IT 341: Introduction to System Administration

Project 2 (*REMOTE version - Summer 2020 to Present*): Administering and Using a Distributed File System

Now that we have set up networking and Network Information Service (NIS) on your Virtual Machine (VM), we can take advantage of this in order to implement some more advanced features for your VMs file system. We will then build on this in order to enable new practices, such as the use of SSH key pairs for user authentication and remote distribution of system-level data updates.

Project II, Part A:

Implementing NFS

After completing this project every box (both `it20` and your VMs) will serve both

- as a *server* (in that they will serve up their own homes)
- and as a *client* (in that they will auto mount everyone else's homes).

Important: In this handout, we assume a few things:

1. The usernames **abird**, **ajb** and **bja** are defined on `it20` – and so can authenticate to other Linux machines on the network via **`NIS`**.
2. There is a home for **abird** on `it20`, i.e. at `it20:/home/abird`.
3. There is a home for **ajb** on `it20`, i.e. at `it20:/home/ajb`.
4. There is a home for **bja** on `itvm28-8a`, i.e. `itvm28-8a:/home/bja`.

These three homes will be moved. When you make the changes for your client host, you may follow the example for **bja** below, but instead of exporting and (auto-) mounting the home directory for **bja**, you will want to export and (auto-) mount the home directories for yourself and for your partner. You will want to (auto-) mount the home directories of your classmates – as was done for **abird** and **ajb** in the examples.

NOTE: The host `itvm28-8a` and its user, **bja** are only examples. Your client will be named `itvm2x-yz` (where you should *know* x, y and z) with two usernames defined, one for you and one for your partner. If you recall, these usernames were created for you – on **`it20`** – as part of the previous project...

On the Server[1], it20

(*This much, I or previous admins have **already** done. You should just read it, in order to understand the server-side configuration done on* `it20`.):

1. Download and install the required NFS packages:

```
sudo apt-get update
sudo apt-get install nfs-common
sudo apt-get install nfs-kernel-server
```

2. Download and install `autofs`

```
sudo apt-get install autofs
```

3. Move the homes to another dir, from the root **/** directory, by changing the **home** dir's name

```
sudo mv /home /home.it20
```

4. Create a new *empty* directory to use as a mount point. Call it **/home**

```
sudo mkdir /home
```

5. Edit `/etc/passwd` to say where user **sysadmin**'s home is:

```
...
landscape:x:104:109::/var/lib/landscape:/bin/false
sysadmin:x:1000:1000:sysadmin,,,:/home.it20/sysadmin:/bin/bash
dhcpd:x:105:113::/var/run:/bin/false
abird:x:1001:1001:Al Bird,,,:/home.it20/abird:/bin/bash
...
```

We want user `sysadmin`, on `it20` and on all the clients, to be *local* and so **not** part of NFS. Having `sysadmin` allows us access to any host, whether or not NFS is running properly. So we are *explicit* as to where its home is (its **new** path location).

5. Edit `/etc/exports` so as to export these homes:

```
# /etc/exports: the access control list for filesystems,
# which may be exported
# to NFS clients. See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes hostname1(rw,sync) hostname2(ro,sync)
#
# Example for NFSv4:
# /srv/nfs4 gss/krb5i(rw,sync,fsid=0,crossmnt)
# /srv/nfs4/homes gss/krb5i(rw,sync)
#
/home.it20 10.0.0.0/24(rw,sync,no_root_squash,no_subtree_check)
```

[1]https://help.ubuntu.com/community/SettingUpNFSHowTo
https://help.ubuntu.com/16.04/serverguide/network-file-system.html

6. Edit `/etc/auto.master` to tell `autofs` about the map for `/home`. The directory `/home` must exist and must be empty. It must not have any files in it. Remember we moved the `/home` directory earlier in the project.

```
# $Id: auto.master,v 1.4 2005/01/04 14:36:54 raven Exp $
#
# Sample auto.master file
# This is an automounter map and it has the following format
# key [ -mount-options-separated-by-comma ] location
...
#/net /etc/auto.net
/home /etc/auto.home
```

7. And, define the map: `/etc/auto.home`

```
# Ampersand in the RHS matches the key itself.
abird        it20:/home.it20/&
ajb          it20:/home.it20/&
it341        it20:/home.it20/&
bja     itvm28-8a:/home.itvm28-8a/&
...
...
```

That is it for now. We will return to this file *later* to make sure homes on other boxes get mounted.

