AUTOMATED BACKUPS

There are few things more satisfying than a good backup

Introduction

Backups rely for the most part, on *unix at least (something called "cron" jobs), wikipedia defines cron as follows:-

Cron is a time-based job scheduler in Unix-like computer operating systems. The name cron comes from the word "chronos", Greek for "time". Cron enables users to schedule jobs (commands or shell scripts) to run periodically at certain times or dates. It is commonly used to automate system maintenance or administration, though its general-purpose nature means that it can be used for other purposes, such as connecting to the Internet and downloading email.

Cron is without doubt the workhorse of the *unix system it schedules all the jobs you want completed at a certain time. You need to think of cron as nothing more than an alarm clock. The only difference with this alarm clock is that multiple alarms can be set to run multiple single instruction commands or scripts. You perhaps better off to think of cron as a table this table allows instructions by way of digits, wild cards and commands that can be entered on one or more instances.

Crontab

In our opinion the best and easiest way to get a cron job to run is using crontab, crontab itself is command invoked from the command line however you do need to specify the sudo command before it and the file option suffix after the command. Crontab can be run as any user but it is more common to run that user as "root" or root user the reason for this is quite simply because root user can access more aspects of the file system than an ordinary user.

You can use crontab with one of three possible options

#> sudo crontab -l

The above command will list cron jobs so entered using the crontab command

#> sudo contab -e

The above command will edit cron jobs.

#> sudo contab -r

The above command will remove cron jobs all of them so please use this command with extreme caution. Use the edit command to remove individual cron jobs.

Using crontab -e for the first time may show you something like this.

```
derek@derek-laptop:~$ crontab -l
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow command
0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
```

The above is only an example so please disregard any commands stated.

The thing to recognise about from the above is this and breaks down as follows

m h dom mon dow command

- # - Remark (Everything after this initial remark is ignored, however it does hold useful information regarding syntax)
- m - Specifies the MINUTE the cron will run (digit or wild card required)
- h - Specifies the HOUR the cron will run (digit or wild card required)
- dom - Specifies the DAY OF MONTH the cron will run (digit or wild card required)
- mon - Specifies the MONTH the cron will run (digit or wild card required)
- dow - Specifies the DAY OF WEEK the cron will run (digit or wild card required)
- command - Specifies the COMMAND the cron will run (command or script file)

Entry	Description	Equivalent To
`@yearly (or @annually)`	Run once a year	`0 0 1 1 *`
`@monthly`	Run once a month	`0 0 1 * *`
`@weekly`	Run once a week	`0 0 * * 0`
`@daily (or @midnight)`	Run once a day	`0 0 * * *`
`@hourly`	Run once an hour	`0 * * * *`
`@reboot`	Run at startup	

Say you wanted a cron job whatever it is, to run at a specific time everyday 12.00 the command you would enter into the cron table or crontab is as follows:-

0 12 * * * /dir/command

The zero denotes the zero minute of the hour (this could be replaced by any digit upto 59), 12 will be the top of the hour "noon" and the three asterisk will effectively run every day of month, every month and every day of week what follows is then the command to be run either as root or as user. User crons are different because these will only run as the specified user so any commands to be initiated must exist within that users own home directory, as root user - root can run any command from any directory within the *unix file system.

Mysql Application Backup

Your database is the most precious application you have as without the data in it you can not recover from any disasters or failures, so it makes sense to backup as much of this data as possible. The problem however is how - "mysql" and other databases have utilities that allow backups from the command line and as an example may look like this:-

#> mysqldump -ac --add-drop-table -uroot -h localhost -p <DATABASE> > <FILE>

Command explination

- mysqldump is the command used to backup mysql
- -ac this is an option specified
- --add-drop-table - This adds a DROP TABLE statement before each CREATE TABLE statement.
- -uroot this can be specified as ("-u root") which is mysql root user
- -p this expects the password for root user to be specified and can be included on the command line after the -p
- -h specifies the hostname not usually required unless collecting from different accessible hosts
- <DATABASE> without any chevrons specified the database to be backed up, it is vital the exact spelling is used including capitals
- > use the redirect to form a path from the above database to the file
- <FILE> again without chevrons specify the file to be backed up to

The above command when written and initiated will copy in "sql" format from a specific named <DATABASE> to a specified named <FILE>it is common to add the suffix of **".sql"** to the end of the backed up named file. In it's current format though it does not copy all the databases it only copies one single named database, this has the disadvantage then that in advance you at least you need to know the name of each separate database.

It should be noted that it is possible to copy all the databases in one go, this however can do more harm than good, as your only option would be to restore all the database's also in one go. If you are running multiple databases at the same time this can present a problem. With databases being edited at different rates it is important to restore the database that is corrupt rather than all the databases. It is bad enough to restore and update one database imagine having to restore all the databases and then to bring them all back up to date, here then the margin for error is great.

Clearly then we need to copy each of the databases at the same time but as separate files, this would allow us to restore only the database that needs to be restored following a failure, breakdown or corruption. The problem though is that neither of the commands mentioned above will on it's own perform this particular function. What is needed then is a means to copy all the databases but as separate files, no one command will do this from the command line so we need to produce a script, a "bash script", the script can be called anything you like. Using a script though does allow us to use variables so we can store a list of databases regardless of who or when they where created and merge this with the command above to provide us with separate files from each database with which to restore from.

Below this article is an attachment which is an example of a script we use to backup our databases. The script is a bash shell script, by that we mean the script was written using the shell script "bash". Download the attachment file and open it to view it's contents, right at the beginning of the file is a line that tells any *unix to use the "bash" interpreter.

!#/usr/bin/bash

This is important because different shells, and for *unix there are many, tells the system how to use the commands named in the script. The variable for username, password and so on are declared towards the beginning of the script and are the only items that should be changed in the script. We did not write this file but we did modify this for our own needs, you may see a lot of what appears to be duplication and this was deliberate attempt to disable parts of the original script and modify with our own requirements.

This script is one of the best we have seen since it explains in meticulous detail practically each line written. In fact this script is almost usable as it is, you need only edit the file to alter the two variables for username and password for root user on mysql. The only other thing you might want to change is destination the script file places the backups. WARNING the backup files are written to a protected area that can only be accessed as user root. Click on the attachment file which should launch to a new browser window then copy the text shown and paste this into an editor name the file what you wish, using a dot sh (.sh) extension denotes the file to be script file of some sought.

Source : http://www.soslug.org/wiki/automated_backups