Name: Ian Carlo T. Bello	**Date Performed:** 09/11/22
Course/Section: CPE 232 – CPE31S24	**Date Submitted:** 09/11/22
Instructor: Dr. Jonathan V. Taylar	**Semester and SY:** 1st Sem – 3rd Year

Activity 4: Running Elevated Ad hoc Commands

1. **Objectives:**

1.1 Use commands that makes changes to remote machines

1.2 Use playbook in automating ansible commands

2. **Discussion:**

Provide screenshots for each task.

Elevated Ad hoc commands

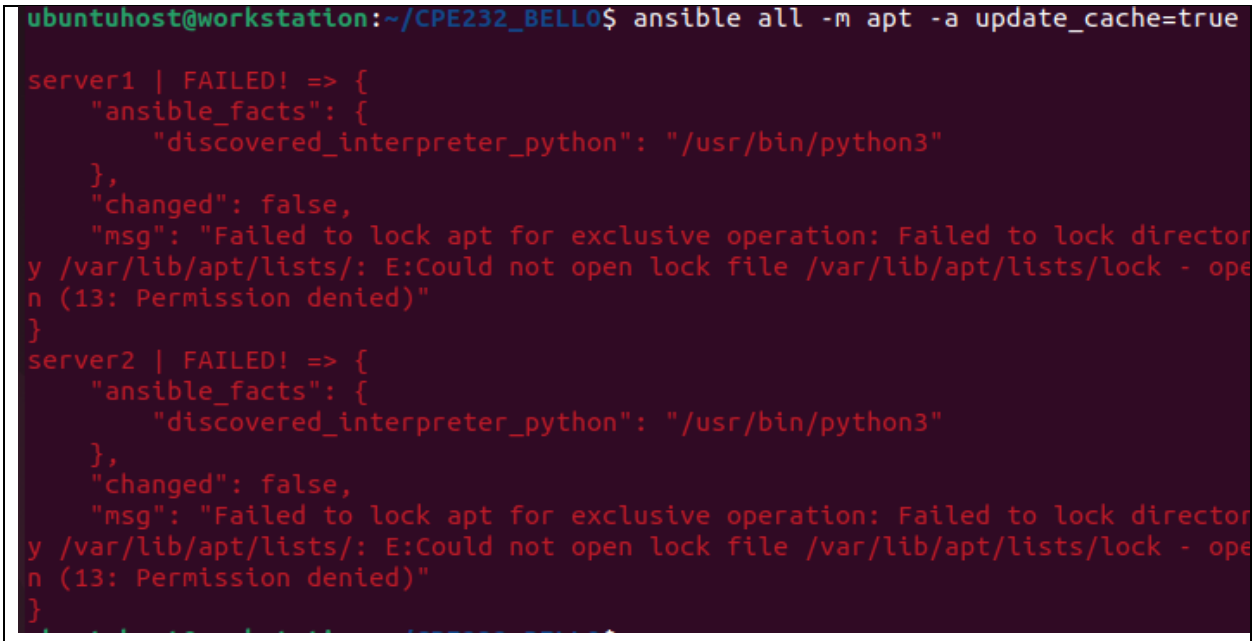
So far, we have not performed ansible commands that makes changes to the remote servers. We manage to gather facts and connect to the remote machines, but we still did not make changes on those machines. In this activity, we will learn to use commands that would install, update, and upgrade packages in the remote machines. We will also create a playbook that will be used for automations.

Playbooks record and execute **Ansible**'s configuration, deployment, and orchestration functions. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process. If Ansible modules are the tools in your workshop, playbooks are your instruction manuals, and your inventory of hosts are your raw material. At a basic level, playbooks can be used to manage configurations of and deployments to remote machines. At a more advanced level, they can sequence multi-tier rollouts involving rolling updates, and can delegate actions to other hosts, interacting with monitoring servers and load balancers along the way. You can check this documentation if you want to learn more about playbooks. Working with playbooks — Ansible Documentation

Task 1: Run elevated ad hoc commands

1. Locally, we use the command *sudo apt update* when we want to download package information from all configured resources. The sources often defined in /etc/apt/sources.list file and other files located in /etc/apt/sources.list.d/ directory. So, when you run update command, it downloads the package information from the Internet. It is useful to get info on an updated version of packages or their dependencies. We can only run an apt update command in a remote machine. Issue the following command:

   ansible all -m apt -a update_cache=true



What is the result of the command? Is it successful?
Permission denied, it wasn't successful because there's no permission.

