Executable Client for Automating Large Implementations of Back-end Units Remotely (ExCALIBUR)

Anna Karlhede
Joar Nykvist

June-August 2017

Contents

1 Introduction

`ExCALIBUR` is a Python-written program designed to log on to several Dell servers, one by one using their iDRAC. Based on the MAC address for each server, the IP-addresess and the interface names are found. An ifcfg-file is then placed in the according folder with appropriate text on each server.

We advice to read this whole document before using the program or developing it further.

2 Execution

Run the program from the bash terminal when placed in the `ExCALIBUR` directory, the command is "`python Main.py en.fifo en2.fifo`".

1. A new terminal window for login to the iDRAC will open, enter the password here and then switch back to the orignial terminal window and press Enter to send the `connect` command.
2. Wait for about three seconds and then press enter twice, this will send the username and the password.
3. Press enter again to change directory.
4. Press enter twice two find the interface name.
5. Press enter four times to check if the text file already exists in the directory.
6. Press enter three times to create the file and write text to it.
7. Press enter to change directory.
8. Press Enter to logout from the server.
9. Press Enter to logout from the iDRAC.
10. Repeat steps 1-9 for the next server.

3 Classes, Methods and Attributes

3.1 Class `Server`

A `Server` object is used to store information about one specific server. It has attributes:

- `macAddress`
  - The Mac address for the server.
- `serverName`

  - The DNS name of the server.

- `interfaceName`
  - The name of the interface for the server.
- `ipAddress`
  - The IP address for the server.
- `strippedIp`
  - A stripped version of the IP address that is meant to be written in a textfile on the server along with other information to bring the server up if down.

3.2 Class `AllServers`

The `AllServers` class provides methods and attributes that allow a user to loop through a list of server names and create a text file on each one with necessary content for bringing it up.

3.2.1 Attributes

- `ipCases`
  - A `list` object containing each one of the possible formats an IP address for the servers we are interested in can have.
- `SMIpList`
  - A `list` object containing three `list` ojbects that in turn hold server names, MAC addresses and IP addresses for the servers we are interested in.
- `fileText`
  - A `string` object holding the text that is meant to be written into a file on the server.
- `currentServer`
  - A `server` object representing the server that is currently being iterated over.
- `IpCasesFile`
  - A `string` object holding the full path to a textfile that contains information about the three possible IP address formats for the servers we are interested in.
- `ServerMacIp`

  - A `string` object holding the full path to a textfile that contains the server name, MAC address and IP address for the servers of interest.

- `ECUT`
  - A `string` object holding the full path to a textfile that contains the text that is to be written to a file on the server.
- `channelToTerminal`
  - A `file` object that redirects user input either to a named pipe or to standard output.
- `channelToPipe`
  - A `file` object that represents a pipe from which user input is read.
- `pipeList`
  - A `list` object that holds the two pipes that are used in the program.
- `possibleExistence`
  - A `string` object that is used to find out whether there is already such a file that we are trying to create on the server.
- `booleanInit`
  - A `bool` object used to for a `while` loop to allow pipe communication. This foes for `booleanEx` and `booleanFin` as well for other `while` blocks.
- `booleanEx`
- `booleanFin`
- `iCounter`
  - A `integer` object used to loop through the list of servers.
- `bStop`
  - A `bool` object used to know whether execution should continue for the current server in the specific cycle of the exterior `while` loop.

3.2.2 Methods

- `readIpCases`
  - Reads `IpCasesFile` and sets `ipCases`.
- `setCurrentServer`
  - Sets `CurrentServer` to an empty `Server` object.
- `stripIp`
  - Takes the IP address from the `currentServer` and converts it to a format that is better for comparison with the possible types of IP addresses. Sets the `strippedIp` attribute for the `currentServer` of this cycle.
- `createSMIpList`
  - Reads `ServerMacIp` and sets `SMIpList`.
- `checkIp`
  - Compares the `strippedIp` attribute of this cycle's `currentServer` object to the possible types of IP addresses. Calls the `wannaContinue` method if there is no match.
- `prepareText`
  - Opens `ECUT` for reading, takes the text and enters server-specific details, finally sets the `fileText` attribute.
- `createFile`
  - Calls the `prepareText` method, waits for the user to press Enter, and then uses the Linux `touch` command to create a file. Finally writes the text provided by the `fileText` attribute to this file using the Linux `echo` command.
- `doInitialThings`
  - Opens a new thread with the `interfaceFun` methods as the target, the thread is closed at the end of this function. Executes sequence to log on to the server.
- `interfaceFun`
  - Finds the interface name for the current server by piping the MAC address to the Linux command `ifconfig`. Uses the `macAddress` attribute of the `currentServer` object for the current cycle.
- `getIpAddress`

  - Finds the IP address for the current server based on the corresponding MAC address by executing and then reading output from the `nslookup` command. Sets the `ipAddress` attribute of the `currentServer` object for the current cycle.

