DRAFT

IT341 Introduction to System Administration

Project 4 - Backup Strategies with `rsync` and `crontab`

Backup is one of the most important things a system administrator does. It is important to decide what data on your network is important, and to back that data up on a regular basis. Preferably, the backup process is automated. It's not a matter of *if* you will have to go to a backed-up archive for restoring a corrupted directory or file system, but *when*. As the bumper sticker says: *Things (sic) happen*.

A nice list of ten open-source backup programs may be found at http://www.techrepublic.com/blog/10-things/10-outstanding-linux-backup-utilities/. One of these is **`rsync`**. What's nice about **`rsync`** is that it is a command-line program and so can be invoked from with other scripts, providing for automation. **`rsync`** can be invoked directly or it can be set up as a daemon so as to keep a backed-up archive in sync with the working copy. **`rsync`** is one of the most widely used Linux backup tools.

Most of the work we have been doing involves **`/etc`** and perhaps **`/home.itvm2x-yz`** on each host; **`/etc`** seems to be the most important. Once we have decided what data is important to us, we must decide *where* to back it up to. There are several possibilities:

1. Somewhere else on the same host (like another disk), under the assumption that it's unlikely that two disks will fail at once.
2. On a disk on another host, under the assumption that it is even less likely that two disks on two hosts will fail at the same time.
3. On a DVD or a tape; these can be stored elsewhere, off site (protecting us from fire). The downside is that we require additional human intervention.
4. (More recently) In the cloud, where they will likely be further backed up ***and*** accessible from different locations. This, however, requires that you be able to trust your cloud provider and their security measures.

For our little network, we will go for the ***second*** option. It is unlikely that two disks on two hosts will fail at the same time. If there is a major fire, data backup is the least of our problems in this short semester. Also, the backup process can be automated (say, using a **`cron`** process for scheduling backups).

Part I - Invoking `rsync` Directly

1. For the server and each client, we must choose a location to use for the backup files. We will use the **`/it341_backups`** directory tree on **`it20`** for a location. *Each team* will have a subdirectory named after their virtual machine and will store their backups inside of it. For example, the team ***itvm23-7b*** would have the following directory:

   `/it341_backups/itvm23-7b`

2. In addition, there will also be a group on **it20**, named after your team. You and your partner will belong to that group. Your backup directory will be owned by root, but its group will be yours. For example, the backup directory for itvm23-7b would look as follows:

   ```
   drwxrwx---  2 root itvm23-7b 4096 Apr 25 16:29 itvm23-7b/
   ```

   By virtue of group membership, you and your partner will have full permissions on your backup directory.

3. Each backup will have a subdirectory named for the date of the backup. For example, on April 17, 2012 we will name the subdirectory 04172012. The fully qualified directory would be ***/it341 backups/itvm23-7b/04172012***

4. For example, say we are backing **itvm28-2b** to **it20**; again, this is just an example. We must first create a location on **it20**. We log onto **it20** as ourselves and change our working directory to **/it341 backups/itvm28-2b** (These directories have already been created. See me if you have difficulties with access and/or writing.) Execute a **pwd** command, to ensure you are in the correct location, and make sure you are able to create files/subdirectories inside there.

5. So we exit, returning to **itvm28-2b**, and execute a **rsync** command. Pay particular attention to who you are logged *into your VM* as versus who you are logging *into it20* as! (Note that some commands may be too long for this page and wrap to the next line):

NOTE: This is one command. You ***do not*** press Enter after **/etc**

```
sysadmin@itvm28-2b:~$ sudo rsync -azvv -e ssh /etc
  abird@it20.it.cs.umb.edu:/it341_backups/itvm28-2b/04172012

opening connection using: ssh -l abird it20.it.cs.umb.edu
rsync --server -vvlogDtprze.iLsf . /it341_backups/itvm28-
2b/04172012
abird@it20.it.cs.umb.edu's password:
sending incremental file list
created directory itvm28-2b/04172012
delta-transmission enabled
etc/
etc/.pwd.lock
etc/adduser.conf
etc/at.deny
etc/auto.home
etc/auto.home.bak
...
<lots more files>
...
etc/xml/
etc/xml/catalog
etc/xml/catalog.old
etc/xml/xml-core.xml
etc/xml/xml-core.xml.old
```

```
total: matches=0 hash_hits=0 false_alarms=0 data=1840375

sent 742037 bytes received 17334 bytes 35319.58 bytes/sec
total size is 1860318 speedup is 2.45
sysadmin@itvm28-2b:~$
```

Who are you logged into your VM as? And...what username did you use to log into **it20**? What is the reason for this? Be prepared to address this in your discussion questions.