The Clients (e.g. itvm28-8a here)

The task on the clients is similar. In fact, when it comes to NFS we are all both servers and clients. We are servers in that we serve up (or export) our local home directories to the network. We are clients in that we (auto) mount the local directories that are served up (exported) by other hosts on the network.

Of course, in these examples, both **itvm28-8a** and user **bja** are just examples. You will want to use your **itvm2x-yz** (*replacing* x, y and z with the actual values) and your *usernames*.

1. Download and install the required NFS and auto-mounting packages.

```
sudo apt-get update
sudo apt-get install nfs-common
sudo apt-get install nfs-kernel-server
sudo apt-get install autofs
```

Installing the application **nmap** is also recommended:

```
sudo apt-get install nmap
```

2. Move the homes to another location, from the `root` **/** directory, by *changing the name* of the original **/home** directory, incorporating your VM's name

sudo mv /home /home.itvm2x-yz   **Example:** `sudo mv /home /home.itvm28-8a`

3. Create a new, *empty* directory to use as a mount point. Call it **/home**

   sudo mkdir /home

4. On every host (both the server and clients) we want user sysadmin to be a truly local user and not part of NFS. Edit /etc/passwd to say where user **sysadmin**'s home is:

   sudo nano /etc/passwd

```
...
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
landscape:x:102:108::/var/lib/landscape:/bin/false
sysadmin:x:1000:1000:sysadmin,,,:/home.itvm28-8a/sysadmin:/bin/bash
sshd:x:103:65534::/var/run/sshd:/usr/sbin/nologin
statd:x:104:65534::/var/lib/nfs:/bin/false
...
```

5. Depending on the editor you are using you may get an error message. Ignore it. At this point sysadmin does not have a home directory. **Why? Think about it and be prepared to address it in the discussion questions.** You need to logout and log in again. Then sysadmin will again have a home directory.

6. Edit /etc/exports so as to export our homes: **sudo nano /etc/exports**

```
# /etc/exports: the access control list for filesystems,
# which may be exported to NFS clients. See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes hostname1(rw,sync) hostname2(ro,sync)
#
# Example for NFSv4:
# /srv/nfs4 gss/krb5i(rw,sync,fsid=0,crossmnt)
# /srv/nfs4/homes gss/krb5i(rw,sync)
#
/home.itvm28-8a 10.0.0.0/24(rw,sync,no_root_squash,no_subtree_check)
```

6. Edit /etc/auto.master to tell autofs about the map for /home. The directory /home must exist and must be empty. It must not have any files in it. Remember we moved the /home directory earlier in the project. (*Do not worry about your file's current contents. Just focus on adding the specified line at the end.*)

   sudo nano /etc/auto.master

```
#
# Sample auto.master file
# This is an automounter map and it has the following format
# key [ -mount-options-separated-by-comma ] location
# For details of the format look at autofs(5).
#
#/misc      /etc/auto.misc
#
# NOTE: mounts done from a hosts map will be mounted with the
#     "nosuid" and "nodev" options unless the "suid" and "dev"
#     options are explicitly given.
#
#/net -hosts
#
```

```
# Include central master map if it can be found using
# nsswitch sources.
#
# Note that if there are entries for /net or /misc (as
# above) in the included master map any keys that are the
# same will not be seen as the first read key seen takes
# precedence.
#
/home /etc/auto.home
```

7. **Then, back to the server `it20`**, an administrative user – in other words, a **sudo-er** – will need to edit `/etc/auto.home` to mount **bja**'s home:

   ```
   # Ampersand in the RHS matches the key itself.
   abird        it20:/home.it20/&
   it341        it20:/home.it20/&
   ajb          it20:/home.it20/&
   bja      itvm28-8a:/home.itvm28-8a/&
   ...
   ```

   By the time your reach this step, the file `/etc/auto.home` on `it20` will have been edited so that it contains a line with your username and where your home can be found on the network. You will need to ***copy*** the file `/etc/auto.home` from `it20` to the same absolute filepath on ***your*** virtual machine. This is best accomplished using **`scp`**

   ```
   sudo scp it341@it20:/etc/auto.home /etc/auto.home
   ```

8. Eventually, all clients will have exported all of their home directories, as well as copied over the `/etc/auto.home` file from `it20`. We will all (server and clients) want to ***automount*** all homes from all servers; indeed, `/etc/auto.home` will look the same on all hosts.

