Easy: Foundational Command Fluency

Focus: Mastering daily-use commands, flags, and basic piping.

1. Navigation & Inspection: List all files in the current directory, including hidden ones, in a long listing format. Sort them by modification time, newest first.
2. File Creation: Create a directory structure `project/{dev,prod,test}/config` with a single command.
3. Basic File Ops: Copy all `.log` files from `/var/log` to a directory `~/backups/logs`, preserving their original attributes.
4. User & Permissions: Change the owner of a script `deploy.sh` to user `deploy` and group `www-data`. Then give the owner execute permissions and the group read permissions.
5. Text Manipulation (head/tail): Display the first 5 lines and the last 10 lines of a large log file `application.log`.
6. Searching (grep): Search for all lines in `nginx.access.log` that contain the string `"500"` and save the results to a new file `errors.log`.
7. Process Management: Find the process ID (PID) of the `nginx` process and send it a `SIGHUP` signal to reload its configuration.
8. Disk Usage: Find the top 5 largest files or directories in the `/home` directory.
9. Downloading: Use `curl` to download a file from a URL and save it with a different name.
10. Basic Piping: List all running processes, filter for those belonging to your user, and count how many there are.

Medium: Scripting Fundamentals & System Interaction

Focus: Combining commands in scripts, variables, conditionals, and loops for automation.

11. Simple Script Skeleton: Write a script that takes a filename as an argument and prints "File exists" or "File does not exist" using an `if` statement.
12. For Loop: Write a script that pings a list of hostnames (read from a file `hosts.txt`) and reports whether each is reachable.
13. User Input: Write a script that asks the user for their name and then greets them.
14. Argument Parsing: Write a script that accepts a `-d` flag for a directory and a `-e` flag for a file extension (e.g., `.txt`). It should count all files with that extension in the given directory.
15. System Info Script: Write a script that outputs a useful system report: current date, disk usage, memory usage, and top 5 processes by CPU.
16. Backup Script: Create a script that creates a timestamped tar.gz backup of a specified directory and moves it to `/backups`.
17. Log Analyzer: Write a script that parses a generic log file and counts the

occurrences of each HTTP status code (e.g., 200, 404, 500).

18. String Manipulation: Write a script that takes a full path to a file (e.g., `/home/user/docs/report.txt`) and extracts just the filename (`report`) and just the directory (`/home/user/docs`).
19. API Interaction: Use `curl` in a script to GET a simple JSON API endpoint (e.g., `https://api.github.com/users/<username>`) and use `jq` to parse and extract a specific value (e.g., the `login` field).
20. Cron Job: Schedule your backup script (from #16) to run every Sunday at 2 AM using a cron job.
21. Function: Write a script that contains a function called `log_message` that appends a timestamped message to a log file. Use the function in your script.
22. Check for Tools: Write a script that checks if `docker`, `git`, and `jq` are installed and available in the `PATH`. If any are missing, print a helpful error message and exit with a non-zero status.
23. File Modification: Write a script that finds all `.conf` files in a directory and comments out (adds a `#` to the beginning of) any line containing the word `DEBUG`.
24. Interactive Delete: Write a script that finds all files older than 30 days in a directory and interactively asks for confirmation before deleting each one.
25. Port Check: Write a script that uses `netcat` or `/dev/tcp` to check if a specific port (e.g., 80, 443, 22) is open on a remote host.

Hard: Advanced Scripting & Robustness

Focus: Error handling, complex text processing, parallelism, and best practices.

26. Error Handling: Modify your backup script to exit immediately and cleanly if the source directory doesn't exist or if the tar command fails. Use `trap` to catch signals and perform cleanup.
27. Configuration Parsing: Write a script to parse a simple key=value config file, handling comments and blank lines. Then use the values in your script.
28. JSON/CSV Processing: Write a script that processes a CSV file (e.g., `users.csv`), uses `awk` or `cut` to extract fields, and creates a new Linux user for each row.
29. Interactive Menu: Create a script that presents a text-based menu (e.g., "1. Check Disk, 2. Check Memory, 3. Exit") and performs actions based on user selection.
30. Log Monitoring: Write a script that tails a log file in real-time and alerts (prints a message) when a specific error pattern appears more than 5 times in a minute.
31. Git Automation: Write a script that automates a git workflow: add all changed files, commit with a timestamped message, and push to the current branch. Handle the case where there's nothing to commit.

