Assessment 2 - Scripting for System Automation COMP9053

Overview
You must design and develop a tool called *timemachine* that takes copies of specified files so that it builds a history of their changes over time.

To complete the assignment you must submit:
- Any required source files
- Any required configuration files
- A file called designdoc.txt containing a rough overview of your program and justification for any design decisions you made

Assignment Specification
You must design and develop a tool called *timemachine* that takes a copy of specified files so that it builds a history of their changes over time. It should take an initial copy when the file is first added to the configuration or the script is first run and then it should take a copy each time it checks *only if the file has changed since the script last checked*. See the box below to obtain a file's modification time.

The tool must support watching and taking copies of multiple files. *(10 marks)*

The tool must read the list of files to observe from a configuration file. And the configuration file must be editable by the user. *(10 marks)*

The tool must be able to take as an optional command line argument the location of the configuration file. If this is not provided it should default to reading from config.dat in the directory the script is run from. *(10 marks)*

The tool must be able to take as an optional command line argument the path to a directory where the copies and any information about the copies should be stored. If this option is not provided, the tool must default to using the directory the script is run from. *(10 marks)*

The tool must log important actions. *(10 marks)*

The tool must gracefully handle errors e.g. due to bad or missing data. *(10 marks)*

The tool must check files once a minute and take a copy of any which have changed. *(5 marks)*

The tool must be robust to starting and stopping. *(5 marks)*

It's probably wise to add this part last, and only if you get everything else working.
The tool should provide a way to modify the configuration via the command line. E.g.
`python3 timemachine.py add somefile` should add *somefile* to the list of files to be observed.
`python3 timemachine.py remove somefile` should remove *somefile* from the list.
`python3 timemachine.py list` should print a list of the files being observed. *(10 marks)*

There are also some stylistic considerations:

The main file to run must be called *timemachine.py.*

The program should be broken into functions where appropriate. *(5 marks)*

The code must be commented. Don't add trivial comments like 'iterate over items in list' - instead comment on the intent of the code and anything non-obvious to a reader. *(5 marks)*

As well as your code you must provide a design document, in a file called *designdoc.txt*. This document should explain the different design decisions you made while developing your program. For instance, if the description above was incomplete and required you to assume something you should list the assumption. If there were some different ways to implement a particular feature you should list the options and explain why you chose the option you did. You should at least justify each module you import.
It should document failure modes and what the script will do in each case.
It should also include some steps you would take when your tool is complete to check that it works correctly.
(10 marks)

Note: you should only use Python libraries and structures encountered in the lectures, exercises or this assignment. You may marks for sections implemented in some other way. This is to discourage copying and pasting of publicly available code.

This assessment will be worth 30% of your final grade

Due before midnight November 25th, 2019.

Standard institute penalties apply for late submissions

Getting file modification time

You can obtain the modification time of a file using `os.path.getmtime()`. This is returned as a Unix timestamp with the number of microseconds appended. You can convert it to a `datetime` using `datetime.datetime.fromtimestamp()`.
Here's an example that gets the modification timestamp for a file, prints it, converts it to a `datetime` and prints that:

```
import datetime import
os

timestamp=os.path.getmtime('/home/admin/alert.sh')

print(timestamp)
print(datetime.datetime.fromtimestamp(timestamp))
```

Scheduling a file to run with cron

We learned about *cron* in lecture 3 but we didn't go into it in great detail. It's used to schedule programs to run at set times e.g. once an hour, once a week etc.
Each user has their own schedule, called a crontab, and you can have multiple entries here. cron will read your crontab and run any programs you have scheduled at the specified times.
When the programs run, their work directory will default to your home directory, so it they read or write a file without a fully qualified path then it's your home directory that will be assumed. So if you e.g. write a log file from a script and don't specify a full path then it'll end up in your home directory.
Note that you can't assume that environment variables that are set when you log in are set when cron runs your programs. So you should make sure your script has any information it needs.
In particular, you can't assume the $PATH variable will be the same. Remember, when you type the name of a program into the shell, it searches the directories in the $PATH to find a program that matches the name. If the $PATH is not what you expect then the program you want to run might not be found, or a different one might be run. So be sure to fully qualify what you need.
Each crontab entry is comprised of a schedule and a program to run. The schedule is compose of 5 fields (6 on some OSs but 5 is the standard) that determine at what minutes, hours, days etc your program should be run.
For this assessment you only need to schedule something to run once every minute and that schedule is described with 5 asterisks, `* * * * *`.
Here's the example entry from lecture 9 (paths changed a little for brevity):
`* * * * * /usr/bin/python3 /home/admin/exercise lecture9.py c /home/admin/config.dat -a /home/admin/alert.sh > /dev/null`
Note that I even fully qualified the access to python3, just in case. It's a good habit to get into.
You can list your current crontab using `crontab -l`.
You can edit your crontab by typing crontab -e and then enter your line at the bottom (e.g. the underlined entry above). You can also set your crontab entry by either piping the entries you want into crontab (`echo '* * * * * /usr/bin/python3 /home/admin/exercise_lecture9.py -c /home/admin/config.dat -a /home/admin/alert.sh > /dev/null' | crontab`) or you can store your entries in a file and redirect it into crontab (e.g. `crontab < yourcrontab.txt`). Note the last two methods will overwrite any existing entries.
You can find more info on configuring a crontab here: http://alvinalexander.com/linux/unix-linux-crontab-every-minute-hour-day-syntax