9. At this point, run two commands:

   ```
   sudo service nfs-kernel-server restart
   sudo service autofs restart
   ```

10. Now, we should be able to log on to any hosts (because NIS makes us known to our network), and we should see our home directories on each of these hosts (because NFS makes them available). *This, of course, is true **only** to the extent that all hosts have successfully implemented networking, NIS, and now NFS.* Indeed, NFS makes it look like there is just one common `/home` visible to all hosts. In reality, our own home directories reside on our own hosts (as `/home.itvm2x-yz`) but we export them to the network and each host automounts these under **`/home`**, as needed.

11. You may notice that if, on your VM, you **`cd`** to `/home`, you may not see all home directories; it may look as follows:

   ```
   abird@it20:~$ cd /home
   abird@it20:/home$ ls
   abird
   ```

But, when you explicitly ask for another user's home...

```
abird@it20:/home$ ls bja
abird@it20:/home$ ls
bja abird
```

This is because the automounter only mounts the directories on demand ... once you actually refer to (or *ask for*) the directory explicitly, as in this example:

```
abird@it20:/home$ ls bja
```

Project II, Part B:

Using **ssh** , **scp** , and **sftp** with Key-Based Authentication

scp and sftp

When you install **ssh**, you also get **scp**, a secure copy for doing secure **cp** procedures from one machine to another (actually, it is a secure version of **rcp** – remote copy), and **sftp**, a secure version of **ftp**, that is a secure file transfer protocol. You can learn about both of these by looking at their **man**
pages: **man scp** **or** **man sftp** (Of course, there is also a man page for **ssh**.)

scp is useful for quickly copying a file from one host to another. For example, say we are on **it20** and we wish to copy **it20's** **/etc/hosts** to **itvm28-8a** ...

(The following is just a demonstration of how the ***scp*** *command can be used. It is not for you to necessarily carry out yourself...)*

- Rather than copy **hosts** directly to directory **/etc** on **itvm28-8a** ...
- ...it is safer to copy it to **itvm28-8a**'s **/tmp** – a directory for holding files temporarily
- Then, once we log on to **itvm28-8a**, we can move it into place (perhaps after saving a backup copy of **itvm28-8a**'s original **/etc/hosts**).
- Anyway, we can use **scp** to do the copy:

```
abird@it20:~$ scp /etc/hosts itvm28-8a:/tmp
abird@itvm28-8a's password:
hosts 100% 628 0.6KB/s 00:00
abird@it20:~$
```

Let's look closer at this command: **scp** **/etc/hosts** **itvm28-8a:/tmp**

1. The first argument to **scp** is the *source* we want to copy. Because it is a local source – on the host we are currently logged into – we need not specify the host.
2. The second argument tells **scp** the *destination* where it should copy the file:
   a. the *host*: **itvm28-8a:**
   b. the *target directory* on that host: **/tmp**
3. Notice **scp** needs **abird**'s password on **itvm28-8a**. (Of course, because of **NIS**, **abird**'s password is the same on all hosts on the network – a good thing.)

We can also copy files from elsewhere to our own host. For example, to copy **itvm28-8a**'s **/etc/hosts** file **to** **it20**'s **/tmp**, we *could* say:

```
abird@it20:~$ scp itvm28-8a:/etc/hosts /tmp
abird@itvm28-8a's password:
hosts 100% 624 0.6KB/s 00:00
```

```
abird@it20:~$
```

(This time, because *the **source** is remote*, we do specify a host. Because *the **destination** is local*, no host.)

