# NASA HW1 - 金哲安(B12902118)

## 1.

### References

- https://www.baeldung.com/cs/merkle-trees (https://www.baeldung.com/cs/merkle-trees)
    - Introduction to merkle trees
- https://www.bogotobogo.com/Linux/linux_File_Types.php (https://www.bogotobogo.com/Linux/linux_File_Types.php)
    - Explanation of Linux file types
- https://www.man7.org/linux/man-pages/man1/test.1.html (https://www.man7.org/linux/man-pages/man1/test.1.html)
    - `test` man page
- https://www.reddit.com/r/bash/comments/111x8g2/test_f_and_e_give_the_same_results_when_testing_a/ (https://www.reddit.com/r/bash/comments/111x8g2/test_f_and_e_give_the_same_results_when_testing_a/)
    - Test for symlinks
- B12902066 宋和峻
    - Discussing about extra space on blank lines when printing usage
- https://chatgpt.com/share/67c2ea9a-52ac-8005-a130-926e3584937f (https://chatgpt.com/share/67c2ea9a-52ac-8005-a130-926e3584937f)
    - Solving long option arguments with `=`

I have written a shell script:

```
#!/bin/bash

usage() {
    local exit_status=$1
    cat <<EOF
merkle-dir.sh - A tool for working with Merkle trees of directories.

Usage:
  merkle-dir.sh <subcommand> [options] [<argument>]
  merkle-dir.sh build <directory> --output <merkle-tree-file>
  merkle-dir.sh gen-proof <path-to-leaf-file> --tree <merkle-tree-file> --output <proof-file>
  merkle-dir.sh verify-proof <path-to-leaf-file> --proof <proof-file> --root <root-hash>

Subcommands:
  build          Construct a Merkle tree from a directory (requires --output).
  gen-proof      Generate a proof for a specific file in the Merkle tree (requires --tree and --output).
  verify-proof   Verify a proof against a Merkle root (requires --proof and --root).

Options:
  -h, --help     Show this help message and exit.
  --output FILE  Specify an output file (required for build and gen-proof).
  --tree FILE    Specify the Merkle tree file (required for gen-proof).
  --proof FILE   Specify the proof file (required for verify-proof).
  --root HASH    Specify the expected Merkle root hash (required for verify-proof).

Examples:
  merkle-dir.sh build dir1 --output dir1.mktree
  merkle-dir.sh gen-proof file1.txt --tree dir1.mktree --output file1.proof
  merkle-dir.sh verify-proof dir1/file1.txt --proof file1.proof --root abc123def456
EOF
    exit $exit_status
}

OPTIONS="h"
```

```
LONGOPTIONS="help,output:,tree:,proof:,root:"
PARSED="$(getopt -l "$LONGOPTIONS" -o "$OPTIONS" --
"$@" 2>/dev/null)"
# Prints the usage if errors happened during passing options
if [[ $? -ne 0 ]]; then
    usage 1
fi
eval set -- "$PARSED"
# echo "Parsed arguments: $@">&2
total_arguments=$(($#-1))
while true; do
    case "$1" in
        -h)
            h="true"
            shift
            ;;
        --help)
            help="true"
            shift
            ;;
        --output)
            output=$2
            shift 2
            ;;
        --tree)
            tree=$2
            shift 2
            ;;
        --proof)
            proof=$2
            shift 2
            ;;
        --root)
            root=$2
            shift 2
            ;;
        --)
            shift
            break
            ;;
        *)
            break
            ;;
    esac
done
```

It can parse the arguments during execution with getopt, but

there is a problem. For this command: `build dir1 --output=your_output/dir1.mktree`, it can be successfully parsed by getopt, but I want to block this usage. Every long option argument should be in the form of `--option <argument>`.

You are a shell script expert. Please come up with a solution to only allow that usage.

You can modify your script to check whether any long option has been given in the `--option=value` format and reject it. Since `getopt` does not differentiate between `--output=value` and `--output value`, you need to check `$PARSED` before evaluating it with `set --`.

