Data Synchronizer (dsync)

Written By: Devon Bautista

October 2016

Abstract

The goal of this script is to synchronize two directories, particularly for backing up large data collections. One directory shall be the dominant directory while the other shall be the subordinate directory. The subordinate directory shall be synchronized to whatever the current state of the dominant directory is. Colloquially, the subordinate directory can be thought of as the backup directory and the dominant directory as the directory being backed up.

Requirements

1. The program shall take two inputs (excluding options): the first of which is the dominant directory and the second of which is the subordinate directory.
2. For every file/directory that is in the dominant directory that is not in the subordinate directory, that file/directory shall be copied from the dominant directory to the subordinate directory.
3. For every file/directory that is in the subordinate directory that is not in the dominant directory, that file/directory shall be deleted from the subordinate directory.
4. For every mutual file/directory between both the dominant and subordinate directory that has different attributes (i.e. size, date modified), the file/directory will be handled according to the option specified by the user. By default, if the file/directory in the dominant directory is newer than the file in the subordinate directory, then the file/directory in the subordinate directory will be replaced by the file/directory in the dominant directory.
5. If one of the directories input by the user doesn't exist, then the program will terminate after notifying the user so.
6. For every file in the subordinate directory which takes priority (is newer/older/smaller/larger depending on the mode) over the corresponding file in the dominant directory, it shall NOT be replaced unless the *force* option is passed.

Options:

- Verbose Option. Send to stdout everything the program does.
- Large File Mode. Large files take precedence when synchronizing.

- Small File Mode. Small files take precedence when synchronizing.
- New File Mode. Newer files take precedence when synchronizing (default).
- Old File Mode. Older files take precedence when synchronizing.

High-Level Design

1. Receive (in order) options, the dominant directory, and the subordinate directory.
2. Parse input.
3. Synchronize the dominant and subordinate directories according to the specified options.
  a. Print actions if verbose option is specified.
4. Display any errors and write them to a log.
5. Terminate

Mid-Level Design

Function Descriptions

main() - Performs the runtime operations of the program. Executed on program start.

parse_args() - Parses arguments passed to the program from the command line. Returns (in order) the mode of syncing, the Boolean value of the verbose flag, the dominant directory, and the subordinate directory.

copytree(*src*, *dst*, *symlinks*, *ignore*) – Intelligently copies all files AND sub-directories in *src* to *dst*. (*symlinks* and *ignore* are *os* module arguments, *False* and *None* by default.)

sync(*mode*, *verbose*, *dom*, *sub*, *force*) – Synchronizes directories *dom* and *sub* based on *mode*. Displays actions if *verbose* is true. If *force* is true, any files in *sub* that take priority (based on *mode*) will replaced by their corresponding file in *dom*.

sync_dirs(*mode*, *verbose*, *force*, *dom*, *sub*) – Obtains file and directory paths for corresponding sub-directories and files for each one in *dom* and *sub*.

Function Outlines

1. *main*()
   1. Parse input using *parse_args*() and store return values.
   2. Pass stored values to *sync_dirs*(*mode, verbose, dom, sub*).
   3. Terminate

2. *parse_args*()
   1. If “-h” is in command line arguments:
      a. Display help

```
        b. Terminate.
    2. If “-v” is in command line arguments:
        a. Set *verbose* to true.
    3. If “-f” is in command line arguments:
        a. Set *force* to true.
    4. If “-l” is in command line arguments:
        a. Set *mode* to “large”.
    5. Otherwise, if “-s” is in command line arguments:
        a. Set *mode* to “small”.
    6. Otherwise, if “-n” is in command line arguments:
        a. Set *mode* to “new”.
    7. Otherwise, if “-o” is in command line arguments:
        a. Set *mode* to “old”.
    8. Otherwise:
        a. Set *mode* to “new”.
    9. If dominant directory exists:
        a. Store dominant  directory.
    10. Otherwise:
        a. Output “Invalid directory: ” *dom*
        b. Terminate
    11. If subordinate directory exists:
        a. Store subordinate directory.
    12. Otherwise:
        a. Output “Invalid directory: ” *sub*
        b. Terminate
    13. Return *mode*, *verbose*, *dom*, *sub*

3. *sync*(*mode*, *verbose*, *dom*, *sub*, *force*)
    1. For all items in *sub*:
        a. If item does not exist in *dom*:
            1. Delete it.
    2. For all item in *dom*:
        a. If item does not exist in *sub*:
            1. Copy it to *sub*.
        b. Otherwise:
            1. If item is NOT a directory:
                a. If *mode* is “large”:
                    1. If file in *dom* is larger than file in *sub*:
                        a. Replace file in *sub* with file in *dom*.
                        b. If *verbose* is true:
                            1. Output action.
```