This invocation of **rsync** requires a little explanation:

a. The options **-azv**,
   - **i.** the **a** stands for archive mode and is equivalent to **-rlptgoD**
     In other words...
     - a recursive copy
     - copying links as links
     - creating parent directories as necessary
     - preserving times, groups and owners
     - preserving devices.
   - **ii.** the **z** says data should be compressed to save space, and
   - **iii.** the **vv** stands for *very verbose* output; there's **v** for verbose, **vv** for very verbose, and **vvv** for even more verbose.

b. The option **-e ssh** specifies the remote shell to be used for doing the transfer; **ssh** is safest.

c. The first path is **/etc** , the directory on the local machine (in the previous example, that is **itvm28-2b**) that we want to back up. If it were **/etc/**, then the *contents* of **/etc** would be archived. **What is the difference? Be prepared to address this in your discussion questions.**

d. The second part, **abird@it20.it.cs.umb.edu:/it341_backups/itvm28-2b/04172012**, is the archive location – where we want to store the backup. We log in as ***abird***, so we will be challenged for ***abird***'s password. You could log in as yourself, or whatever. Notice that we ***cannot*** use the account ***sysadmin*** because it is local to your virtual machine!

Restoring Files From an Archive

To restore a file system from an archive, we simply run **rsync** backwards. One such example would be the following. (**But don't do this!!!**):

```
sudo rsync -azvv -e ssh abird@it20.it.cs.umb.edu:/it341_backups/itvm28-
                                                    2b/04172012 /tmp/etc
```

Were we to run this on **itvm28-2b**, we would copy the contents of the archive back into the **/tmp/etc** directory on **itvm28-2b** – where we could then locally copy the files into **/etc**. (**Again, DO NOT DO THIS!)**

Of course, when acting as a system administrator, we might run such commands as **root** (on both systems), *but we don't want to risk hurting ourselves here*!

When To Do Backups?

The question arises: when should we back up our file systems? It depends. For this course, it might be good to back up each host *once a week* – once each new service (e.g. *NIS*, *NFS*, and *DNS*) has been successfully installed. We can make the process easier (and so more likely that we will do it) if we write a simple ***script***.

As **sysadmin**, create and edit this file, **/usr/local/bin/backup**

You can use the command ***sudo nano /usr/local/bin/backup***

Add the following contents:

```
#!/bin/bash
#
# backup dir -- takes one argument. dir which names a repository.
rsync -azvv -e ssh /etc $2@it20.it.cs.umb.edu:/it341_backups/`hostname`/$1
```

Notice that **/usr/local/bin** is in **$PATH**:

```
sysadmin@itvm28-2b:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games

sysadmin@itvm28-2b:~$
```

Then, make ***backup*** executable,

```
sysadmin@itvm28-2b:~$ sudo chmod +x /usr/local/bin/backup
```

We can execute it with an argument to name the archive:

```
sysadmin@itvm28-2b:~$ sudo backup 04172012 abird
[sudo] password for sysadmin:
opening connection using: ssh -l abird it29.it.cs.umb.edu rsync --
server -vvlogDtprze.iLsf . /it341_backups/itvm28-2b/02142012
abird@it20.it.cs.umb.edu's password:
sending incremental file list
created directory /it341_backups/itvm28-2b/02142012
```

```
delta-transmission enabled
etc/
etc/.pwd.lock
etc/adduser.conf
etc/at.deny
etc/auto.home
etc/auto.home.bak
…
```

Now, if we log on to `it20` as **`abird`**, we can do an **`ls`** to see the archive we have just created:

```
abird@it20:~$ ls -l /it341_backups/itvm28-2b/04172012
total 4
drwxr-xr-x 3 abird abird 4096 2012-04-17 01:02 04172012
abird@it20:~$
```

Each partner should attempt the `backup` script. One partner should do this with their *own personal account*, and the other should do this with the `sysadmin` account. If your first attempt fails, then *figure out what went wrong* and *try again*.

Backing up file systems manually requires that we *remember* to do so. A better solution might be to *automate* the backup process…

Part 2 -- Automating the Backup Process

1. **Make sure your Ethernet cable is plugged into the right-hand side**
   This is the jack that communicates with our server, `it20`, whose domain has the name `it.cs.umb.edu`. (If need be, review your notes about **Ethernet jacks**)

2. **Log in to your virtual machine as** `sysadmin`

3. **Create the directory `/guests`**
   You need to create a directory that other machines can use for backups. This directory should be in the root directory `/`

   ```
   cd /
   sudo mkdir guests
   ```

4. **Make this directory available to all users**

   ```
   sudo chmod 777 guests
   ```

5. **Go to `/usr/local/bin`**

   ```
   cd /usr/local/bin
   ```

6. **Create the script autobackup.sh**

```
sudo nano autobackup.sh
```

Enter the following:

```
#!  /bin/bash
#
# shell script to perform daily backups of /etc

if [ $# -lt 2 ]
then
    echo Usage:  $(basename $0)  BACKUP_HOST  BACKUP_DIRECTORY
    exit 1
fi

backup_host=$1
backup_dir=$2
date=$(date +%F)

rsync -azvv -e ssh /etc $backup_host.it.cs.umb.edu:$backup_dir/$date
```

7. **Make this file executable:**

```
sudo chmod  755  autobackup.sh
```

8. **Logout** sysadmin

```
exit
```

9. ***ssh* into** another **virtual machine with your personal account**
   It should be a machine where you can log in without a password, as of Project 6. Also, the ***sysadmin*** of the other VM will need to have completed Step 4 above, so that you can do Step 10...

