation from CSV

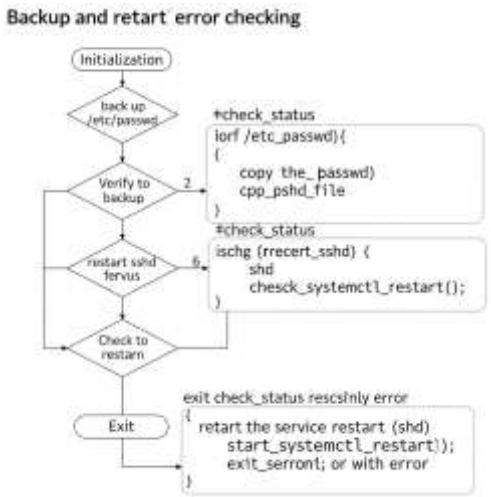
Managing user accounts in bulk can be streamlined by combining AWK and Shell scripting. A common workflow involves reading a CSV file where each row contains user details like username, UID, group, and shell. AWK is used to parse this CSV, extracting the relevant fields and formatting them into user creation commands (useradd, passwd, etc.). The Shell script loops through each formatted line and executes the commands to add the users to the system. This approach eliminates repetitive manual entry, ensures consistency, and supports integration with HR systems or onboarding workflows. Sed may also be used beforehand to clean malformed CSVs or remove unwanted characters (Mahmud et al., 2018).



8. Best Practices in Custom Scripting

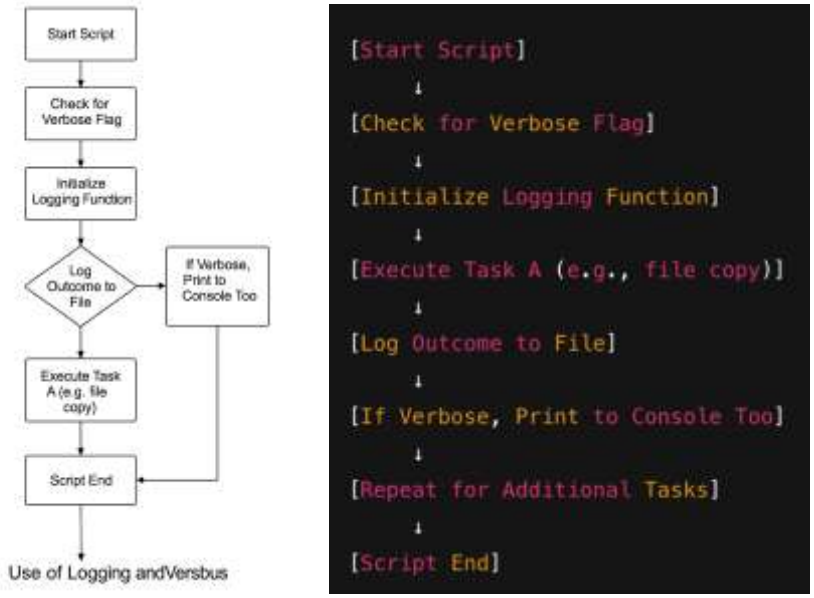
Error Handling and Exit Codes

Effective scripting requires robust error handling. Always check for potential failure points and use conditional statements to respond appropriately. The exit command with a meaningful status code (0 for success, non-zero for errors) helps identify script outcomes. Use constructs like || and && to handle failure cases, and test command results using $?. For critical tasks, wrapping commands in if or case statements ensures controlled behavior under failure conditions (Pearce 2003).

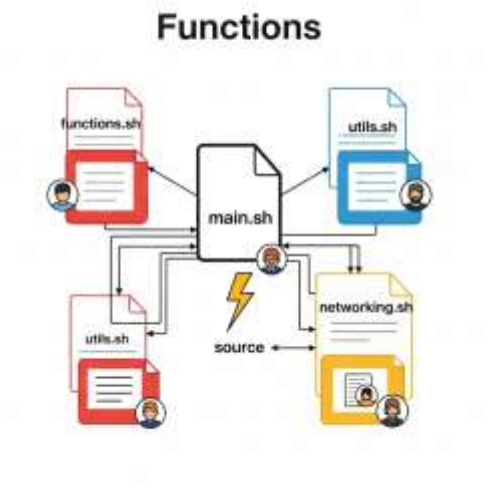


Use of Logging and Verbosity Flags

Incorporating logging mechanisms improves debuggability and transparency. Redirect output to log files using >> or tee, and use echo or logger for status messages. Include a -v (verbose) or --debug flag to give users optional insight into what the script is doing. This makes troubleshooting easier and ensures the script provides feedback without being overly noisy by default (Anu et al., 2019).