               2. Otherwise if file in *dom* is smaller than file in *sub*:
                  a. If *force* is true:
                     1. Replace file in *sub* with file in *dom*.
            b. If *mode* is “small”:
               1. If file in *dom* is smaller than file in *sub*:
                  a. Replace file in *sub* with file in *dom*.
                  b. If *verbose* is true:
                     1. Output action.
               2. Otherwise if file in *dom* is larger than file in *sub*:
                  a. If *force* is true:
                     1. Replace file in *sub* with file in *dom*.
            c. If *mode* is “new”:
               1. If file in *dom* is newer than file in *sub*:
                  a. Replace file in *sub* with file in *dom*.
                  b. If *verbose* is true:
                     1. Output action.
               2. Otherwise if file in *dom* is older than file in *sub*:
                  a. If *force* is true:
                     1. Replace file in *sub* with file in *dom*.
            d. If *mode* is “old”:
               1. If file in *dom* is older than file in *sub*:
                  a. Replace file in *sub* with file in *dom*.
                  b. If *verbose* is true:
                     1. Output action.
               2. Otherwise if file in *dom* is newer than file in *sub*:
                  a. If *force* is true:
                     1. Replace file in *sub* with file in *dom*.

4. *sync_dirs*(*mode*, *verbose*, *force*, *dom*, *sub*)
   1. If either directory is empty:
      a. *sync*(*mode*, *verbose*, *dom*, *sub*)
   2. Otherwise:
      a. For all sub-directories in *dom*:
         1. Get corresponding file paths for each sub-directory in *sub* and *dom*.
         2. *sync*(*mode*, *verbose*, dom_path, sub_path, *force*)

Pseudocode

```
function main()
    mode, verbose, dom, sub ← parse_args()
    sync_dirs(mode, verbose, dom, sub)
```

```
function parse_args()
    if "-h" in command line arguments:
        output help
        terminate
    if "-v" in command line arguments:
        verbose ← true
    otherwise
        verbose ← false
    if "-f" in command line arguments:
        force ← true
    otherwise
        force ← false
    if "-l" in command line arguments:
        mode ← "large".
    else if "-s" in command line arguments:
        mode ← "small".
    else if "-n" in command line arguments:
        mode ← "new".
    else if "-o" in command line arguments:
        mode ← "old".
    else
        mode ← "new".
    if is_directory(second-last command line argument) is true
        dom ← second-last command line argument
    else
        output "Invalid directory: " + dom
        terminate
    if is_directory(last command line argument) is true
        sub ← last command line argument
    else
        output "Invalid directory: " + sub
        terminate
    return mode, verbose, force, dom, sub

function sync(mode, verbose, dom, sub, force)
    for sub_item in list_items(sub):
        if sub_item not in list_items(dom):
            remove(sub_item)
    for dom_item in list_items(dom):
        if dom_item not in list_items(sub):
            copy(dom, join_path(sub, dom_item))
```

```
else:
    if dom_item not is_directory():
        if mode == “large”:
            if stats(dom + dom_item).size > stats(sub+ dom_item).size:
                copy(dom + dom_item, sub + dom_item)
                    if verbose == true:
                        output “Replacing dom_item with larger file.”
            else if stats(dom + dom_item).size < stats(sub+ dom_item).size:
                if force == true:
                    replace(sub + dom_item, dom + dom_item)
        else if mode == “small”
            if stats(dom + dom_item).size < stats(sub + dom_item).size:
                copy(dom + dom_item, sub + dom_item)
                    if verbose == true
                        output “Replacing dom_item with smaller file.”
            else if stats(dom + dom_item).size > stats(sub + dom_item).size:
                if force == true:
                    replace(sub + dom_item, dom + dom_item)
        else if mode == “new”
            if stats(dom + dom_item).date > stats(sub + dom_item).date:
                copy(dom + dom_item, sub + dom_item)
                if verbose == true
                    output “Replacing dom_item with newer file.”
            else if stats(dom + dom_item).date < stats(dom + dom_item).date:
                if force == true:
                    replace(sub + dom_item, dom + dom_item)
        else if mode == “old”
            if stats(dom + dom_item).size > stats(sub + dom_item).size:
                copy(dom_path + dom_item, sub + dom_item)
                if verbose == true
                    output “Replacing dom_item with older file.”
            else if stats(dom + dom_item).date < stats(dom + dom_item).date:
                if force == true:
                    replace(sub + dom_item, dom + dom_item)
```

