Backup Python

Gabriel Shelton

Problem

I have and see regularly a problem of over-cluttered folders with projects years old. I hope to have this series of scripts solve that by compiling those projects into a compressed and eventually encrypted format. As well I would like the utility of having a more live backup system that syncs local copies and branches in git, without it ever having to go to GitHub.

Proposal

I propose a python module that serves to hold a backup script and an outer driver script that will unpack the command line parameters that are passed and send it to the module. Then adjacent to the driver, or in a selected directory (through a configuration file, likely JSON format), will be the stored directory where the backup's are stored.

Eventually after the main backup module script is setup, in addition, I will setup a NodeJS server that will receive requests and forward them to the backup driver in the proper format, after sanitizing them.

Next, will be a sub-module that will handle communication with the NodeJS server and allow the user to communicate and send requests, sanitation will happen here as well, before the request is sent.

Finally, encryption and multi-user support will be added for the server-client, and I will implement an archive system that will compress the backup's down and store them way back; then will delete them from the client's computer.

Timeline

Feature	**Start-Date**	**End-Date**	**Notes**
Initial Setup	2018-12-11	2018-12-12	
Local Backup	2018-12-13	2018-12-14	
Server Middleman	2018-12-15	2018-12-16	
Client → Server			
Encryption: Client → Server			
Archive & Encryption			

Time-Log

Date	Start-Time	End-Time	Total-Time	Work (Feature / Topic)
2018-12-11	21:30	22:21	.85 hours	Initial Setup
2018-12-15	14:54	19:30		Configuration File / Command Pattern setup
2018-12-21	18:43	19:21		Abstract factory to build commands.
2018-12-24	03:00	03:33		Write logic to get all of the workspaces from the workspace names.

Script

The script will have a set of commands that will be parsed one word at a time and go down towards an abstract factory and command pattern. Once the right command has been identified, it will be constructed and ran. The script code will do backups of local code only with rsync, although it might be possible for it to do remote connections because of rsync's robustness.

The executable will take in two parameters: [action, workspace], the action will decide what kind of command is made. The workspace will be passed as a parameter to the constructor to be operated on.

The workspace-name refers in the configuration file to a workspace-path that the script can use. Optionally you can choose all to do all workspaces.

script/test.py will hit the code with the five standard actions + 1 more for each with all as the workspace, then several invalid inputs, as well as possible shell injections.

Script Commands

Executable	Action	Workspace [all]
backup-script → script/main.py	push	workspace-name
...	pull	...
...	sync	...
...	archive	...
…	unarchive	...