Group 2

NATracker

Journaling File System

Miles Calloway, Zach Stofko, Daniel Finn

Brief Introduction

Purpose: Build a file / directory journaling tool for tracking changes and replaying edits through a user-friendly interface.

Team Members and Roles:

- **Miles Calloway:** Backend, Replay, Folder Tracking.
- **Zach Stofko:** Frontend (GUI), Replay and Folder Tracking Implementation.
- **Daniel Finn:** Install.sh, Uninstall.sh, GUI Settings, Term Project Presentation.

Installation Process (How it Works)

- Users will download the **Install.sh** script from the **Github Repository**.
- It can be run from the **terminal** in any location using the command

  sudo bash Install.sh
- Running **Install.sh** will install the **NATracker** program and all necessities into the **opt** folder. It also installs dependencies like Python 3, GTK3, and iNotify.
- Users can use the program by going to applications and selecting the newly-added **NATracker** application. The icon is shown here:

Install.sh

```
# install necessary Python and GTK dependencies
sudo apt install python3 -y
sudo apt install python3-gi gir1.2-gtk-3.0 -y
sudo apt install pip -y
sudo pip install inotify_simple --break-system-packages
sudo pip install dill --break-system-packages
sudo apt install dbus -y # install dbus to fix issue with GUI
sudo apt install dbus-x11 -y
```

Installing dependencies mentioned before like Python 3, iNotify, and the GTK3 Python Wrapper.

```
# clone repo in to opt folder on the computer
git clone "$REPO_URL" /opt/NATracker
if [ $? -eq 0 ]; then
```

Cloning the GitHub Repository on the user's machine in the opt folder.

```
# create the desktop icon
ICON="/usr/share/applications/folderTrackerGUI.desktop"
echo "[Desktop Entry]
Version=0.0
Type=Application
Name=NATracker
Exec=bash FolderTrackerGUI -desktop
Icon=/opt/NATracker/GUI/icon2.png
Terminal=false
Categories=Utility;Application;" > "$ICON"
```

Creating the application icon for the user so they can click on it to open the program.

Core Backend Functionality

Watchers.py:

- Handles journal creation and updates for each tracked directory.
- Functions to add and remove watcher locations to a database / config file.
- Uses inotify_simple to monitor changes in real-time.
    - This is a python wrapper for Inotify
- Journals are stored in hidden folders, one for each tracked file.

ThingThatWillRunOnStartup.py:

- Automatically starts folder watchers after system reboot.
- Initializes all configured watchers by spawning processes for each.

Watchers.py

```python
def loadWatchers():
    #This may happen on first usage.
    if not os.path.exists("/etc/opt/NATracker/watchers.pkl"):
        if not os.path.exists("/etc/opt/NATracker"):
            os.mkdir("/etc/opt/NATracker")
        watchersD = allWatchers()
        watchersD.watchers = []
        return watchersD
    #This is the normal case.
    with open("/etc/opt/NATracker/watchers.pkl", "rb") as f:
        return pickle.load(f)

def checkForWatcher(location):
    #strip path of any trailing slashes/spaces

    if not os.path.exists(location):
        return None
    #load the watchers
    watchersD = loadWatchers()
    #check if this is already being watched
    for watcher in watchersD.watchers:
        if watcher.location == location:
            return True
    return False
```

loadWatchers:

- Loads watcher configs from file in /etc/opt/NATracker/

- If there is no watcher structure it creates it.

checkForWatcher:

- checks if a directory exists and if it is already being monitored.

```python
def addWatcher(location, runwatcher = True):
    exsistingWatcher = checkForWatcher(location)
    if (exsistingWatcher == True):
        print(f"Directory {location} is already being watched.")
        return False
    if (exsistingWatcher == None):
        print (f"Directory {location} does not exist.")
        return False
    watchersD = loadWatchers()

    #add the watcher
    thisWatcher = Watcher()
    thisWatcher.location = location
    watchersD.watchers.append(thisWatcher)
    #save the watchers
    try:
        #make /.NATracker in the dir
        if not os.path.exists(location + "/.NATracker"):
            os.mkdir(location + "/.NATracker")
        #copy /opt/NATracker/ScriptForFolder/WatchThisFolder.py to dir/.NATracker/WatchThisFolder.py
        with open("/opt/NATracker/ScriptForFolder/WatchThisFolder.py", "r") as f:
            with open(location + "/.NATracker/WatchThisFolder.py", "w") as f2:
                f2.write(f.read())
        with open("/etc/opt/NATracker/watchers.pkl", "wb") as f:
            pickle.dump(watchersD, f)
    except:
        print("Failed to save watcher. Maybe not root???")
        return
    #as subprocess
    if runwatcher:
        subprocess.Popen(["python3", location + "/.NATracker/WatchThisFolder.py"], stdin=None, stdout=Non
def removeWatcher(location):
    exsistingWatcher = checkForWatcher(location)
    if (exsistingWatcher == False):
        print(f"Directory {location} is not being watched.")
        return False
    if (exsistingWatcher == None):
        print (f"Directory {location} does not exist.")
        return False
    watchersD = loadWatchers()
    #remove the watcher
    for watcher in watchersD.watchers:
        if watcher.location == location:
            watchersD.watchers.remove(watcher)
            if os.path.exists(location + "/.NATracker/WatchThisFolder.py"):
                os.remove(location + "/.NATracker/WatchThisFolder.py")
    #save the watchers
    try:
        with open("/etc/opt/NATracker/watchers.pkl", "wb") as f:
            pickle.dump(watchersD, f)
    except:
        print("Failed to save watcher. Maybe not root???")
        return
```

addWatcher:

- Creates a .NATracker subdirectory in the specified path.

- Copies the monitoring script “WatchThisFolder.py” to the NATracker.

- Starts the monitoring script as a subprocess

removeWatcher:

- Removes a watcher and its associated data.

- Deletes the NATracker/WatchThisFolder.py file for the directory.

- Updates and saves the watcher list

Key Features

Now that we have installed NATracker, we can review some of the key features of the application.

- During installation, a **cron job** was created so our program is always running in the background and tracking changes in **real-time**.
- The user can choose to track a folder, which will create journals for all .txt files in the directory.
- The user can also choose to replay a text file, which will create a file with changes specified.
- Lastly, the user can choose to uninstall the program from the settings in the GUI interface. This includes removing background processes and all tracking.

Folder Tracking

Folder tracking relies on three files.

1. **WatchThisFolder.py**
   - Monitors directories for changes in real-time
   - Creates and updates journals for all .txt files in a tracked folder.
   - Logs events such as file creation, modification, or deletion.
2. **TrackerSetup.py**
   - Responsible for initializing and configuring folder tracking.
3. **tracker.py**
   - Integrates folder tracking functionality into the GUI by calling on functions from the other two files above.

WatchThisFolder.py

```
inotify = INotify()
watch_flags = flags.CREATE | flags.DELETE | flags.MODIFY | flags.DELETE_SELF | flags.MOVED_TO | flags.MOVED_FROM
wd = inotify.add_watch(currentDir, watch_flags)
```

This keeps a journal to record changes in text files and keeps watch for any changes in a directory, like creation, modification, or the deletion of a file.

TrackerSetup.py

```
# called by GUI to add tracking to a folder
def addTracking(directory, RunWatcher):
    #check that the directory exists
    if not os.path.exists(directory):
        print(f"Directory {directory} does not exist.")
        return
    returnStatus = Watchers.addWatcher(directory, RunWatcher)
    if returnStatus == False:
        print("Error adding tracking.")
        exit(1)
    else:
        print("Tracking added for " + directory)
```

Functions like this exist to be called on the GUI.

tracker.py

- tracker.py implements the folder tracking functionality, allowing users to manage their tracked folders by adding tracked directories, removing directories, or removing all tracked folders with the click of a button. This code will also display a list of tracked folders.

```python
def on_add_directory_clicked(app):
    dialog = Gtk.FileChooserDialog(
        title="Select Directory to Track",
        parent=app,
        action=Gtk.FileChooserAction.SELECT_FOLDER,
    )
    dialog.add_buttons(Gtk.STOCK_CANCEL, Gtk.ResponseType.CANCEL, Gtk.STOCK_OPEN, Gtk.ResponseType.OK)

    response = dialog.run()
    if response == Gtk.ResponseType.OK:
        directory = dialog.get_filename()
        if directory not in app.tracked_folders:
            add_tracking(app, directory)
        else:
            show_error_message(app, "Duplicate Folder", "The folder is already being tracked.")
    dialog.destroy()

def on_remove_directory_clicked(app):
    selected_rows = app.folder_list_box.get_selected_rows()
    for row in selected_rows:
        directory = row.get_child().get_text()
        remove_tracking(app, directory)

# new function to remove EVERY tracked folder
def on_remove_all_folders_clicked(app):
    # loops through all tracked folders and removes them
    for folder in app.tracked_folders.copy():
        remove_tracking(app, folder)

    # updates GUI
    app.remove_button.set_sensitive(False)  # disable button after removal
    show_tracked_folders(app)  # update the list to show it's empty
```

