How To Create An Off-Site Backup Of Your Site With Rsync On Centos 6

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: http://faq.asphosthelpdesk.com/article.php?id=286

There are two ways to do backup: incremental and full backups. A full backup copies all the data, while the snapshot only handles the changes since the last update.

Full Backup

Full backup typically does the following:

- Make an archive of all the files in a folder
- Copy the resulting archive to a remote server.

As noted, our data will reside in /var/www/wordpress directory. We will designate a backup folder location.

```
mkdir -p /backup/wordpress
```

The above command will create a /backup directory, and a /backup/wordpress directory, if they do not exist. To create a full snapshot of our data, we use the linux tool called tar.

```
tar -czf /backup/wordpress/initial_backup.tar.gz
/var/www/wordpress
```

Tar will **c**reate a g**z**ip archive in a **f**ile *initial_backup.tar.gz*. We could add a **v**flag (so we get*tar -czvf*) if we want a verbose output (list of filenames). We name the file initial_backup.tar.gz so that we know that this is an initial backup, it is archived with tar and iz is zipped in a gzip format. Tar will use whatever arguments we provide as a source, in our case it will backup */var/www/wordpress*directory. We could pass two or more arguments, whether they be files or folders: ie.

```
tar -czf /backup/cms_systems_backup.tar.gz /var/www/wordpress
/var/www/drupal /var/www/joomla
```

The last command would backup all of our installed cms systems.

Now, for our future backups, we may want to add a date when the backup was taken:

```
tar -czf /backup/wordpress/wordpress-`date '+%m%d%y'`.tar.gz
/var/www/wordpress
```

Lets' see what we have now:

```
[root@Backup ~]# ls -l /backup/wordpress/
total 9760
```

```
-rw-r--r-- 1 root root 4995743 Apr 17 12:16 initial_backup.tar.gz
-rw-r--r-- 1 root root 4995743 Apr 17 12:25 wordpress-041713.tar.gz
[root@Backup ~]#
```

We have two files, one called initial_backup, one called wordpress-041713 (for April 17th 2013, the time of this writing). Now, to schedule this daily, we need to create a crontab entry. Crontab is a linux task scheduler: we tell it when to do something and what task to actually do. Anyway, we open up the crontab editor:

```
EDITOR=nano crontab -e
```

It will open a crontab file in a text editor. By default, DO CentOS images include vim as editor, which requires a bit of setup, so we have used a simpler editor for this purpose, called nano editor. We could have just used the default editor with:

```
crontab -e
```

Now we need to tell cron to backup, say, every day at 3.30am, when there is hopefully not a lot of traffic. We will also tell it to email us any findings. We put this content into crontab:

```
MAILTO=email@example.com
30 3 * * * /bin/tar -czf /backup/wordpress/wordpress-`date
+\%m\%d\%y`.tar.gz /var/www/wordpress
```

We save the file with CTRL-X and confirm with Y and Enter. The above command will tell linux to repeat our command every day at 3:30. We also told cron to email us with the results. You will receive the message: */bin/tar: Removing leading `/' from member names*, as a sign that everything went through. In case of errors, the message will contain the details so we can fix this. So, a daily backup is ready and working.

Copy the Backups to Another Remote Server

To copy the backups to another remote server, we will use scp - secure copy. First, we need to generate an SSH key:

```
ssh-keygen
```

We can leave the passphrase empty for now, and use the */root/.ssh/id_rsa_backup*key file (or /home/username/.ssh/id_rsa_backup if we're not running as root). Now we can check the public key part:

```
cat .ssh/id_rsa_backup.pub
```

We need to copy this public part of SSH key to the remote server, to a file authorized_keys. I assume we already have a remote server called *backup.example.com* and a user *backup*. This could be an empty newly created, but the user has to be created beforehand. We will only do this part once.

```
scp .ssh/id_rsa_backup.pub
```

```
backup@backup.example.com:/home/backup/backup_key.pub
```

