

Exercise 07 - Remote Commands - Script 6

Goal:

The goal of this exercise is to create a shell script that executes a given command on multiple servers.

Scenario:

The number of systems you manage is growing and you need a way to quickly execute the exact same command on all of your systems. Because it takes too much of your time to type the same command on every single system you manage, you decide to write a script that will do this for you.

Shell Script Requirements:

You think about what the shell script must do and how you would like it operate. You come up with the following list.

The script:

- Is named "run-everywhere.sh".
- Executes all arguments as a single command on every server listed in the /vagrant/servers file by default.
- Executes the provided command as the user executing the script.
- Uses "ssh -o ConnectTimeout=2" to connect to a host. This way if a host is down, the script doesn't hang for more than 2 seconds per down server.
- Allows the user to specify the following options:
 - -f FILE This allows the user to override the default file of /vagrant/servers. This way they can create their own list of servers execute commands against that list.
 - -n This allows the user to perform a "dry run" where the commands will be displayed instead of executed. Precede each command that would have been executed with "DRY RUN: ".
 - -s Run the command with sudo (superuser) privileges on the remote servers.
 - -v Enable verbose mode, which displays the name of the server for which the command is being executed on.
- Enforces that it be executed without superuser (root) privileges. If the user wants the remote commands executed with superuser (root) privileges, they are to specify the -s option.
- Provides a usage statement much like you would find in a man page if the user does not supply a command to run on the command line and returns an exit status of 1. All messages associated with this event will be displayed on standard error.
- Informs the user if the command was not able to be executed successfully on a remote host.
- Exits with an exit status of 0 or the most recent non-zero exit status of the ssh command.

Create Three Virtual Machines:

First, start a command line session on your local machine. Next, move into the working folder you created for this course.

```
cd shellclass
```

Initialize the vagrant project using the usual process of creating a directory, changing into that directory, and running "vagrant init". We'll name this vagrant project "multinet".

```
mkdir multinet
cd multinet
vagrant init jasonc/centos7
```

Configure the Virtual Machines

Here are the details on the three virtual machines that you are going to create:

Name	IP Address
admin01	10.9.8.10
server01	10.9.8.11
server02	10.9.8.12

Edit the Vagrantfile and create three stanzas of configuration. One for each virtual machine.

```
config.vm.define "admin01" do |admin01|
  admin01.vm.hostname = "admin01"
  admin01.vm.network "private_network", ip: "10.9.8.10"
end

config.vm.define "server01" do |server01|
  server01.vm.hostname = "server01"
  server01.vm.network "private_network", ip: "10.9.8.11"
end

config.vm.define "server02" do |server02|
  server02.vm.hostname = "server02"
  server02.vm.network "private_network", ip: "10.9.8.12"
end
```

Start the Virtual Machines and Log into admin01

Now you're ready to start the VMs and connect to it admin01.

```
vagrant up  
vagrant ssh admin01
```

Add Host Entries for server01 and server02

You can manually edit the /etc/hosts file using root privileges or use the following commands to add the lines to the /etc/hosts file.

```
echo 10.9.8.11 server01 | sudo tee -a /etc/hosts  
echo 10.9.8.12 server02 | sudo tee -a /etc/hosts
```

Ensure You Can Ping the Virtual Machines by Name

Use the ping command to ensure you can communicate to the virtual machines by name. You want to see a reply from the IP address of 10.9.8.11 from server01 and 10.9.8.12 from server02. If that is not the case, correct the /etc/hosts entries and/or make sure the virtual machines are running.