Try editing the command and add something that would elevate the privilege. Issue the command *ansible all -m apt -a update_cache=true --become --ask-become-pass.* Enter the sudo password when prompted. You will notice now that the output of this command is a success. The *update_cache=true* is the same thing as running *sudo apt update*. The --become command elevate the privileges and the *--ask-become-pass* asks for the password. For now, even if we only have changed the packaged index, we were able to change something on the remote server.

You may notice after the second command was executed, the status is CHANGED compared to the first command, which is FAILED.

```
ubuntuhost@workstation:~/CPE232_BELLO$ ansible all -m apt -a update_cache=true
--become --ask-become-pass
BECOME password:
server2 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1662889795,
    "cache_updated": true,
    "changed": true
}
server1 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1662889794,
    "cache_updated": true,
    "changed": true
}
ubuntuhost@workstation:~/CPE232_BELLO$
```

2. Let's try to install VIM, which is an almost compatible version of the UNIX editor Vi. To do this, we will just changed the module part in 1.1 instruction. Here is the command: *ansible all -m apt -a name=vim-nox --become --ask-become-pass.* The command would take some time after typing the password because the local machine instructed the remote servers to actually install the package.

2.1 Verify that you have installed the package in the remote servers. Issue the command *which vim* and the command *apt search vim-nox* respectively. Was the command successful? Yes

```
ubuntuhost@server1:~$ which vim
/usr/bin/vim
```

```
ubuntuhost@server1:~$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/jammy,now 2:8.2.3995-1ubuntu2 amd64 [installed]
  Vi IMproved - enhanced vi editor - with scripting languages support

vim-tiny/jammy,now 2:8.2.3995-1ubuntu2 amd64 [installed,automatic]
  Vi IMproved - enhanced vi editor - compact version

ubuntuhost@server1:~$
```

```
ubuntuhost@server2:~$ which vim
/usr/bin/vim
```

```
ubuntuhost@server2:~$ apt search vim-nox
Sorting... Done
Full Text Search... Done
vim-nox/jammy,now 2:8.2.3995-1ubuntu2 amd64 [installed]
  Vi IMproved - enhanced vi editor - with scripting languages support

vim-tiny/jammy,now 2:8.2.3995-1ubuntu2 amd64 [installed,automatic]
  Vi IMproved - enhanced vi editor - compact version

ubuntuhost@server2:~$
```

2.2 Check the logs in the servers using the following commands: *cd /var/log*. After this, issue the command *ls,* go to the folder *apt* and open history.log. Describe what you see in the history.log.

It shows that we recently made logs that we installed vim-nox successfully.