We'll be prompted for the backup users' password. We copied the file, now let's add it where it should be, in authorized_keys. I cannot assume that this user already has the file and folder set up, so lets check that info:

```
ssh backup@backup.example.com "mkdir -p /home/backup/.ssh"
ssh backup@backup.example.com "chmod 700 /home/backup/.ssh"
ssh backup@backup.example.com "touch /home/backup/.ssh/authorized_keys"
ssh backup@backup.example.com "chmod 600 /home/backup/.ssh/authorized_keys"
ssh backup@backup.example.com "mkdir -p /home/backup/backups"
```

The few commands above created a directory for SSH to work with, if it didn't exist, and also the authorized_keys file, which needs to be present for backups to work. We also created a *backups* directory to store our files to. Now what is left is to copy our public key to that file.

```
ssh backup@backup.example.com "cat /home/backup/backup_key.pub >>
/home/backup/.ssh/authorized_keys"
```

Now we can use this key to copy stuff in the future.

Now, let's copy the backup file over there:

```
scp -i .ssh/id_rsa_backup
/backup/wordpress/wordpress-041713.tar.gz
backup@backup.example.com:/home/backup/backups
```

If our key setup was correct, the file will be copied and we won't be asked for passwords. We can check that the file is really there:

```
ssh backup@backup.example.com "ls -l /home/backup/backups"
```

Ok, we can now schedule this action to the crontab too. Start the crontab editor again:

```
EDITOR=nano crontab -e
```

We will now alter our backup line: we want to add info to copy our backup archive when it's created. So, we append the new command so that it looks like this:

```
30 3 * * * /bin/tar -czf /backup/wordpress/wordpress-`date
+\%m\%d\%y`.tar.gz /var/www/wordpress;/usr/bin/scp -i
/root/.ssh/id_rsa_backup /backup/wordpress/wordpress-`date
+\%m\%d\%y`.tar.gz
backup@backup.example.com:/home/backup/backups
```

Note: this is not the usual way to do it, it would be better to setup a script which does all the tasks and then schedule the script. But for the brevity of this article, we'll use that form.

Incremental Backup

But what if we have our own backup software at another server? We just want to synchronize the data over, and then leave the other server to do the backup work. In addition, we want to preserve file stamps. Then we use **rsync**. The use case here is that we want to just incrementally copy over everything from */var/www/wordpress*to a remote server, this time to a */home/backup/snapshots/wordpress*directory. Here is a simple command to do all that:

```
ssh backup@backup.example.com "mkdir -p /home/backup/sync"
rsync -avz --delete -e "ssh -i /root/.ssh/id_rsa_backup"
/var/www/wordpress backup@backup.example.com:/home/backup/sync
```

The first line creates a snapshot directory and the second copies the *changed* files over. That means the files that were modified, newly created or deleted. We can schedule it in cron too:

```
EDITOR=nano crontab -e
```

The crontab line should look like this:

```
30 3 * * * /usr/bin/rsync -avz --delete -e "ssh -i
/root/.ssh/id_rsa_backup" /var/www/wordpress
backup@backup.example.com:/home/backup/sync
```

Now our remote server will always have a fresh synced copy of the data, and we can do the backup there.

Backup Database

We can also backup our database. First, we want to dump the data. If we followed the wordpress install guide, we also have a database *wordpress*, accessed by user *wordpressuser* with password *password*. We can do the initial dump like this:

```
mkdir /backup/mysql
mysqldump < wordpress -u wordpressuser -ppassword | gzip >
/backup/mysql/initial.sql.gz
```

This command created a initial.sql.gz gziped SQL file. To do it on a daily basis, we can schedule it in cron, like before. Our resulting cron line should look like this:

```
0 4 * * * /usr/bin/mysqldump < wordpress -u wordpressuser
-ppassword | /bin/gzip > /backup/mysql/mysql--`date +\%m\%d\%y`.sql.gz
```

Now we could also combine it with scp or rsync to copy it remotely.

```
0 4 * * * /usr/bin/mysqldump < wordpress -u wordpressuser
-ppassword | /bin/gzip > /backup/mysql/mysql-`date +\%m\%d\%y`.sql.gz;
/usr/bin/scp -i /root/.ssh/id_rsa_backup /backup/mysql/mysql-`date
```

```
+\%m\%d\%y`.sql.gz  backup@backup.example.com:/home/backup/
```

With this setup, we have a basic backup of our data set up for a case of emergency.