Modular Script Design (functions, sourcing)



Breaking your script into reusable functions enhances readability and maintainability. Use functions for repeated logic (e.g., logging, validation). Source shared libraries or helper scripts using . ./functions.sh, allowing for modular codebases. Modular design enables easier testing, reuse, and collaboration between sysadmin teams (Davis et al., 2011).

Portability Tips (POSIX vs Bash)

If portability across Unix-like systems is a concern, stick to POSIX-compliant syntax (#!/bin/sh). Avoid Bash-specific features like arrays or [[ ]] unless targeting Bash explicitly. Use tools like shellcheck to lint scripts and catch compatibility issues. Also, test scripts in different environments (e.g., Debian vs CentOS) to ensure consistent behavior (Abbott 2018).

Version Control and Documentation

Store all scripts in version control systems like Git. This helps track changes, roll back to stable versions, and collaborate effectively. Use comments liberally to document script purpose, parameters, and logic. Include usage examples and headers with metadata (author, date, version). A README.md for larger script collections aids understanding for team members or future you (Ronnau et al., 2005).

Security Considerations

Security is paramount when writing custom scripts, especially in system administration where scripts often have elevated privileges. One of the biggest risks is shell injection, which occurs when user input is executed as code. To prevent this, always sanitize inputs, avoid eval, and prefer built-in variable handling or command arrays that don't invoke the shell. Quoting variables and using input validation logic can also mitigate these risks. Handling sensitive data securely is another critical aspect. Avoid hardcoding passwords, API keys, or tokens into scripts. Instead, leverage environment variables, secured configuration files with restricted permissions, or secrets management tools (Foo et al., 1999). Output from such variables should not be logged or echoed to the terminal to prevent accidental exposure. File permissions must be set conservatively. Script files should have execute permissions only for trusted users (chmod 700 or chmod 750), and any temporary or output files should avoid world-readable or writable settings to prevent information leaks or tampering. Finally, limiting sudo usage reduces the attack surface. Scripts should only escalate privileges for specific tasks that require them, and not run the entire script as root unless absolutely necessary.

Consider using sudo on individual commands and always validate the calling user before doing so (Dai et al., 2020).

Conclusion

Scripting remains a vital skill in the toolkit of any system administrator due to the speed, flexibility, and control it offers for managing daily tasks. Whether it's monitoring system health, parsing logs, managing users, or automating backups, tools like Shell, AWK, and Sed provide lightweight yet powerful solutions that are easy to deploy and modify. These scripting tools empower sysadmins to react quickly to issues, streamline repetitive workflows, and maintain consistent environments without the overhead of heavy software. The key takeaway is to use the right tool for the right task and often, combining Shell, AWK, and Sed yields the most efficient results. Shell scripts can orchestrate system-wide operations, AWK excels at data extraction and reporting, and Sed provides fast in-place text manipulation. Together, they form a synergistic trio for solving real-world problems. As a next step, sysadmins should begin curating a personal library of reusable scripts that can be adapted across environments. For those ready to tackle larger-scale automation and configuration management, exploring languages like Python or tools like Ansible offers a natural progression. Mastering both foundational scripting and modern tools ensures long-term adaptability and effectiveness in a rapidly evolving infrastructure landscape.

References

1. Abbott, D. (2018). POSIX threads. *Linux for Embedded and Real-Time Applications*, 139–155. https://doi.org/10.1016/b978-0-12-811277-9.00009-2