Server 1:

```
ubuntuhost@server1:~$ cd /var/log
ubuntuhost@server1:/var/log$ ls
alternatives.log  dist-upgrade  fontconfig.log   openvpn
apt               dmesg         gdm3             private
auth.log          dmesg.0       gpu-manager.log  speech-dispatcher
auth.log.1        dmesg.1.gz    hp               syslog
boot.log          dmesg.2.gz    installer        syslog.1
boot.log.1        dmesg.3.gz    journal          ubuntu-advantage.log
bootstrap.log     dmesg.4.gz    kern.log         ubuntu-advantage-timer.log
btmp              dpkg.log      kern.log.1       unattended-upgrades
cups              faillog       lastlog          wtmp
```

```
ubuntuhost@server1:/var/log/apt$ ls
eipp.log.xz  history.log  term.log
ubuntuhost@server1:/var/log/apt$ cat history.log
```

```
Start-Date: 2022-09-11  17:53:18
Commandline: /usr/bin/apt-get -y -o Dpkg::Options::=--force-confdef -o Dpkg::Op
tions::=--force-confold install vim-nox
Requested-By: ubuntuhost (1000)
Install: fonts-lato:amd64 (2.0-2.1, automatic), liblua5.2-0:amd64 (5.2.4-2, aut
omatic), ruby-net-telnet:amd64 (0.1.1-2, automatic), rubygems-integration:amd64
 (1.18, automatic), libruby3.0:amd64 (3.0.2-7ubuntu2.1, automatic), rake:amd64
(13.0.6-2, automatic), vim-nox:amd64 (2:8.2.3995-1ubuntu2), ruby:amd64 (1:3.0~e
xp1, automatic), vim-runtime:amd64 (2:8.2.3995-1ubuntu2, automatic), ruby3.0:am
d64 (3.0.2-7ubuntu2.1, automatic), libjs-jquery:amd64 (3.6.0+dfsg+~3.5.13-1, au
tomatic), ruby-rubygems:amd64 (3.3.5-2, automatic), javascript-common:amd64 (11
+nmu1, automatic), ruby-xmlrpc:amd64 (0.3.2-1ubuntu0.1, automatic), ruby-webric
k:amd64 (1.7.0-3, automatic)
End-Date: 2022-09-11  17:53:49
ubuntuhost@server1:/var/log/apt$
```

Server 2:

```
ubuntuhost@server2:~$ cd /var/log
ubuntuhost@server2:/var/log$ ls
alternatives.log  dist-upgrade  fontconfig.log   openvpn
apt               dmesg         gdm3             private
auth.log          dmesg.0       gpu-manager.log  speech-dispatcher
auth.log.1        dmesg.1.gz    hp               syslog
boot.log          dmesg.2.gz    installer        syslog.1
boot.log.1        dmesg.3.gz    journal          ubuntu-advantage.log
bootstrap.log     dmesg.4.gz    kern.log         ubuntu-advantage-timer.log
btmp              dpkg.log      kern.log.1       unattended-upgrades
cups              faillog       lastlog          wtmp
ubuntuhost@server2:/var/log$ cd apt
ubuntuhost@server2:/var/log/apt$ ls
eipp.log.xz  history.log  term.log
ubuntuhost@server2:/var/log/apt$ cat history.log
```

```
Start-Date: 2022-09-11  17:53:18
Commandline: /usr/bin/apt-get -y -o Dpkg::Options::=--force-confdef -o Dpkg::Options::=--force-confold install vim-nox
Requested-By: ubuntuhost (1000)
Install: fonts-lato:amd64 (2.0-2.1, automatic), liblua5.2-0:amd64 (5.2.4-2, automatic), ruby-net-telnet:amd64 (0.1.1-2, automatic), rubygems-integration:amd64 (1.18, automatic), libruby3.0:amd64 (3.0.2-7ubuntu2.1, automatic), rake:amd64 (13.0.6-2, automatic), vim-nox:amd64 (2:8.2.3995-1ubuntu2), ruby:amd64 (1:3.0~exp1, automatic), vim-runtime:amd64 (2:8.2.3995-1ubuntu2, automatic), ruby3.0:amd64 (3.0.2-7ubuntu2.1, automatic), libjs-jquery:amd64 (3.6.0+dfsg+~3.5.13-1, automatic), ruby-rubygems:amd64 (3.3.5-2, automatic), javascript-common:amd64 (11+nmu1, automatic), ruby-xmlrpc:amd64 (0.3.2-1ubuntu0.1, automatic), ruby-webrick:amd64 (1.7.0-3, automatic)
End-Date: 2022-09-11  17:53:52
ubuntuhost@server2:/var/log/apt$
```

3. This time, we will install a package called snapd. Snap is pre-installed in Ubuntu system. However, our goal is to create a command that checks for the latest installation package.