Again, we are asked for **`abird`**'s password on **`itvm28-8a`**

We can recursively copy *whole directories* from one host to another. For example, to copy **`itvm28-8a`**'s entire **`/etc`** to our (**it20**'s) **`/tmp`**, we would say

```
abird@it20:~$ scp -r itvm28-8a:/etc /tmp
abird@itvm28-8a's password:
defaultdomain 100% 6 0.0KB/s 00:00
adjtime 100% 48 0.1KB/s 00:00
global 100% 459 0.5KB/s 00:00
config 100% 1568 1.5KB/s 00:00
mtab 100% 629 0.6KB/s 00:00
scp: /etc/shadow: Permission denied
…
a whole lot of files
…
README 100% 371 0.4KB/s 00:00
K16dhcdbd 100% 1506 1.5KB/s 00:00
abird@it20:~$
```

Notice that **`scp`** will not copy **`/etc/shadow`** across; if it did allow it, anyone could take a look at a host's **`/etc/shadow`**, whether they were ***sudo***ers or not. If you want to have full access, you should work as user **root**. (Or, you should ask yourself if you really ***want*** to have such full access – you can really do damage to your system!)

Key-Based Authentication

One thing you may have noticed is that it would be a lot easier if we could push stuff (common files, etc.) from **`it20`** out to the client **`itvm28-8a`**. *And we would like to do so without having to supply a password every time.*

So, we will set up key-based authentication. Following the text, we will use a non-empty passphrase. Of course, this puts us back in the position of having to supply a pass phrase in place of a password. But we can then use **`ssh-agent`** for managing the passphrase exchange whenever we are challenged. As you have read in the text, **`ssh-agent`** caches the passphrase in memory while the current shell is active; when the shell dies, the pass phrase goes with it.

OK, so now our **`ssh`** client is **`it20`** and our **`ssh`** servers (from whom we want to push out files) are the VMs. In our example, we will set up key-based authentication with **`itvm28-8a`**; we use it here only as an example.

On the virtual server:

1. You should first read the section on key-based authentication in the textbook.
2. Log in to your VM as yourself. For the user Al Bird, and in the examples below, it's `abird`. When you do this, use your own username that we created for you on `it20` in the previous project.
3. The first thing we have to do is generate a public/private key pair with the `ssh-keygen` utility. We will use the passphrase `qazxsw` (which is easier to type than you might think).

```
abird@itvm28-8a:~$ ssh-keygen -t rsa -b 2048
Generating public/private rsa key pair.
Enter file in which to save the key (/home/abird/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/abird/.ssh/id_rsa.
key has been saved in /home/abird/.ssh/id_rsa.pub.
The key fingerprint is:
2e:98:6f:72:9f:70:9c:37:11:c1:fc:ed:91:9b:b8:09 abird@itvm28-8a
The key's randomart image is:
+--[ RSA 2048]----+
| o.              |
| o.              |
| ..  .  .        |
| .. +            |
| S . o +         |
| o o .E.. +      |
| o o = o. o      |
| ..o+ o .o       |
| +..o            |
+-----------------+
abird@itvm28-8a:~$
```

Type `qazxsw`

Leave *blank* and press **Enter**

4. Append the content of `/home/abird/.ssh/id_rsa.pub` **to** `/home/abird/.ssh/authorized_keys` ...thus insuring that any file there already is not overridden; if the `authorized_keys` file doesn't already exist, it will be created.

```
abird@itvm28-8a:~$ cd .ssh
abird@itvm28-8a:~$ cat id_rsa.pub >> authorized_keys abird@itvm28-8a:~$
ls -l
total 20
-rw-r--r-- 1 abird abird 598 2011-03-22 14:01 authorized_keys
-rw------- 1 abird abird 736 2011-03-22 13:56 id_rsa
-rw-r--r-- 1 abird abird 598 2011-03-22 13:56 id_rsa.pub
-rw-r--r-- 1 abird abird 7096 2011-03-21 10:12 known_hosts
abird@itvm28-8a:~$
```

Recall, the **>>** denotes an append.

