CMPE 283 Worksheet 03

Project 03

Deadline 0900 Monday, October 13

Classwork

In this class we will continue setting up our droplets and Linux on our own computers.

Students who wish to set up a virtual server on a different provider from Digital Ocean or who want to use different operating system choices are absolutely free to do so. You should note that the instructions in the worksheet may not work exactly in other Linux distributions, though they will almost certainly work with slight modifications.

Any student who does not want to set up a droplet for money reasons, should contact me immediately so we can look for ways to solve the problem.

You should also note that your lecturer has not used Microsoft software at all for the past 14 years, so I will not have the knowledge to help you solve problems with Microsoft software, though you are, of course, free to use it if you have the necessary licences and are prepared to make the effort to support it yourself.

Setting up Linux as an alternative boot on your own computer will make it easier for you to experiment and learn offline and make certificated access to your droplet or other server easier. I strongly recommend you do this.

You should be able to do many of the tasks required from this course from a tablet or even a smart phone. However, the easiest and most productive way to do the practical work in this course will be from a personal computer with Linux installed. Solving the problems you will encounter with *any* method of accessing your server is *part of the course*.

Setting up Linux on your own computer

If you have not yet done this, do it now.

I have a couple of USB sticks with Ubuntu on that you can use in the labs.

1. Backup the data on your computer.
2. Boot Ubuntu from a USB stick. 64 bit verion is preferable on a modern computer, essential if RAM is more than 2 Gbyte.
3. Check that internet access and some common functions are working in the Ubuntu booted from USB stick.
4. If all is ok click "Install" and follow the on screen instructions.

Doing more to set up your droplet

I suggest you have Linux up and running on your computer while you are working on the server it will be handy, and also good practice.

If you do not yet have Linux on your computer, you will need to use a program like Putty to access your droplet.

```
su
```

Getting ready to set up Apache

If you have decided to use a different web server program from Apache, and you are free to do so, you will need to figure out your own ways to achieve the confgurations we achieve here.

Now we are going to set up Apache and try to make it reasonably secure.

Some preliminaries

Apache is a program that takes in requests, finds answers to those requests and sends them back.

Sometimes this is done by finding a file in the server file system. The relationship between the request coming in and the response may be as simple as the part of the request coming after the first / giving the location of the file that should be sent, or something more complicated.

Apache is a fairly complicated program, because is has been extended and extended to "scratch the itches" of all kinds of people. We will only use a tiny part of what it can do. The trick is to learn just enough. There are other server programs that are more lightweight (lighttpd) more powerful (example nginx). We use Apache

simply because the immense size of its user base (around half of all web sites in the world) means we can find ready solutions to problems.

Apache is produced by the Apache Foundation. The Apache Foundation runs many many open source projects. It has a different approach from the strict Free Software approach – because the Apache Foundation is funded directly or indirectly (by allowing employees to work on Apache projects) by companies like IBM. The Apache licence allows these companies to take the source code from Apache projects, modify and close it and turn them into proprietary software. It is an interesting comment on the way modern capitalism works that this is a profitable way of investing for these companies.

In Debian, Apache is referenced as apache2, to distinguish it from the previous version of apache. The program has different names in different operating systems. It may appear as apache or even httpd.

The working of apache is controlled by a series of configuration files. These are text files, kept under the /etc/apache2 directory in Debian. These files contain something close to a programming language that tell the apache program how to handle requests.

The Apache program lets incoming requests that can come from anywhere on the internet access files on your server. And it lets those requests specify names and places for those files. This is potentially very dangerous. So we need to set up apache very carefully to make sure that incoming requests can only affect our system in exactly the ways we want them to, and no other. So the apache config files and our use of directory and file permissions are important.

Some remarks about config files

We already altered one config file, `sshd_config`, to set up SSH access to the server.

Config files have some common general properties

1. Lines beginning with # are comments. Typically a config file will be setup with explanation lines telling you some of what you need to use the file. You still need man or info or other documentation to learn everything. One way that the config file may help you is that things you may want to do are already set up for you, but preceded by a # at the beginning of the line. All you need to do to turn on whatever it is that is currently turned off is to remove the #. We will see examples
2. Big programs may have many config files. Often a master config file will instruct the program to read in all the files in a directory. Sometimes that directory's name will end with ".d", so you know it is a directory. But not

always. If you see a ".d" name, you can be pretty sure that it is a directory. This is a way of having a program work on lots of different things (for example different web sites, or different domain names) and allowing you to alter they way one works without disturbing the others at all.