   3.1 Issue the command: *ansible all -m apt -a name=snapd --become --ask-become-pass*

```
ubuntuhost@workstation:~/CPE232_BELLO$ ansible all -m apt -a name=snapd --become --ask-become-pass
BECOME password:
server2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1662889795,
    "cache_updated": false,
    "changed": false
}
server1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1662889794,
    "cache_updated": false,
    "changed": false
}
ubuntuhost@workstation:~/CPE232_BELLO$
```

   Can you describe the result of this command? Is it a success? Did it change anything in the remote servers?
   There was no change made because snapd is already installed to both servers. Yes, it is since there were logs in the history of the servers that says install the snapd but it is already installed so there was no change made.

   3.2 Now, try to issue this command: *ansible all -m apt -a "name=snapd state=latest" --become --ask-become-pass*

```
ubuntuhost@workstation:~/CPE232_BELLO$ ansible all -m apt -a "name=snapd state=
latest" --become --ask-become-pass
BECOME password:
server1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1662889794,
    "cache_updated": false,
    "changed": false
}
server2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "cache_update_time": 1662889795,
    "cache_updated": false,
    "changed": false
}
ubuntuhost@workstation:~/CPE232_BELLO$
```

Describe the output of this command. Notice how we added the command *state=latest* and placed them in double quotations.

The output is just the same because the snapd is already updated to the latest since we did update and upgrade in the previous activities. We added state=latest to focus on looking for update from this function which is snapd.

4. At this point, make sure to commit all changes to GitHub.

```
ubuntuhost@workstation:~/CPE232_BELLO$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
ubuntuhost@workstation:~/CPE232_BELLO$
```

Task 2: Writing our First Playbook

1. With ad hoc commands, we can simplify the administration of remote servers. For example, we can install updates, packages, and applications, etc. However, the real strength of ansible comes from its playbooks. When we write a playbook, we can define the state that we want our servers to be in and the place or commands that ansible will carry out to bring to that state. You can use an editor to create a playbook. Before we proceed, make sure that you are in the directory of the repository that we use in the previous activities (*CPE232_yourname*). Issue the command *nano install_apache.yml*. This will create a playbook file called *install_apache.yml*. The .yml is the basic standard extension for playbook files.

   When the editor appears, type the following:

```
 GNU nano 4.8                 install_apache.yml
---
- hosts: all
  become: true
  tasks:

  - name: install apache2 package
    apt:
      name: apache2
```

```
ubuntuhost@workstation:~/CPE232_BELLO$ cat install_apache.yml
---
- hosts: all
  become: true
  tasks:

  - name: install apache2 package
    apt:
      name: apache2
ubuntuhost@workstation:~/CPE232_BELLO$
```

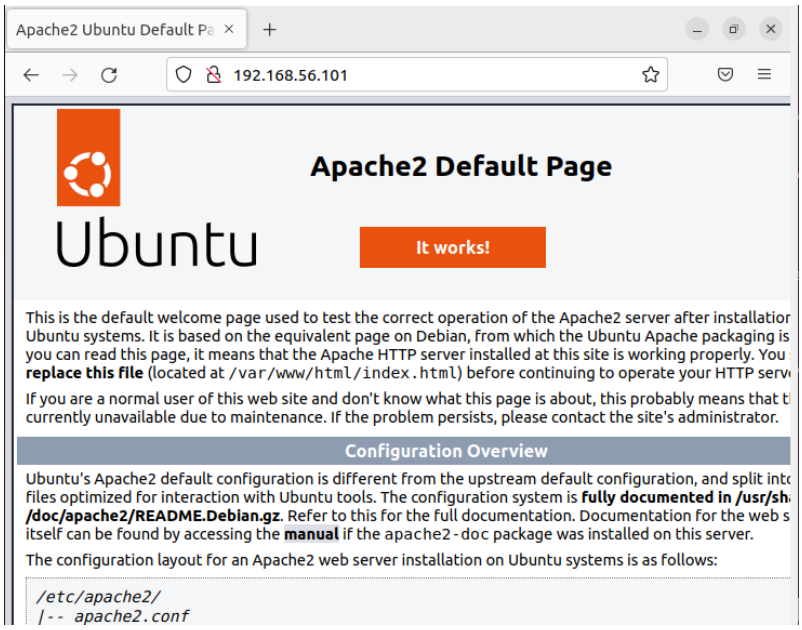
Make sure to save the file. Take note also of the alignments of the texts.