5. Now **ssh** to another machine to see if it works. (***NOTE:*** *This is assuming that both **your VM** and **the target machine** have successfully implemented **NFS**!*)

```
abird@itvm28-8a:~$ ssh itvm26-7c
Enter passphrase for key '/home/abird/.ssh/id_rsa':
Linux it20 2.6.32-29-generic-pae #58-Ubuntu SMP Fri Feb 11
19:15:25 UTC 2011 i686 GNU/Linux
Ubuntu 10.04 LTS

Welcome to Ubuntu!
* Documentation: https://help.ubuntu.com/

System information as of Tue Mar 22 14:04:04 EDT 2011

System load: 0.0 Memory usage: 13% Processes: 85

Usage of /: 7.3% of 18.82GB Swap usage: 0% Users logged in: 0

Graph this data and manage this system at
https://landscape.canonical.com/

Last login: Tue Mar 22 14:01:02 2011 from itvm28-
8a.it.cs.umb.edu
abird@itvm26-7c:~$
```

Instead of asking for **abird**'s password, it asks for the *pass**phrase*** for **abird**'s authentication key. Make sure you understand **why** this works. Be prepared to write about it in your discussion questions….

6. We can make even more progress. We would like to be able to **ssh** into other machines *without* having to supply the passphrase each time. The **ssh-agent** utility allows us to do this within the scope of a single, ongoing login session.

7. Again, let us log out and return to **itvm28-8a**. Here we invoke **ssh-agent** with the name of the shell we want to use as its argument:

```
abird@it20:~$ exit
Connection to it20 closed.
abird@itvm28-8a:~$ ssh-agent /bin/bash
abird@itvm28-8a:~$
```

8. We now invoke **ssh-add**

```
abird@itvm28-8a:~$ ssh-add
Enter passphrase for /home/abird/.ssh/id_rsa:
Identity added: /home/abird/.ssh/id_rsa (/home/abird/.ssh/id_rsa)
abird@itvm28-8a:~$
```

ssh-add adds RSA (or DSA) identities to the authentication agent, **ssh-agent**. When run without arguments, it adds the files

~/.ssh/id_rsa, **~/.ssh/id_dsa** and **~/.ssh/identity**. Alternative file names can be given on the command line. If any file requires a passphrase, **ssh-add** asks for the passphrase from the user.

9. Now, let's try to log into **it20** again.

```
abird@itvm28-8a:~$ ssh it20
Linux it20 2.6.32-29-generic-pae #58-Ubuntu SMP Fri Feb 11
19:15:25 UTC 2011 i686 GNU/Linux
Ubuntu 10.04 LTS

Welcome to Ubuntu!
* Documentation: https://help.ubuntu.com/

System information as of Tue Mar 22 14:09:50 EDT 2011

System load: 0.0 Memory usage: 13% Processes: 84
Usage of /: 7.3% of 18.82GB Swap usage: 0% Users logged in: 0

Graph this data and manage this system at
https://landscape.canonical.com/

Last login: Tue Mar 22 14:04:04 2011 from itvm28-
8a.it.cs.umb.edu
abird@it20:~$
```

Great! The point of this is that, once your authentication key is available on all hosts, thanks to NFS mounting your home directory, you can use **ssh-agent** and **ssh-add** to set up a shell from which you can perform an **ssh**-based task (**ssh**, **scp**, **sftp**, **rdist**, etc.) *without* being challenged for a password or passphrase.

Project II, Part C:

Using **rdist** to Distribute Files

Now that you can get to other machines without supplying a password or pass phrase each time, we can set about automatically distributing files to the clients. This is something we would want to do in an industrial-strength network. There are two tools that are useful to this task: **rdist** and **sed**. **rdist** is used to distribute files; **sed** is used to modify them slightly to accommodate the specifics of various hosts on the network. We'll look at **rdist** here and **sed** in Assignments 3 and 4.

rdist stands for "remote distribution". It is used to distribute files from one host to others. The idea is, that when one wants to maintain files that are to be identical on many hosts, one **maintains** them on one host – making modifications to files *only on that host* – and then uses **rdist** to **distribute** them to the other hosts. If the files on the various machines get out of sync, you just run **rdist**.