**Solution:**

Add the following validation before calling `eval set -- "$PARSED"`:

bash Copy Edit

```bash
# Check for incorrect usage of long options in the form --option=value
for arg in "$@"; do
  if [[ "$arg" =~ ^--(output|tree|proof|root)= ]]; then
    echo "Error: Long options must be specified as '--option <argument>', not '--opti
    exit 1
  fi
done
```

**Explanation:**

1. This loop runs through `"$@"` (the original arguments before `getopt` processing).
2. It uses a regular expression (`^--(output|tree|proof|root)=`) to detect any long option that is followed by `=`, meaning an incorrect format like `--output=your_output/dir1.mktree`.
3. If such an option is found, it prints an error message and exits with status `1`, rejecting the incorrect usage.

This ensures that options must always be provided in the form `--option <argument>`.

## 2.

## References


- https://www.simplilearn.com/xxd-command-in-linux-article (https://www.simplilearn.com/xxd-command-in-linux-article)
  - `xxd` tutorial
- https://www.shellscript.sh/ (https://www.shellscript.sh/)
  - Basic shell script concepts

- https://medium.com/vswe/bash-shell-script-cheat-sheet-15ce3cb1b2c7 (https://medium.com/vswe/bash-shell-script-cheat-sheet-15ce3cb1b2c7)
  - Basic bash syntax
- https://stackoverflow.com/questions/19670061/bash-if-false-returns-true-instead-of-false-why (https://stackoverflow.com/questions/19670061/bash-if-false-returns-true-instead-of-false-why)
  - Explanation on the behaviors of `[]` and `[[]]`
- https://cool.ntu.edu.tw/courses/46924/files/7073160?module_item_id=1935310 (https://cool.ntu.edu.tw/courses/46924/files/7073160?module_item_id=1935310)
  - `[]` testing options
- https://man7.org/linux/man-pages/man1/find.1.html (https://man7.org/linux/man-pages/man1/find.1.html)
  - Man page of `find`
- https://unix.stackexchange.com/questions/78625/using-sed-to-find-and-replace-complex-string-preferrably-with-regex (https://unix.stackexchange.com/questions/78625/using-sed-to-find-and-replace-complex-string-preferrably-with-regex)
  - Using extended regex (for `+` and grouping) in `sed`
- https://regex101.com/ (https://regex101.com/)
  - Regex builder and validation
- https://docs.google.com/presentation/d/1pcLXptV4PBxsH-T35E0CCAu5jakcaXj02s4vphmS7p8/edit#slide=id.g33122053a3d_2_36 (https://docs.google.com/presentation/d/1pcLXptV4PBxsH-T35E0CCAu5jakcaXj02s4vphmS7p8/edit#slide=id.g33122053a3d_2_36)
  - Integer computation in bash
- https://stackoverflow.com/questions/13111967/raise-to-the-power-in-shell (https://stackoverflow.com/questions/13111967/raise-to-the-power-in-shell)
  - Raising powers in bash
- https://gary840227.medium.com/linux-bash-array-介紹-6e30ffe87978 (https://gary840227.medium.com/linux-bash-array-%E4%BB%8B%E7%B4%B9-6e30ffe87978)

    - Using arrays in bash
- https://opensource.com/article/18/5/you-dont-know-bash-intro-bash-arrays (https://opensource.com/article/18/5/you-dont-know-bash-intro-bash-arrays)
    - More arrays in bash

## 3.

---

### References

- https://stackoverflow.com/questions/61073688/how-to-use-in-the-name-of-a-bash-variable (https://stackoverflow.com/questions/61073688/how-to-use-in-the-name-of-a-bash-variable)
    - Hyphens are not allowed in bash variables
- https://stackoverflow.com/questions/38987849/the-function-definitions-in-shell-use-dash (https://stackoverflow.com/questions/38987849/the-function-definitions-in-shell-use-dash)
    - Hyphens are allowed in bash function names
- https://stackoverflow.com/questions/70746111/why-does-read-not-have-a-normal-man-page (https://stackoverflow.com/questions/70746111/why-does-read-not-have-a-normal-man-page)
    - `read` in bash
- https://www.grymoire.com/Unix/Sed.html#uh-1 (https://www.grymoire.com/Unix/Sed.html#uh-1)
    - `sed` tutorial
- https://www.tutorialspoint.com/awk/awk_overview.htm (https://www.tutorialspoint.com/awk/awk_overview.htm)
    - `awk` tutorial

# 4.

## References