2. Run the yml file using the command: *ansible-playbook --ask-become-pass install_apache.yml.* Describe the result of this command.
   We checked if both servers are reachable and we install apache2 package into both of them.

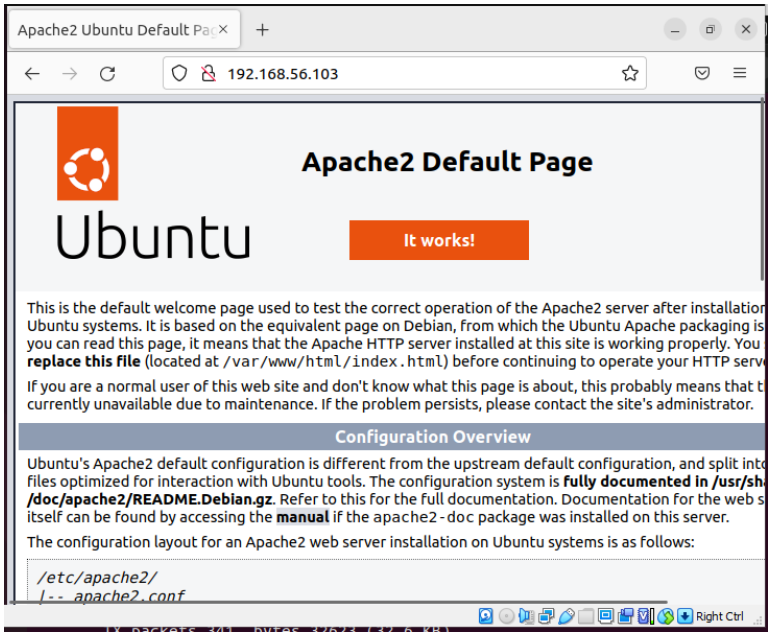
```
ubuntuhost@workstation:~/CPE232_BELLO$ ansible-playbook --ask-become-pass insta
ll_apache.yml
BECOME password:

PLAY [all] *********************************************************************
*

TASK [Gathering Facts] *********************************************************
*
ok: [server1]
ok: [server2]

TASK [install apache2 package] *************************************************
*
changed: [server2]
changed: [server1]

PLAY RECAP *********************************************************************
*
server1                    : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
server2                    : ok=2    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0

ubuntuhost@workstation:~/CPE232_BELLO$
```

3. To verify that apache2 was installed automatically in the remote servers, go to the web browsers on each server and type its IP address. You should see something like this.

Server 1



Server 2



4. Try to edit the *install_apache.yml* and change the name of the package to any name that will not be recognized. What is the output? No changes or install are made.

```
ubuntuhost@workstation:~/CPE232_BELLO$ ansible-playbook --ask-become-pass insta
ll_apache.yml
BECOME password:

PLAY [all] *********************************************************************
*

TASK [Gathering Facts] *********************************************************
*
ok: [server1]
ok: [server2]

TASK [install randomnameiancute package] ***************************************
*
ok: [server1]
ok: [server2]

PLAY RECAP *********************************************************************
*
server1                    : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
server2                    : ok=2    changed=0    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0

ubuntuhost@workstation:~/CPE232_BELLO$
```

5. This time, we are going to put additional task to our playbook. Edit the *install_apache.yml*. As you can see, we are now adding an additional command, which is the *update_cache.* This command updates existing package-indexes on a supporting distro but not upgrading installed-packages (utilities) that were being installed.

```
---
- hosts: all
  become: true
  tasks:

  - name: update repository index
    apt:
      update_cache: yes

  - name: install apache2 package
    apt:
      name: apache2
```

Save the changes to this file and exit.

```
ubuntuhost@workstation:~/CPE232_BELLO$ cat install_apache.yml
---
- hosts: all
  become: true
  tasks:

  - name: update repository index
    apt:
      update_cache: yes

  - name: install apache2 package
    apt:
      name: apache2
ubuntuhost@workstation:~/CPE232_BELLO$
```