- `doesFileExist`
  - Starts a thread with the `fileFun` method as its target, this thread is later closed. Changes directory to where the textfile is to be placed by sending `cd` along with the path after having moved up one level with "`cd ..`". Compares the `possibleExistence` attribute to a `string` with the name of the file we are trying to create. If the names match it means the file already exists and an `Exception` is thrown.
- `fileFun`
  - Pipes the name of the file we are trying to create to the `ls` command, reads the result and sets the `possibleExistence` attribute to this.
- `doFinalThings`
  - Starts a thread with the `finFun` function as its target and finally then closes it.
- `finFun`
  - Executes the commands for logging out of the server.
- `prepareOnceForAll`
  - Calls `readIpCases` and `createSMIpList`.
- `wannaContinue`
  - Asks the user if they wish to terminate the program or proceed to the next server. Sets `bStop` to `False` and calls `doFinalThings`. If the user wishes to terminate the programl, `iCounter` is set to the length of `SMIpList` plus one, to make sure the `while` loop in `main` is exited.
- `doAllThings`
  - Calls `getIpAddress` and `stripIp`. If `bStop` is `False`, `checkIp` `doInitialThings`, `doesFileExist`, and `createFile` are run. Either way, `doFinalThings` is run at the end.

3.3 Class `Main`

The `Main` class creates an `AllServers` object and configures the necessary text files, runs the `prepareOnceForAll` and the `setCurrentServer` methods of the `AllServers` class, starts a `while` loop based on the condition that the value of the attribute `iCounter` is smaller than the length of `SMIpList`, both attributes of the `AllServers` class. In this loop, the `doAllThings` method of the `AllServers` class is run for each server of interest. Earlier on, two named pipes are created for reading output from and writing input to the terminal, and connection to the iDRAC is established via SSH.

4 Bugs, Problems and Fixes

A few complications remain in the program, we have worked hard to solve them but not had success. Some problems have their origin in the program being developed on a user without sudo rights, some derive from difficulties with the iDRAC interface. Below we describe the problems we see in the program, and our thoughts on how to solve them.

4.1 Additional Terminals not closing

For each cycle in the `while` loop in `Main`, a new terminal window is opened. When working on this tool, we did not manage to find a way to close these.

4.1.1 Fix

Our recommendation for fixing this problem is to simulate an `ALT + F4` press, or an `ALT +` \ press, after shifting to this window temporarily. We recommend this to be done well after the log on procedure is completed to not interfere. But at the same time, well before log on to the next server is to be done.

4.2 Terminal not responding

It happens occasionally that the interface supplied after the `connect` command is sent does not respond to input from the program. This is not an error in the programming, but something that can happen when connecting manually as well. Clearly, the program flow is corrupted when this happens, and it usually corrupts the flow for the next server on the list as well.

4.2.1 Fix

We recommend to fix this by reading the output supplied from the terminal at this stage, and to alter the program such that it only proceeds if the terminal is responding, where this is defined by the output displayed. We think that the best thing to do if the terminal does not respond is to break the program and start over again.

4.3 Failure to run the Program

At some points we had issues running the program at all from the terminal.

4.3.1 Fix

We learned that this can be fixed by removing the `en2.fifo` file in the folder and letting the program create it, which is one of the first things it does if it finds that the file does not already exist.

4.4 Text Files placed in unexpected Directories

In the cases where there is a problem with log on, the procedure to create and save the text file is often still carried out, sometimes leaving a text file in an unintended directory.

4.4.1 Fix

It is our idea that this should be mended by providing enhanced error handling, such that the file is only created if all previous steps have been successful.

4.5 Exceptions do not break the Program

At several places in the code there are exceptions placed to catch errors that we know might happen here, and then handle them the way we deem appropriate. In some cases however, especially if the exception is thrown from or caught inside a thread, the program proceeds through a few lines after the exception.

4.5.1 Fix

We believe that the key to solving this issue is finding a way to close the threads when an exception rises, threads are tricky and seemingly impossible to close from within, but we think that if communication between the two threads can be established, this problem is not difficult to resolve.

5 Recommended Future Development

We highly recommend that the program flow is made fully automatic, meaning moving away from the approach where the sequences are controlled by Enter-presses. We strongly believe this is possible, and that the way to achieve this is reading terminal output, implementing pauses in appropriate places using `time.sleep(int)`, especially before logging on after having supplied the `connect` command.

6 Recommended Tools

Below we list some tools that we think can be useful for further development of `ExCALIBUR`.

`https://github.com/openstack/python-dracclient`
A client made for using Python to manage machines with iDRAC cards. We did not use this client because the documentation was poor, but we think that it can be of great help if documentation is added in the future. We recommend to keep an eye out.

`https://github.com/apcros/Drac-Perl`
A client made for managing iDRAC-equpped machines from programs written in the `Perl` programming language. The install does not work currently, but we had a dialogue with the owner and they plan to fix this. We believe that it could be helpful to write a script in Perl and call this script from the `Main.py` if the install is fixed. The owner responds quickly to emails and is very accommodating.

`https://pexpect.readthedocs.io/en/stable/`
We thing that Pexpect can be a good alternative way of handling SSH and login.

`https://linux.die.net/man/1/pkill` Pgrep and pkill are Linux commands used for handling processes. We think they can be useful for closing the superfluous terminal windows.

`http://www.linuxjournal.com/article/2156`
We use named pipes to communicate to the terminal. This link provides a description of what the yare and how they work.

Note: Do not push to Github repository before removing password from code.