The way the script manages arguments and options

First, it makes a check for the --help option. If the option exists, it displays usage instructions and exits.

It then uses getopts to process:

-n (show line numbers)

-v (invert match)

It stores these choices into Boolean variables (show_line_numbers, invert_match).

Following parsing of the options, the arguments are rearranged so that the search string and the filename are in the correct position.

It checks:

If the search string or the filename is missing → it displays an error.

If the file doesn't exist → it displays an error.

It constructs the final grep command dynamically based on the chosen options and executes it.

Reflective Section

How the script accepts arguments and options

I used getopts to handle the flags -n and -v. Based on the presence of these flags, I initialize two Boolean variables. After handling the flags, I shift the positional parameters to correctly obtain the search string and filename.

If I were going to implement support for regex, or for options like -i, -c, -l, how would I adjust the structure?

I would expand getopts to accommodate additional flags (e.g., -c for counting matches, -l for listing matching filenames only).

I would include additional Boolean variables to track those new options.

Then, I would construct the correct grep command dynamically depending on the combination of flags.

Supporting full regex wouldn't require much alteration, as grep already supports regex. However, I might add input validation or provide guidance on escaping special characters.

What was the most difficult part to implement and why?

Frankly, the most challenging part was handling the combination of options (-v, -n, or both like -vn, -nv) while still correctly retrieving the search string and filename.

Using getopts solved this neatly, but initially, manually parsing the options was messy.