10. **Create a directory for your backups**

```
cd  /guests
mkdir  VIRTUAL_MACHINE_NAME
```

11. **Exit the ssh connection**

```
exit
```

VIRTUAL_MACHINE_NAME refers to your own VM. For example, **itvm24-2a**

OTHER_VIRTUAL_MACHINE refers to the other VM where you are placing your backups. For example, **itvm26-2a**

12. **Run `autobackup.sh`**

```
autobackup.sh  OTHER_VIRTUAL_MACHINE  /guests/VIRTUAL_MACHINE_NAME
```

13. **Check to see if the script worked:**

```
ssh  YOUR_UNIX_USERNAME@OTHER_VIRTUAL_MACHINE  'ls /guests/VIRTUAL_MACHINE_NAME'
```

Here, we are using a feature of ***ssh*** that allows you to run a Unix command on the remote machine without logging in. You should see a directory whose name is today's date in **YYYY-MM-DD** format.

14. **Remove the backup directory on the other virtual machine you just created:**

```
ssh  YOUR_UNIX_USERNAME@OTHER_VIRTUAL_MACHINE  'rm -rf /guests/VIRTUAL_MACHINE_NAME/TODAYS_DATE'
```

15. **Check to *make sure* the backup directory has been removed:**

```
ssh  YOUR_UNIX_USERNAME@OTHER_VIRTUAL_MACHINE  'ls /guests/VIRTUAL_MACHINE_NAME'
```

16. **Create a `cron` job to run `autobackup.sh`**

(BEFORE CONTINUING: Read the bullet points below, as well as my note at the bottom regarding "Making **`crontab`** work with **`ssh-agent`**"**)**

After you have...

- read through the end of this document ***completely***
- and completed the preparatory steps for the **`keychain`** utility

...run ***`crontab`***

```
crontab -e
```

- If you have not created **`cron`** jobs on this machine yet, then there should be nothing in this file.
- Create an entry to run **`autobackup.sh`** in 5 minutes.
- To do this, you will have to look at the time and pick a time 5 minutes ahead.
- Remember that **`cron`** uses a 24-hour clock, so 1 PM is written at 13.
- (To help with this, research the **format of a `crontab` File**)
- The command part of the cron job should look like this:

```
/usr/local/bin/autobackup.sh  OTHER_VIRTUAL_MACHINE  /guests/VIRTUAL_MACHINE_NAME
```

17. **Check the other virtual machine to make sure the backup worked:**

```
ssh  YOUR_UNIX_USERNAME@OTHER_VIRTUAL_MACHINE  'ls /guests/VIRTUAL_MACHINE_NAME'
```

18. **Take a snapshot of the current state of your virtual machine:**

Select *VM* ⟶ *Snapshot* ⟶ *Take snapshot*
In the *Name* box type

```
Project 4, Part II
```

then click *Take Snapshot*.

Making `crontab` work with `ssh-agent`

I have seen that **`crontab`** and **`ssh-agent`** do not always play well together. Specifically, even if you have an **`ssh-agent`** running, it may not recognize it when running autobackup.sh from crontab. To make this work, I recommend the following steps:

1. On your VM, as **sysadmin**, execute the following command:
   ```
   sudo apt-get install keychain
   ```
2. On your VM, as **sysadmin**, add the following line to your script code, right above the **`rsync`** line:
   ```
   source $HOME/.keychain/$(/bin/hostname)-sh
   ```
3. On your VM, as *yourself* (i.e., your personal account), execute the following command:
   ```
   keychain $HOME/.ssh/YOUR_PRIVATE_KEY
   ```
   (**`YOUR_PRIVATE_KEY`** will probably be either **`id_dsa`** or **`id_rsa`**)
4. Finally, as *yourself*, create the task in ***crontab***, if you have not done so already. It should work now because the previous steps created an **ssh agent** that should persist after you log off (so long as your VM is still up and running).
5. If these steps do not work, then see if you can troubleshoot the issue and get it to work. (If so, document this in your admin log.) If that fails, create a second set of keys (this time, no passphrase), place the public key into your **`authorized_keys`** file, and remove the *keychain* line from the script.