Website Blocker

Rationale

I built this website blocker to stay focused after I realized how much time I was wasting on social media and streaming sites. Instead of relying on third-party tools, and having just finished the Introduction to Bash Scripting module on Hack The Box, I wanted to create my own simple script to block and unblock websites locally.

New Things I Learned

Since I completed the Introduction to Bash Scripting module on HTB, I already had most of the required knowledge to write the website blocking script myself, but I did learn a few new things along the way.

Working on this project taught me a lot about system-level file manipulation. I gained a deeper understanding of the hosts file, which is a powerful way to control domain resolution locally. By adding an entry like `127.0.0.1 example.com`, I could redirect a domain to localhost, effectively blocking it.

I also had to implement domain validation using regular expressions, ensuring that only valid domain names were processed. Additionally, I revisited how to handle command-line arguments dynamically using positional arguments and `case` statements, which allowed the script to accept options like `--add` and `--remove`.

Writing the Website Blocker

I first implemented a `show_help` function to provide users with guidance on how to use the script. This function outputs the correct syntax for running the script and explains the available options: `--add` for blocking domains and `--remove` for unblocking them.

To handle input validation, I created a `validate_domain` function. This function uses a regular expression to ensure that any domain provided by the user is valid. If an invalid domain is detected, the script immediately exits with an error message.

The core functionality of the script lies in the `modify_hosts` function. This function accepts an action (`add` or `remove`) and a list of domains. For each domain, it first validates the domain name. If the action is `add`, the script checks if the domain is already blocked by searching for the corresponding entry in the `hosts` file. If not, it appends the entry to the file. For the `remove` action, the script searches for the domain in the file and deletes the corresponding line if it exists. I used simple but effective operations like `grep`, `echo`, and `sed` to manipulate the file.

Since modifying the `hosts` file requires administrative privileges, I also added a check at the beginning of the script to ensure it is run as root.