Bash Scripting Project: Creating a Multi-Tool Utility

Overview

In this project, you will create a comprehensive bash utility script called "toolz" that combines multiple helpful system tools into a single, easy-to-use command-line interface. This project will help you practice bash scripting fundamentals, command-line parameter handling, and system administration tasks.

Learning Objectives

- Create modular bash functions for different utilities
- Implement command-line argument parsing
- Practice interactive user input collection and validation
- Apply system administration commands in practical contexts
- Write clear usage documentation within the script

Requirements

Core Functionality

Your "toolz" script must implement the following functionality:

1. **Find Helper** (`-f` option)
   - Create an interactive helper for the "find" command
   - Guide users through specifying search criteria step by step
   - Allow customization of search location, patterns, and other find parameters
2. **System Information** (`-s` option)
   - Display essential system metrics in a readable format:
     - Available memory
     - Number of running processes
     - Disk usage statistics
3. **Process Management** (`-p` option)
   - Show the top CPU/memory consuming processes
   - Allow sorting by different metrics (CPU, memory, runtime)
   - Include an option to kill processes interactively
4. **User Management** (`-u` option)
   - Show currently logged-in users and their activities
   - Display user account information

    - Implement basic user management functions (with appropriate permissions)

5. **Help System** (`-h` option)
    - Display comprehensive usage information
    - Include examples of each feature
    - Provide detailed descriptions of all available options

Implementation Details

Process Management (`-p`)

Your process management implementation should:

- Display a list of processes sorted by resource usage
- Include at least three sorting options (e.g., CPU, memory, runtime)
- Provide an interactive mode to filter or kill processes
- Use commands like `ps`, `top`, or `kill` appropriately

User Management (`-u`)

Your user management implementation should:

- Show active user sessions with details like login time and activity
- Provide options to display detailed user account information
- Include functionality to show user resource usage
- Implement appropriate permission checks before any administrative actions
- Make use of commands like `who`, `w`, `last`, `id`, and `passwd`

Help System (`-h`)

Your help system should:

- Provide clear explanations of each option and sub-option
- Include practical examples showing syntax and common use cases
- Format the output for easy readability (consider sections, indentation)
- Show appropriate error messages when invalid options are used

Script Structure Requirements

- Use functions to organize code
- Include appropriate error handling
- Validate user input

- Add comments explaining complex operations
- Follow bash scripting best practices
- Make the script executable

Bonus Challenges

- Implement color coding for better readability
- Add configuration options that persist between runs
- Create a simple menu-driven interface when run without parameters
- Add user privilege level detection for appropriate permissions
- Implement confirmation prompts for potentially dangerous operations

Submission Requirements

1. Your complete bash script file named "toolz"
2. A brief document explaining:
   - Your implementation approach and design decisions
   - How you structured your code and the purpose of each function
   - Any challenges you encountered and how you resolved them
3. Examples of using each feature you implemented

Code Structure and Organization Requirements

This project places significant emphasis on proper code structure and organization. Your submission must demonstrate:

1. **Modular Design Using Functions**
   - Create separate functions for each major tool/feature
   - Implement helper functions for repetitive tasks
   - Ensure each function has a clear, single purpose
2. **Well-Structured Code**
   - Organize your script in a logical flow
   - Group related functions together
   - Use meaningful variable and function names
   - Maintain consistent indentation and formatting
3. **Proper Documentation**
   - Include a header comment explaining the script's purpose
   - Document each function with description, parameters, and return values

   - Add inline comments for complex logic or non-obvious operations

4. **Error Handling**
   - Implement appropriate error checking
   - Display meaningful error messages
   - Ensure the script exits gracefully on errors

5. **Input Validation**
   - Validate all user inputs before processing
   - Provide helpful feedback when invalid input is detected

Remember: Good code structure is not just about making the script work—it's about making it maintainable, readable, and extendable.

Resources

- Bash scripting guide: GNU Bash Manual
- Linux command references: `man find`, `man ps`, `man grep`, etc.
- System administration commands: `free`, `top`, `df`, `netstat`, `ss`

Good luck, and have fun creating your multi-tool utility!