Here is an example. (Portions typed are in bold.)

```
[vagrant@admin01 ~]$ ping -c 1 server01  
PING server01 (10.9.8.11) 56(84) bytes of data.  
64 bytes from server01 (10.9.8.11): icmp_seq=1 ttl=64 time=0.212 ms  
  
--- server01 ping statistics ---  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 0.212/0.212/0.212/0.000 ms  
[vagrant@admin01 ~]$ ping -c 1 server02  
PING server02 (10.9.8.12) 56(84) bytes of data.  
64 bytes from server02 (10.9.8.12): icmp_seq=1 ttl=64 time=0.220 ms  
  
--- server02 ping statistics ---  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 0.220/0.220/0.220/0.000 ms
```

Configure SSH Authentication

Create an SSH key pair on admin01 with the `ssh-keygen` command. Accept all the defaults by pressing ENTER. (Even press enter when prompted for a password as you do NOT want to assign a password to the SSH key you are creating.)

Here is an example run `ssh-keygen`. (Portions typed are in bold.)

```
[vagrant@admin01 ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vagrant/.ssh/id_rsa): (press ENTER)
Enter passphrase (empty for no passphrase): (press ENTER)
Enter same passphrase again: (press ENTER)
Your identification has been saved in /home/vagrant/.ssh/id_rsa.
Your public key has been saved in /home/vagrant/.ssh/id_rsa.pub.
The key fingerprint is:
19:84:ea:58:24:f3:7e:18:7c:79:8b:35:83:4b:a1:af vagrant@admin01
The key's randomart image is:
+--[ RSA 2048 ]-----+
|          ..          |
|  o . o.             |
| * o +.             |
|   B = =o            |
|  = * =S+           |
| . + = .            |
|    o               |
|    E               |
+-----+

```

Copy the public key to server01. When prompted to "continue connecting," type yes. When prompted for the password, enter "vagrant".

[This space intentionally left blank. Instructions continue on the following page.]

```
[vagrant@admin01 ~]$ ssh-copy-id server01
The authenticity of host 'server01 (10.9.8.11)' can't be established.
ECDSA key fingerprint is cb:39:b2:73:7c:39:a9:84:92:73:de:f7:aa:2f:33:5b.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to
filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are
prompted now it is to install the new keys
vagrant@server01's password: vagrant
```

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'server01'"
and check to make sure that only the key(s) you wanted were added.

Copy the public key to server02. When prompted for the password, enter "vagrant".

```
[vagrant@admin01 ~]$ ssh-copy-id server02
The authenticity of host 'server02 (10.9.8.12)' can't be established.
ECDSA key fingerprint is cb:39:b2:73:7c:39:a9:84:92:73:de:f7:aa:2f:33:5b.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to
filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are
prompted now it is to install the new keys
vagrant@server02's password: vagrant
```

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'server02'"
and check to make sure that only the key(s) you wanted were added.

Make sure you can log into each server without a password:

```
[vagrant@admin01 ~]$ ssh server01 hostname
server01
[vagrant@admin01 ~]$ ssh server02 hostname
server02
```

Navigate to the /vagrant Directory and Create the /vagrant/servers File

```
cd /vagrant
echo server01 > servers
echo server02 >> servers
```

Confirm the contents of the file:

```
cat servers
server01
server02
```

Write the Shell Script

At this point, you can either create the script inside the virtual machine using the `vim`, `nano`, or `emacs` text editors or you can create the file using your favorite text editor on your local operating system. (Atom from <https://atom.io/> is a good choice.)

When creating your script, refer back to the [shell script requirements](#). If you want or need more detailed steps to help you write your script, refer to the [pseudocode](#) at the end of this document. It was intentionally placed at the end of the document because I want to encourage you to write the script on your own. It's fine if you need the pseudocode. As you get more scripting practice, you'll be able to script without any additional aids.

Test Your Script

Once you've finished writing the script, test it by:

- Executing it with super user privileges.
- Executing it without any options or arguments.
- Executing it with an invalid option.
- Executing the command "hostname" on all the servers listed in `/vagrant/servers`.
- Executing the command "hostname" with the `-n` option.
- Executing the command "uptime" with the `-v` option.
- Executing the command "id" with the `-s` and `-n` options.
- Executing the command "id" with the `-s` and `-v` options.
- Creating a file named `/vagrant/test` that only contains the `server01` host and executing the "hostname" command against that list.
- Passing a nonexistent file to the `-f` option.
- Creating an account name `test1` on all the servers listed in `/vagrant/servers`.
- Creating an account named `test2` with the comment "Test Two" on all servers.
- Displaying the last two lines in the `/etc/passwd` file on all servers.
- Executing a command that doesn't exist.
- Taking a server off the network and then executing the script.