Code for functions that are called when the designated buttons are pressed. An example would be on_add_directory_clicked, which creates a file explorer window so the user can locate the folder they wish to be tracked. This file explorer is referred to as a type of “dialog” in GTK3.

```python
def add_tracking(app, directory):
    subprocess.run(
        ["python3", "/opt/NATracker/TrackerSetup.py", "--dir", directory, "--DontRunWatcher"],
        capture_output=True,
        text=True,
    )
    subprocess.Popen(
        ["python3", directory + "/.NATracker/WatchThisFolder.py"],
        stdin=None,
        stdout=None,
        stderr=None,
        close_fds=True,
        start_new_session=True,
    )
    show_tracked_folders(app)

def remove_tracking(app, directory):
    subprocess.run(
        ["python3", "/opt/NATracker/TrackerSetup.py", "--dir", directory, "--remove"],
        capture_output=True,
        text=True,
    )
    show_tracked_folders(app)
```

These functions call on code from TrackerSetup.py (which was previously mentioned) to add and remove tracking from folders.

Replaying

Replay.py handles all functionality for replaying file changes in the GUI.

1. It loads journal entries and allows the user to select specific timestamps for reconstruction.
2. Then, replay.py reconstructs the file to its state at a chosen time stamp using the stored journal data.
3. Lastly, it provides options to view and save the reconstructed file through the GUI.

replay.py

```
# called to create the text file but only up to a specified change
def recreateUpToEntry(diffJournal, entry):
    file = {}
    #insert contents before diff
    contentsBeforeDiffSplit = diffJournal.contentsBeforeDiff.splitlines()
    for line in range(0, len(contentsBeforeDiffSplit)):
        file[line] = contentsBeforeDiffSplit[line]
    for change in diffJournal.JournalEntrys:
        returnOnNext = False
        for journalEntry in change:
            if journalEntry[2]:
                file[journalEntry[0]] = journalEntry[1]
            else:
                file[journalEntry[0]] = ""
            if change == entry:
                returnOnNext = True
        if returnOnNext:
            return dictToString(file)
    return dictToString(file)
```

This function takes in the journal of the text file you want to replay along with the timestamp you wish to revert back to and returns the recreated text file.

Uninstallation Process

Through the settings in the GUI, users can access the uninstall tools, which will completely remove NATracker from the system. This includes:

- Deleting all tracking for the folders and their hidden .NATracker subdirectories.
- Removing dependencies like inotify_simple.
- Cleaning out cron jobs and symbolic links created during installation.

Uninstall.sh

Everything that the Install.sh script does is undone by our Uninstall.sh script. All files, cron jobs, symlinks, and dependencies added during the installation, along with our watched folders are removed during the Uninstall.

Main steps:

1. Root privilege check (needed to run)
2. Removal of tracked folders
3. Clean out and remove all directories
4. Removal of the cron jobs (to delete startup scripts)
5. Uninstall of python dependencies (ex) inotify_simple)

```
(sudo crontab -l | grep -v "$CRON_JOB") | sudo crontab -
echo "removed NATracker cron job from crontab."
```

```
# defining main paths used for uninstall
INSTALL_DIR="/opt/NATracker" # main install dir
SYMLINK_PATH="/usr/local/bin/NATracker" # command line access
SYMLINK_GUI_PATH="/usr/local/bin/FolderTrackerGUI" # for the gui
CRON_JOB="@reboot python3 \"$INSTALL_DIR/ThingThatWillRunOnStartup.py\""
WATCHERS_DIR="/etc/opt/NATracker"
PICKLE_FILE="$WATCHERS_DIR/watchers.pkl" #tracked folder
python3 - <<EOF

import pickle
import os

pickle_file = "$PICKLE_FILE" # path to pickle file w tracked data
if os.path.exists(pickle_file): # to make sure it exists alr
    with open(pickle_file, "rb") as f: # open and load
        data = pickle.load(f)
        for watcher in data.watchers:
            folder = watcher.location # get tracked folders location
            tracker_path = os.path.join(folder, ".NATracker")
            if os.path.exists(tracker_path):
                print(f"removing tracked folder: {tracker_path}") # shows removal
                os.system(f"rm -rf \"{tracker_path}\"") # removes the folder
EOF
```

Demonstration Time!