32. Docker Container Manager: Write a script that lists all running Docker containers, allows the user to stop one by name or ID, and then removes it.
33. Password Generator: Write a script that generates a random, secure password of a specified length (e.g., 16 characters) with a mix of upper, lower, numbers, and symbols.
34. Parallel Execution: Write a script that pings a list of 50 hosts from a file. Use GNU `parallel` or background processes (`&`) to ping them concurrently, not sequentially, to speed up the task.
35. Self-Documenting Script: Write a script that uses a `usage()` function and a `while getopts` loop to handle command-line flags (`-h` for help, `-v` for verbose mode, `-f <file>` for input file).
36. Safe `rm`: Create a wrapper function or script for `rm` that moves files to a "trash" directory instead of deleting them outright.
37. String Replacement in Bulk: Find all files (`.php`, `.html`) in a project and replace all instances of `old.example.com` with `new.example.com`.
38. Validate IP Address: Write a function that uses a regular expression within `bash` to validate if a given string is a valid IPv4 address.
39. Script Locking: Implement a mechanism in a long-running script to ensure only one instance of itself can run at a time (using a PID lock file).
40. SSH Automation: Write a script that uses SSH keys to run a simple command (e.g., `df -h`) on multiple remote servers listed in a file and collates the output.

Tricky: Edge Cases & Deep Understanding

Focus: Quoting, word splitting, exit codes, and subtle bash behaviors.

41. The Space Problem: Write a script that loops over all files in a directory and prints their names. Make it work correctly for filenames with spaces, newlines, and other special characters. (Hint: use `find -print0` and `while IFS= read -r -d ''`).
42. Quoting Hell: Explain the difference between `$@` and `$*`, and demonstrate a scenario where their behavior differs critically, especially when arguments contain spaces.
43. Source vs Execute: Create two scripts. Explain and demonstrate the difference between sourcing (`source script.sh`) a script and executing it (`./script.sh`).
44. Subshell Gotcha: Write a script that demonstrates how a variable set inside a loop piped into a `while` read loop is not available outside the loop. Then fix it.
45. Exit Code Chaining: Write a command that runs `command1` and only runs `command2` if `command1` fails. Then write a command that runs `command2` only if `command1` succeeds.

46. The Null Command: What is the purpose of `:` (the colon command)? Demonstrate its use in a script for a placeholder or a no-op.
47. Parameter Expansion Magic: Use parameter expansion to:
    a) Get the length of a variable string.
    b) Remove a trailing suffix from a string (e.g., `file.txt -> file`).
    c) Provide a default value if a variable is unset.
48. Arithmetic in Bash: Write a script that calculates the average of a list of numbers from a file, using only bash arithmetic (not external tools like `awk` or `bc`).
49. Test Builtin: Write an `if` statement using the `[[ ]]` test builtin that checks if a string matches a regex pattern and if a file is both readable and writable.
50. Signal Trapping: Write a script that traps the `SIGINT` (Ctrl-C) signal and prints "Interrupt received, cleaning up..." before exiting gracefully, instead of being killed immediately.
51. Here Documents: Write a script that generates a dynamic SQL or configuration file from variables using a Here Document (`<<EOF`), ensuring variable expansion happens correctly.
52. Process Substitution: Use process substitution (`<(command)`) to compare the output of two commands using `diff` without creating temporary files.
53. Debugging: Run a complex script with `set -x` to see trace output. Then use `trap 'echo $BASH_COMMAND' DEBUG` to achieve a similar but more customizable effect.
54. The `exec` Command: Demonstrate using `exec` inside a script to replace the current shell process with a new command (e.g., `exec java -jar app.jar`).
55. Read a file line by line: Write the most robust and efficient method to read a file line by line in bash, handling all edge cases.

How to Approach These Exercises

1. Try First: Don't just copy the answers. Try to solve them using `man` pages and online resources.
2. Test Rigorously: Test your scripts with different inputs, including edge cases (empty input, paths with spaces, non-existent files).
3. Review & Refactor: Once it works, see if you can make it more efficient, readable, or robust. Look up "ShellCheck" and use it to analyze your scripts.
4. Understand the "Why": For the tricky problems, the goal is to understand the underlying bash mechanic, not just get the right output.

Mastering these exercises will give you a deep, practical mastery of shell scripting that is invaluable for any DevOps role.