Bash: how to set up a basic incremental backup system

Implementing an incremental backup system on a remote server is essential to ensure data security, minimize storage space usage, and optimize execution times. Using Bash and cronjobs, you can automate backup processes, ensuring that they run regularly without manual intervention. In this article, we will explore the basic steps to set up an incremental backup system.

An **incremental backup** is a type of backup that stores only the files that have been changed or added since the previous backup. Compared to a full backup, it requires less disk space and reduces execution time.

For this purpose, we will use `rsync`, a powerful tool that synchronizes files and directories between local and remote servers efficiently, supporting incremental transfer.

Let's start by making sure that `rsync` is installed on both the local and remote servers. You can install it with the following commands:

```
sudo apt update
sudo apt install rsync
```

To avoid having to enter the password every time we perform the backup, we configure SSH authentication with public key.

```
ssh-keygen -t rsa
```

We then copy the public key to the remote server:

```
ssh-copy-id user@remote_server
```

We create a Bash script that performs the incremental backup. Suppose we want to backup the directory `/var/www` (containing web files) to a remote server.

```
#!/bin/bash

# Configuration
SRC_DIR="/var/www/" # Source directory
DEST_DIR="/path/backup/" # Destination directory on
remote server
REMOTE_USER="user" # Username of remote server
REMOTE_HOST="remote_server" # Host of remote server
BACKUP_DATE=$(date +'%Y-%m-%d') # Current date for
backup
LOG_FILE="/var/log/backup.log" # Log file

# Incremental backup using rsync
rsync -avz --delete --link-dest=${DEST_DIR}/latest
${SRC_DIR}
${REMOTE_USER}@${REMOTE_HOST}:${DEST_DIR}/$BACKUP_DAT
E >> ${LOG_FILE} 2>&1

# Update the latest backup
ssh ${REMOTE_USER}@${REMOTE_HOST} "rm -rf
```

```
${DEST_DIR}/latest && ln -s
${DEST_DIR}/${BACKUP_DATE} ${DEST_DIR}/latest"
```

Script explanation:

- **SRC_DIR**: the local directory we want to save.
- **DEST_DIR**: the backup directory on the remote server.
- **BACKUP_DATE**: it is used to create a new folder for each incremental backup, using the current date.
- **`rsync -avz`**: synchronizes files from `SRC_DIR` to `DEST_DIR` on the remote server. The `--link-dest` flag refers to the last backup, so only changed or new files are copied.
- **`ssh`**: This is used to update the `latest` symbolic link on the remote server, so that it always points to the last backup performed.

Then let's make the script executable:

```
chmod a+x backup.sh
```

The next step is to automate the execution of the backup script using cron, the Linux scheduling daemon.

Edit the crontab file for the root user or the specific user (if you want to perform backups on a remote server):

```
crontab -e
```

Add a new line to run the script every day at 02:00 for example:

```
0 2 * * * /path/backup.sh
```

This command tells `cron` to run the backup script every day at 02:00. You can change this schedule to suit your needs.

It is good practice to check the status of your backups and logs regularly. You can check the log file created by the script:

```
cat /var/log/backup.log
```

Make sure there are no errors and that the files are copied correctly to the remote server.

Optimization and Security

- **Compression**: `rsync` supports the `-z` option to compress files during transfer, reducing bandwidth usage.
- **Encryption**: Use SSH to ensure that data is encrypted during transfer.
- **Off-site backups**: It is good practice to keep a copy of your backups in a different geographical location, in case of natural disasters or hardware failures.
- **Automatic deletion**: You can add a function to automatically delete the oldest backupshi of a certain number of days, using a `find` command in your script.

Example:

```
ssh ${REMOTE_USER}@${REMOTE_HOST} "find ${DEST_DIR} -
```

```
type d -mtime +30 -exec rm -rf {} \;"
```

This command deletes backups older than 30 days.

Conclusion

With this guide, you have set up an incremental backup system using Bash, `rsync` and `cronjob`. This solution allows you to save disk space and bandwidth compared to a daily full backup and ensures that your critical data is safely stored on a remote server. Customize the cron script and schedule to suit your business needs to maximize the efficiency of your backup system.