Test Cases

Test Case 1:
Verifies Software Requirement(s): 1
Description: Verifies that the program takes, at minimum, two strings representing directories.
Input Data: dom = ~/dir1 sub = ~/dir2
Expected Behavior: Program will attempt to compare the two directories.
Actual Behavior: Program received arguments and existed, since there were no changes between them.
Result: PASSED

Test Case 2:
Verifies Software Requirement(s): 2
Description: Verifies that the program will copy a file from dom to sub if it exists in dom and does not exist in sub.
Input Data: dom = ~/dir1, sub = ~/dir2
Expected Behavior: Program will copy a file located in dir1 to dir2.
Actual Behavior: Program copied test file from dir1 to dir2.
Result: PASSED

Test Case 3:
Verifies Software Requirement(s): 2
Description: Verifies that the program will NOT copy a file from dom to sub if sub contains an identical file.
Input Data: dom = ~/dir1, sub = ~/dir2
Expected Behavior: Program will do nothing.
Actual Behavior: Program did nothing.
Result: PASSED

Test Case 4:
Verifies Software Requirement(s): 3
Description: Verifies that if a file is in sub that is not in dom, it will be deleted.
Input Data: dom = ~/dir1, sub = ~/dir2
Expected Behavior: Program will delete the file in ~/dir2 that is not in ~/dir1.
Actual Behavior: Program deleted file in ~/dir2.
Result: PASSED

Test Case 5:
Verifies Software Requirement(s): 3
Description: Verifies that if a directory is in sub that is not in dom, it will be deleted.
Input Data: dom = ~/dir1, sub = ~/dir2
Expected Behavior: Program will delete the directory in ~/dir2 that is not in ~/dir1.
Actual Behavior: Program deleted directory in ~/dir2 that was not in ~/dir1.
Result: PASSED

Test Case 6:
Verifies Software Requirement(s): 4
Description: Verifies that if a file in dom is newer than a nearly-identical file in sub, the file in sub will be replaced by the one in dom.
Input Data: dom = ~/dir1, sub = ~/dir2
Expected Behavior: The older file in ~/dir2 will be replaced by a copy of the file in ~/dir1.
Actual Behavior: The older file in ~/dir2 was replaced with the corresponding file in ~/dir1 that was newer.
Result: PASSED

Test Case 7:
Verifies Software Requirement(s): 5
Description: Verifies that if a directory doesn’t exist, then it will output the error and terminate.
Input Data: dom = ~/dirdoesntexist
Expected Behavior: Program will print that ~/dirdoesntexist is an invalid directory and will then exit.
Actual Behavior: Program output that ~/dirdoesnotexist was invalid and exited.
Result: PASSED

Test Case 8:
Verifies Software Requirement(s): 6
Description: Verifies that program will NOT replace a priority file in the subordinate directory with its corresponding file in the dominant directory if force option is NOT passed.
Input Data: dom = ~/dir1, sub = ~/dir2
Expected Behavior: Program will output that priority file is in subordinate directory but is not replacing it. File will not be replaced.
Actual Behavior: Program output that priority file was in subordinate directory and that it is not replacing it. File was not replaced.
Result: PASSED

Test Case 9:
Verifies Software Requirement(s): 6
Description: Verifies that program WILL replace a priority file in the subordinate directory with its corresponding file in the dominant directory if force option IS passed.
Input Data: dom = ~/dir1, sub = ~/dir2
Expected Behavior: Program will output that priority file exists in subordinate directory and that it is replacing it. File will be replaced.
Actual Behavior: Program output that priority file existed in subordinate directory and that it would replace it. File was replaced.
Result: PASSED