Some remarks about directories and file permissions and an exercise

Our server may have many web sites, and many users. The Unix file and directory permission system is designed to offer some protection against these different users damaging one another's work. And that different user may always be a user identity that has been hijacked in some way by an outsider.

Here is a small introduction to a big subject. We will work on out own computer because working through these examples would be a bit dangerous on the server. Linux(Unix has named users and named groups. Usually each user is a member of a group with the same name as the user. Users can then also join other groups, which are usually created specially and do not correspond to a single user.

Each file and each directory is owned by a user and owned by a group. They do not have to be the same.

Each file and each directory has a set of permissions which say what the user who owns it (u), a member of the group that owns it (g), and what some other person (o) can do with the file.

The things you can do with a file are read it (r), write it (w) and execute it as a program (x).

so when you do

```
ls -l
```

you will see, for each file, something like this

```
- rwxr-xr--
```

This means that the owning user can read write and execute, the owning group can read and execute, others can only read the file. If you see a d at the beginning, then this file is a directory and some things have different meanings. That is advanced stuff for later.

A little exercise.

From your desktop terminal

```
echo \My File Contents"
```

The echo command simply outputs what you give it on the command line The ¿ character on the command line takes what would have been output and puts it in a file

```
echo \My File Contents" > testfile
```

(You did use the up arrow key to avoid retyping didn't you? If not, try again) By the way, ¿ will overwrite the contents of any already existing file, ¿¿ will add to the cıontents

```
echo \Another line" >> testfile
cat testfile
```

Remember cat takes a file and outputs its contents. You have made a two line file called testfile

```
ls -l
```

You can see the permissions for this file chmod will set the permissions. Use the up arrow key to help you keep inputting the following commands and see what happens. Remember if you use the up arrow twice or more you can go back two or more commands.

```
chmod u-r testfile
ls -l testfile
cat testfile
```

You got an error message? Yes, you took away your own permission to read the file. Get permission back.

```
chmod u+r testfile
ls -l testfile
cat testfile
```

Now you can read the file again. Finally an exercise on directories. Don't forget to use tup arrow and tab completion to make these commands easy to enter

```
mkdir testdirectory
cp testfile testdirectory
```

(You can enter cp testf¡tab¿ testd¡tab¿)

```
cp testfile testdirectory/testfile2
```

(Using up arrow and tab completion you actually only need two keystrokes to do this line can you see how?) Now you have made a subdirectory with two identical files in it.

```
cat testdirectory/*
```

No lets make the whole subdirectory unreadable by anyone

```
chmod -R ugo-r testdirectory
```

-R makes the chmod command work on all the contents of subdirectories down to the bottom of the tree.

```
cat testdirectory/*
```

Again you get an error message.

```
ls -l testdirectory
ls -ld testdirectory
```

The second command looks at the directory for its own permissions, rather than its contents. Now let us get our access back

```
chmod -R ugo+r testdirectory
ls -l testdirectory
ls -ld testdirectory
cat testdirectory/*
```

Finally, we do not want to leave all this rubbish around in our home directory, so we clear up.

```
rm -R testdirectory
rm testfile
ls -l
```

That should be self explanatory. The rm command removes files. rm -R removes directories recursively including all their contents subdirectories, subsubdirectories and so on. So

```
WARNING: DO NOT ENTER THIS COMMAND
rm -R /*
```

if you have root privileges, will remove every single file on your server, including the operating system.

There are two sorts of linux system administrator: those who have already issued that command, and those that are going to. I am in the first category. It wasn't a nice experience. The command I entered looked quite different to that one, but it had exactly the same effect. It took about 1 second for me to realise what I had done. I banged CTRL-C (hold down CTRL and C together) to interrupt the command. By the time the command stopped, irreparable damage had been done to a system with a dozen web sites on it. And the last backup was almost 24 hours old. A lot of information be lost when you have to restore from a day old backup.

Lessons

Only use root privileges (via su or sudo) when you absolutely have to.

When you have root privileges be extra careful with tab completion and the up arrow keys. They let you do things more quickly. Including stupid things that you will regret doing without thinking first.

Test sets of commands or command scripts in dummy directories that do not matter before you do anything to the real system.

Take automatic backups no less than once an hour. Be sure that you can restore from your backups. If you are doing anything you suspect might be dangerous, take another backup just before you do it.

Assignment

1. Finish up what you started in class. Be prepared to show what you have done in the next lab class.

2. Write up an account of what you did and what you learned. What was wrong or missing in these instructions? Could you now explain what you have done so far to a friend?

3. post your account to online.bilgi.edu.tr along with a link to your droplet.