6. Run the playbook and describe the output. Did the new command change anything on the remote servers?
   The only changes are from the command update repository index.

```
ubuntuhost@workstation:~/CPE232_BELLO$ ansible-playbook --ask-become-pass insta
ll_apache.yml
BECOME password:

PLAY [all] *********************************************************************
*

TASK [Gathering Facts] *********************************************************
*
ok: [server1]
ok: [server2]

TASK [update repository index] *************************************************
*
changed: [server1]
changed: [server2]

TASK [install apache2 package] *************************************************
*
ok: [server1]
ok: [server2]

PLAY RECAP *********************************************************************
*
server1                    : ok=3    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
server2                    : ok=3    changed=1    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
```

7. Edit again the *install_apache.yml*. This time, we are going to add a PHP support for the apache package we installed earlier.

```
---
- hosts: all
  become: true
  tasks:

  - name: update repository index
    apt:
      update_cache: yes

  - name: install apache2 package
    apt:
      name: apache2

  - name: add PHP support for apache
    apt:
      name: libapache2-mod-php
```

Save the changes to this file and exit.

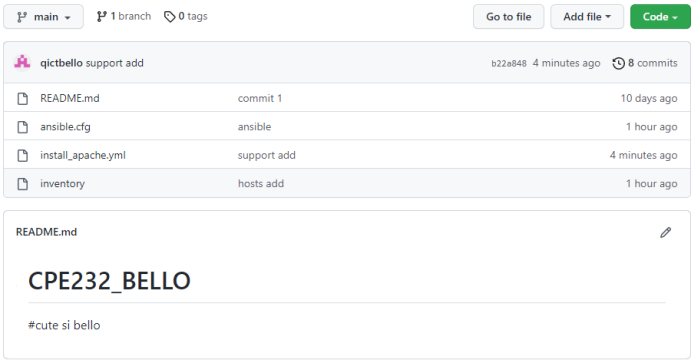
```
ubuntuhost@workstation:~/CPE232_BELLO$ cat install_apache.yml
---
- hosts: all
  become: true
  tasks:

  - name: update repository index
    apt:
      update_cache: yes

  - name: install apache2 package
    apt:
      name: apache2

  - name: add PHP support for apache
    apt:
      name: libapache2-mod-php
ubuntuhost@workstation:~/CPE232_BELLO$
```

8. Run the playbook and describe the output. Did the new command change anything on the remote servers? Yes we 2 in each servers are changes which

from updating the repository and adding PHP support for apache in both servers.

```
TASK [Gathering Facts] *********************************************************
*
ok: [server1]
ok: [server2]

TASK [update repository index] *************************************************
*
changed: [server1]
changed: [server2]

TASK [install apache2 package] *************************************************
*
ok: [server1]
ok: [server2]

TASK [add PHP support for apache] **********************************************
*
changed: [server1]
changed: [server2]

PLAY RECAP *********************************************************************
*
server1                    : ok=4    changed=2    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0
server2                    : ok=4    changed=2    unreachable=0    failed=0
skipped=0    rescued=0    ignored=0

ubuntuhost@workstation:~/CPE232_BELLO$
```

9. Finally, make sure that we are in sync with GitHub. Provide the link of your GitHub repository.



https://github.com/qictbello/CPE232_BELLO

Reflections:

Answer the following:

1. What is the importance of using a playbook?
   In my opinion, the importance of playbook is having a list of things to do or commit in just one click example are we made recently we did update repository then install apache if it wasn't installed and add PHP support for apache in just one command by using this playbook. We can also use this playbook as our daily command to run every day to check every app we wanted to get updated in just one command.
2. Summarize what we have done on this activity.
   We used ansible in our repository to execute multiple commands for each server in one command. We tried to update using ansible, install packages like snapd and vim-nox. We also learned to use playbook to execute multiple commands for all servers in one command.

"I affirm that I shall not give or receive any unauthorized help on this activity and that all work shall be my own."