One might ask, what's the difference between **rdist** and **rsync**? The answer is: it's a matter of purpose.

- **rdist** is for *distributing* files on the network
- **rsync** is for *backing up (and restoring)* file systems. We will explore **rsync** in the next project...

Each has behaviors particular to its purpose. What might be a scenario where you would want to use **rdist**? Where you would want to instead use **rsync**? Be prepared to address this in your discussion questions.

Installing rdist:
The first thing we have to do is install **rdist**, on our host; I've installed it on it20 but you can install yours on your virtual machine server/client[2]. Assuming you are logged in as **sysadmin**...do this:

```
sudo apt-get update [3]
sudo apt-get install rdist
```

Now, log out of sysadmin, and log in *as yourself*. We are taking advantage of the fact that we set up key authentication in the last part of the project. (NOTE: *You cannot complete this unless you have successfully implemented* ***key-based authentication*** *from earlier!)*

Configuring rdist:

(You will do this part by creating myrdist**, making it** executable**, and creating** Distfile**)**

(If you are not familiar with scripting in a Linux-based environment, consider brushing up on this topic. Google phrases like **linux shell script tutorial**)

By default, **rdist** uses the (non-secure) rsh for transport. We want to use the more secure **ssh**. So, we create our own script **myrdist** – which invokes **rdist** with the proper parameters:

[2] In practice, one chooses a single host, usually a server from which to distribute files but we are just playing here – so play away.

[3] When you do this, it is a good idea to install any updates available.

We start by creating the **myrdist** file (which *each* partner should do):

```
johndoe@itvm28-8a:~$ nano myrdist
```

- Add the following to the file:
-

```
#!/bin/sh
# A preconfigured rdist that uses ssh

SSH="`which ssh`"
RDISTD="`which rdistd`"

rdist -p "$RDISTD" -P "$SSH" "$@"
```

Save the file, and make sure that this file is *executable*:

```
chmod 755 myrdist
```

Then, we must define a *distfile* (the default name is **Distfile**), which describes the sorts of distributions we might want to do. A *distfile* is similar to make's *makefile*; it provides for a list of target tasks, and specifies how each task is to be carried out.

(*The following is just an example, for illustrative purposes only...*)

Consider where we want to copy /etc/hosts from the local client to **itvm28-1b**

- The **hosts:** label specifies the target operation to conduct
- and what follows describes ***what*** must be done:

```
# Distfile for distributing files

hosts: /etc/hosts -> ( itvm28-1b )
install /etc/hosts ;
```

This specifies that to satisfy the target hosts, we copy file /etc/hosts to **itvm28-1b**, and install it as /etc/hosts there. (**Again, you should not actually do this!!!**)

Be extremely careful; a wrong Distfile can cause havoc! See below.

Running rdist

To run **rdist**, we simply type **rdist**, or to use our configured version, **myrdist**. But before doing so, we can modify our **Distfile** by adding a verify option to the install:

```
# Distfile for distributing files

hosts: /etc/hosts -> ( itvm28-1b )
install -overify /etc/hosts ;
```

This says what **rdist** ***would*** do in the current environment, but it doesn't *actually* do it. **rdist** generally only overwrites files whose modify dates are *older* than the files being copied, although one may change this behavior using *options*. See the **rdist** man page.

What are some of the most important options available for **rdist**? Why is that? Be prepared to write about this in your discussion questions...

Play in a sandbox

Before using **rdist** to distribute files to an important directory such as `/etc`, set up a sandbox and practice there. For example, use `/tmp` as the destination directory.

NOTE: The following section is not intended to be carried out by you. It is just here to serve as an ***example***.

```
# Distfile for distributing files

hosts: /etc/hosts -> ( itvm28-1b )
install /tmp/hosts ;
```

To execute **rdist**, we simply type

```
rdist
```

Be careful because **rdist** can overwrite a directory with a file. Saying

```
hosts: /etc/hosts -> ( itvm28-1b )
install /etc/hosts ;
```

is fine as you specify a file name as the destination; **rdist** will write a new `hosts` file in `/etc`. However…

```
hosts: /etc/hosts -> ( itvm28-1b )
install /etc ;
```

is disastrous. It would overwrite the `/etc` directory with the file **hosts**, removing `/etc` contents altogether. (It has happened to previous instructors!)