Remember that the first time you execute the script you'll need to make sure it has executable permissions.

```
chmod 755 run-everywhere.sh
```

Here is an example run of the script. (Portions typed are in bold.)

```
sudo ./run-everywhere.sh
```

Do not execute this script as root. Use the **-s** option instead.

Usage: **./run-everywhere.sh** [-nsv] [-f FILE] COMMAND

Executes COMMAND as a single command on every server.

-f FILE Use FILE for the list of servers. Default: /vagrant/servers.

-n Dry run mode. Display the COMMAND that would have been executed and exit.

-s Execute the COMMAND using sudo on the remote server.

-v Verbose mode. Displays the server name before executing COMMAND.

```
echo ${?}
```

```
1
```

Make sure the script displays a usage message if we don't supply a command to execute on the remote hosts.

```
./run-everywhere.sh
```

Usage: **./run-everywhere.sh** [-nsv] [-f FILE] COMMAND

Executes COMMAND as a single command on every server.

-f FILE Use FILE for the list of servers. Default: /vagrant/servers.

-n Dry run mode. Display the COMMAND that would have been executed and exit.

-s Execute the COMMAND using sudo on the remote server.

-v Verbose mode. Displays the server name before executing COMMAND.

```
echo ${?}
```

```
1
```

Make sure the script displays usage message if we supply an invalid option

```
./run-everywhere.sh -x hostname
```

```
./run-everywhere.sh: illegal option -- x
```

Usage: **./run-everywhere.sh** [-nsv] [-f FILE] COMMAND

Executes COMMAND as a single command on every server.

-f FILE Use FILE for the list of servers. Default: /vagrant/servers.

-n Dry run mode. Display the COMMAND that would have been executed and exit.

-s Execute the COMMAND using sudo on the remote server.

-v Verbose mode. Displays the server name before executing COMMAND.

```
echo ${?}
```

```
1
```

Execute the "hostname" command.

```
./run-everywhere.sh hostname  
server01  
server02
```

Execute the script using the dry run (-n) option.

```
./run-everywhere.sh -n hostname  
DRY RUN: ssh -o ConnectTimeout=2 server01 hostname  
DRY RUN: ssh -o ConnectTimeout=2 server02 hostname
```

Execute the uptime command using the verbose (-v) option.

```
./run-everywhere.sh -v uptime  
server01  
13:01:30 up 3:22, 0 users, load average: 0.00, 0.01, 0.03  
server02  
13:04:04 up 4 min, 0 users, load average: 0.01, 0.05, 0.04
```

Execute the script using the dry run (-n) and sudo (-s) options.

```
./run-everywhere.sh -ns id  
DRY RUN: ssh -o ConnectTimeout=2 server01 sudo id  
DRY RUN: ssh -o ConnectTimeout=2 server02 sudo id
```

Execute the id command using the verbose (-v) and sudo (-s) options.

```
./run-everywhere.sh -sv id  
server01  
uid=0(root) gid=0(root) groups=0(root)  
server02  
uid=0(root) gid=0(root) groups=0(root)
```

Create a file named /vagrant/test that only contains the server01 host. Execute the hostname command against that list.

```
echo server01 > test  
./run-everywhere.sh -f test hostname  
server01
```


Make sure that the script exits if provided with a server file that does not exist.

```
./run-everywhere.sh -f /path/to/nowhere hostname  
Cannot open server list file /path/to/nowhere.
```

Add a test1 account on all the servers. Remember that creating accounts requires superuser (root) privileges! Because the "useradd" command does not generate output, check to see if the accounts were created by using the "id" command.

```
./run-everywhere.sh -s useradd test1  
./run-everywhere.sh id test1  
uid=1001(test1) gid=1001(test1) groups=1001(test1)  
uid=1001(test1) gid=1001(test1) groups=1001(test1)
```

Add a test2 account with the comment of "Test Two" on all the servers. Because we want to use a quoted string on the remote system, we have to put the command in quotes. To quote a quote, use the opposing quotation mark.