2. Aho, A. V., Kernighan, B. W., & Weinberger, P. J. (1979). AWK — a pattern scanning and processing language. *Software: Practice and Experience*, *9*(4), 267–279. https://doi.org/10.1002/spe.4380090403
3. Alfares, H. K. (2022). Plant shutdown maintenance workforce team assignment and Job Scheduling. *Journal of Scheduling*, *25*(3), 321–338. https://doi.org/10.1007/s10951-021-00718-2
4. Anu, H., Chen, J., Shi, W., Hou, J., Liang, B., & Qin, B. (2019). An approach to recommendation of verbosity log levels based on logging intention. *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 125–134. https://doi.org/10.1109/icsme.2019.00022
5. Beard, B. (2016). Cleaning up maintenance files. *Practical Maintenance Plans in SQL Server*, 107–130. https://doi.org/10.1007/978-1-4842-1895-2_6
6. Bente, I., Vieweg, J., & von Helden, J. (2010). Privacy enhanced trusted network connect. *Lecture Notes in Computer Science*, 129–145. https://doi.org/10.1007/978-3-642-14597-1_8
7. Dai, T., Karve, A., Koper, G., & Zeng, S. (2020). Automatically detecting risky scripts in infrastructure code. *Proceedings of the 11th ACM Symposium on Cloud Computing*, 358–371. https://doi.org/10.1145/3419111.3421303
8. Dakic, V., & Redzepagic, J. (2022). *Linux command line and Shell scripting techniques: Master practical aspects of the linux command line and then use it as a part of the Shell scripting*. Packt Publishing.
9. Davis, D., Burry, J., & Burry, M. (2011). Understanding visual scripts: Improving collaboration through Modular Programming. *International Journal of Architectural Computing*, *9*(4), 361–375. https://doi.org/10.1260/1478-0771.9.4.361
10. Dougherty, D., & Robbins, A. (2008). *SED & AWK: UNIX power tools*. O'Reilly.
11. Ebrahim, M., & Mallett, A. (2018). *Mastering linux shell scripting: A practical guide to linux command-line, bash scripting, and Shell Programming, 2nd Edition*. Packt Publishing.
12. Flynt, C., Tushar, S., & Lakshman, S. (2017). *Linux shell scripting cookbook: Over 110 incredibly effective recipes to solve real-world problems, automate tedious tasks, and take advantage of linux's newest features*. Packt Publishing.
13. Foo, S., Chor Leong, P., Cheung Hui, S., & Liu, S. (1999). Security considerations in the delivery of Web-based applications: A case study. *Information Management & Computer Security*, *7*(1), 40–50. https://doi.org/10.1108/09685229910255197
14. Hanakawa, N., & Obana, M. (2019). A computer system quality metric for infrastructure with configuration files' changes. *Proceedings of the 2nd International Conference on Software Engineering and Information Management*, 39–43. https://doi.org/10.1145/3305160.3305168
15. Kalkhanda, S. (2018a). *Learning awk programming: A fast, and simple cutting-edge utility for text-processing on the unix-like environment*. Packt Publishing.
16. Kalkhanda, S. (2018b). *Learning awk programming: A fast, and simple cutting-edge utility for text-processing on the unix-like environment*. Packt Publishing.
17. Lennert, J. F., Retzner, W., Rodgers, M. G., Ruel, B. G., Sundararajan, S., & Wolfson, P. D. (2004). The automated backup solution-safeguarding the communications network infrastructure. *Bell Labs Technical Journal*, *9*(2), 59–84. https://doi.org/10.1002/bltj.20026

18. Mahmud, S. M., Hossin, M. A., Jahan, H., Noori, S. R., & Bhuiyan, T. (2018). CSV-annotate: Generate annotated tables from CSV file. *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*. https://doi.org/10.1109/icaibd.2018.8396169
19. Naik, G. S. (2018). *Learning linux shell scripting: Leverage the power of shell scripts to solve real-world problems*. Packt Publishing.
20. Naik, G. S. (2019). *Mastering python scripting for system administrators: Write scripts and automate them for real-world administration tasks using python*. Packt Publishing Ltd.
21. Parker, S. (2011). *Shell scripting: Expert recipes for linux, bash, and more*. Wiley ; John Wiley distributor.
22. Parsons, A. S., Wijesekera, T. P., & Rencic, J. J. (2020). The management script: A practical tool for teaching management reasoning. *Academic Medicine*, *95*(8), 1179–1185. https://doi.org/10.1097/acm.0000000000003465
23. Pearce, M. (2003). Error handling and exception management. *Comprehensive VB .NET Debugging*, 361–420. https://doi.org/10.1007/978-1-4302-0778-8_13
24. Robbins, A. (2015). *Effective AWK programming: Universal text processing and pattern matching*. O'Reilly Media.
25. Robbins, A., & Beebe, N. H. F. (2008). *Classic shell scripting: Hidden commands that unlock the power of unix*. O'Reilly Media, Inc.
26. Rönnau, S., Scheffczyk, J., & Borghoff, U. M. (2005). Towards XML version control of office documents. *Proceedings of the 2005 ACM Symposium on Document Engineering*, 10–19. https://doi.org/10.1145/1096601.1096606
27. Tharatipyakul, A., Li, J., & Cesar, P. (2020). Designing user interface for facilitating live editing in streaming. *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–8. https://doi.org/10.1145/3334480.3383037
28. Topham, D. W. (1990). Programming with AWK. *A System V Guide to UNIX and XENIX*, 207–224. https://doi.org/10.1007/978-1-4612-3246-9_14