ACL: Automated Command Line

Introduction

ACL is an interpreted language designed to automated repetitive tasks on the command line. The interpreter will be based on Java mainly because existing implementation of Expect suffers from the problems of C, where you need a separate interpreter for every OS. Using Java will ensure that the interpreter is useful regardless of the OS used and be able to run it whenever a JVM for the OS exist.

Design

Here are some design features of ACL:

1. Simple and easy to use – The syntax and semantics of ACL would be like a hybrid between Expect and C/Java. Efforts will be made to ensure that the language is intuitive and easy to read and understand.
2. Cross platform compatibility – Able to run ACL on every platform where a JVM exist. This is a lofty goal and efforts will be made to ensure that this goal is met.
3. Interpreted – This will ensure that it is fast and easy to write and test ACL scripts.

Keywords

Here are the initial proposed keywords of ACL:

1. **session** – To spawn a new telnet/SSL/… session.
2. **receive** – To wait for a certain sequence of characters from the spawn session.
3. **send** – To send a certain sequence of characters to the spawned session.
4. **interact** – To place the user in interactive mode with the spawned session so that the user can type and interact directly with the spawned session
5. **if, else, while, for** – Used for flow control in the script. Structure is very similar to C/Java.
6. **function** – To define a function.

Variables

There are only two types of variables currently supported:

1. **string** – i.e., a = “a string”
2. **integer** – i.e., a = 123

Programming Examples

Here is a simple program to login to 2 layers of server, send 2 commands and return control to the user:

```
---
function login_ssh (server_name, user, password) {
        session "ssh server_name"

        receive "Login:"
        send user
        receive "Password:"
        send password
}

user = "john"
password = "secret"

login_ssh(“first_server”, user, password)

receive "$" // Wait for the prompt "$"
send "ssh second_server"
receive "Login:"
send user
receive "Password:"
send password

receive "$"
send "first command"
receive "$"
send "second command"

interact
----
```

Here is a simple program to automate killing a monster in a text-based MUD (Multi-User Dimension) game:

```
function login_mud (server_name, user, password) {
        session "ssh server_name"

        receive "Login:"
        send user
        receive "Password:"
        send password
}

function walk_to_monster() {
        receive "#"
        send “north”
        receive "#"
        send “west”
        receive "#"
        send “west”
        receive "#"
        send “north”
        receive "#"
        send “north”
        receive "#"
        send “east”
}

function walk_to_sleep() {
        receive "#"
        send “west”
        receive "#"
        send “south”
        receive "#"
        send “south”
        receive "#"
        send “east
        receive "#"
        send “east
        receive "#"
        send “south
}

user = "ultimate_killer"
password = "i_kill_for_fun_and_enjoyment"

login_mud(“mud_server”, user, password)
```

```
for (i = 1; i < 100; i++) {
        walk_to_monster()
        receive "#"
        send “kill goblin”
        receive "goblin is DEAD!!”
        walk_to_rest()
        receive "#"
        send “sleep”
        receive "You are fully rested."
}
----
```