For example, to preserve single quotes in a string, surround the string with double quotes:

```
echo "'Test Two'"  
'Test Two'
```

To preserve double quotes in a string, surround the string with single quotes:

```
echo '"Test Two"'  
"Test Two"
```

If we want to preserve our quotes for the argument to the -c option, we need to make sure they are quoted:

```
./run-everywhere.sh -ns 'useradd -c "Test Two" test2'  
DRY RUN: ssh -o ConnectTimeout=2 server01 sudo useradd -c "Test Two" test2  
DRY RUN: ssh -o ConnectTimeout=2 server02 sudo useradd -c "Test Two" test2  
./run-everywhere.sh -s 'useradd -c "Test Two" test2'  
./run-everywhere.sh id test2  
uid=1002(test2) gid=1002(test2) groups=1002(test2)  
uid=1002(test2) gid=1002(test2) groups=1002(test2)
```

Check to see the the command made it to the password file for the test2 user.

```
./run-everywhere.sh -v tail -2 /etc/passwd  
server01  
test1:x:1001:1001::/home/test1:/bin/bash  
test2:x:1002:1002:Test Two:/home/test2:/bin/bash  
server02  
test1:x:1001:1001::/home/test1:/bin/bash  
test2:x:1002:1002:Test Two:/home/test2:/bin/bash
```

Execute a command that does exist. Make sure the exit status of the script is non-zero.

```
./run-everywhere.sh i-like-eggs  
bash: i-like-eggs: command not found  
Execution on server01 failed.  
bash: i-like-eggs: command not found  
Execution on server02 failed.  
echo $?  
127
```

Power down one of the servers and execute a command using the script. Make sure the exit status is non-zero.

```
exit  
vagrant halt server02  
==> server02: Attempting graceful shutdown of VM...  
vagrant ssh admin01  
Last login: Mon Jan 29 12:08:50 2018 from 10.0.2.2  
cd /vagrant  
./run-everywhere.sh -v uptime  
server01  
 13:50:24 up  3:11,  0 users,  load average: 0.00, 0.01, 0.03  
server02  
ssh: connect to host server02 port 22: Connection timed out  
Execution on server02 failed.  
echo $?  
255
```

[This space intentionally left blank. Instructions continue on the following page.]

Reference Material:

Vagrantfile for multinet

Here are the contents of the shellclass/multinet/Vagrantfile file with all the comments removed.

```
Vagrant.configure(2) do |config|
  config.vm.box = "jasonc/centos7"

  config.vm.define "admin01" do |admin01|
    admin01.vm.hostname = "admin01"
    admin01.vm.network "private_network", ip: "10.9.8.10"
  end

  config.vm.define "server01" do |server01|
    server01.vm.hostname = "server01"
    server01.vm.network "private_network", ip: "10.9.8.11"
  end

  config.vm.define "server02" do |server02|
    server02.vm.hostname = "server02"
    server02.vm.network "private_network", ip: "10.9.8.12"
  end

end
```

[This space intentionally left blank. Instructions continue on the following page.]

Pseudocode

You can use the following pseudocode to help you with the logic and flow of your script.

```
# Display the usage and exit.

# Make sure the script is not being executed with superuser privileges.

# Parse the options.

# Remove the options while leaving the remaining arguments.

# If the user doesn't supply at least one argument, give them help.

# Anything that remains on the command line is to be treated as a single
command.

# Make sure the SERVER_LIST file exists.

# Loop through the SERVER_LIST

    # If it's a dry run, don't execute anything, just echo it.

    # Capture any non-zero exit status from the SSH_COMMAND and report to
the user.
```