On the other hand, if you are distributing several files, as in…

```
hosts: ( /etc/hosts /etc/nsswitch.conf ) -> ( itvm28-1b )
install /etc ;
```

...then the behavior is as you would expect: the two files `hosts` and `nsswitch.conf` are copied into directory `/etc` on **itvm28-1b**. What precautions can you take in order to keep yourself from making such? These will be addressed in the discussion questions...

Also, you can ask that one directory overwrite another directory. The following would overwrite `/etc` on **itvm28-1b** with the `/etc` on the machine running **rdist**

```
hosts: /etc -> ( itvm28-1b )
install /etc ;
```

An Exercise (Do this)

As an exercise, on your client, we will start by copying `/etc` to `/tmp`

```
johndoe@itvm28-8a:~$ cp -r /etc /tmp/etc_copy
```

(NOTE: Because you are using your personal account – instead of **`sysadmin`** – you will get **`Permission denied`** for some specific file paths, on account of file permissions. This is okay, as long as you were able to copy *some* of **`/etc`** into **`/tmp/etc_copy`**)

Confirm that *some* of **`/etc`** was copied over:

```
johndoe@itvm28-8a:~$ ls -l /tmp/etc_copy
```

Each ~~partner~~ should do this next step:

Write a `Distfile` that copies some of those files that we have been defining, and that we want on all clients, over to another client

```
johndoe@itvm28-8a:~$ nano Distfile
```

Your **`Distfile`** will need three components:

- **The local source:** Your VMs **`/tmp/etc_copy`** directory
- **The target machine:** Some VM *other than yours* that has also installed **`rdist`**
- **The target install location:** A **`/tmp/`***`your username`***`/etc_copy`** directory on that machine. For user **`johndoe`**, that would be **`/tmp/`***`johndoe`***`/etc_copy`**

Save **`Distfile`** and then run **`myrdist`** (this assumes the user has also created the **`myrdist`** file per earlier instructions...):

```
johndoe@itvm28-8a:~$ ./myrdist
```

To confirm this worked, you can then **`ssh`** into the target machine and see if the directory was created with the appropriate contents...

Experiment with various **`Distfile`**s. And, be careful! **Never use a target that doesn't involve `/tmp`**

Discussion Questions:

1. After you edit `/etc/passwd` on your VM, you might get an error message to the effect that `sysadmin` does not have a home directory.
    a. Whether or not you experience this, why *could* something like that happen?
    b. After you log `sysadmin` out -- and then back in -- `sysadmin` should have a home directory once more. Why would this be the case?
2. How is mounting other users' homes in this project dependent upon the correct configuration of *networking* and *NIS* in Project 2?
3. When you make changes in your home directory (such as adding, deleting, or editing files) while logged into `it20` or someone else's VM, why do those changes persist in your home directory, as it exists on your own VM?
4. What differentiates `ssh`, `scp`, and `sftp` from their non-secure counterparts (`rsh`, `rcp`, and `ftp`)? In other words, what ***makes*** them "secure"?
5. In Part II, Step #5, we saw that the example user `abird` was prompted for a *pass**phrase*** instead of a password. Why does this occur?
6. Using key pairs...
    a. When a user (on his/her local machine) authenticates to a remote host using a key pair, what are the respective roles of the *public* and *private* keys?
    b. Which *keys* (public vs. private) must be on which *hosts* (local vs. remote) for authentication to take place, and why is this the case?
7. How does *NFS-based mounting* enable us to create one pair of key files in our home directories – in our `.ssh` folders – and then be able to log into (and from) any Linux machine within the IT Lab LAN?
8. Identify some of the most important options to the `rdist` command and explain their significance.
9. If you were going to use `rdist` to push an update to other machines' `/etc` directories, what are some *precautions* you can take to ensure that you do not accidentally damage/destroy any essential files on those machines?
10. How can `ssh-agent` make it easier to execute the `rdist` command?