Abdullah ÇAY

Bash and Linux - Cron, environment variables and other tools (EN)

45 minutes   Normal

Machine status

Ubuntu Server 18.04 LTS

SSD Volume Type

64-bit x86

Stopped

Connect

Reset   Start

DataScientest • com

Introduction to Linux

Cron, environment variables and other tools

Cron

Cron comes from greek, "Chronos" meaning time. It is a process that will execute different tasks at a given frequency: for example, a process can be set to run every hour thanks to Cron.

To define the processes and the schedule, we use a program called Crontab which uses a file containing process configuration. Cron is often used to perform frequent tasks (clearing logs or cache, sending reports, ...).

A process that is run by Cron is called a Cron Job.

Cron Job syntax

To tell Cron when to run a job, we must specify the schedule. This can be done by running a command similar to this one.

```
* * * * * command/script
```

In this first part, each star correspond to a different timing

- first `*` are the Minutes (0-59) ;
- second `*` are the Hours (0-23) ;
- third `*` are the Days of the month (1-31) ;
- fourth `*` are the Month of the year (1-12) ;
- fifth `*` are the Days of the Week (0-6, from sunday to saturday).

For example, this job is going to be executed at 5:15 every day:

```
15 5 * * * command/script
```

> Can you guess when this script is going to be run

```
0 12 * * 1 command/script
```

This script is going to be run every monday at 12.

There are other possibilities:

- `*` means every value
- `,` can be used to specify mutliple values
- `-` means a range of value (`2-5` is equivalent to `2,3,4,5`)
- `/` can be used to specify a step value: (`*/15 * * * *` means every 15 minutes)

This website is interesting to read Cron Job expressions and to train to get some.

Finally, note that we can use `jan`, `feb`, ... to specify months or `mon`, `tue`, ... to specify days of the week.

Practice

We are going to create a job that runs every minute.

```
#!/bin/bash
echo "Cron Job" >> ~/data_cron.txt
```

Change the execution rights

```
chmod +x ~/my_cron_job.sh
```

To add the Cron Job, we must run `crontab -e`. It will offer us to choose an editor (1 is for `nano`). And then we can paste our Cron expression:

Paste the following line: `* * * * * ~/my_cron_job.sh`

Now your job will be executed every minute. Let it run a few minutes and check regularly the content of the `~/data_cron.txt` file. We can also use `crontab` to perform other tasks:

- `crontab -e` : allows you to edit current user crontab file
- `crontab -l` : allows you to print current user crontab file
- `crontab -u [username]` : allows you to modify another user crontab file
- `crontab -r` :allows you to remove current user crontab file

Environment variables

In an OS, we can define environment variables. Environment variables are defined when the system is started or when a session is started. They can be used to define path to object, to define the language of the system, ...

There are important variables that are defined:

Run the following command

```
# paths to executable files
echo $PATH
# path to current user home
echo $HOME
# current user
echo $USER
# current working diretory
echo $PWD
```

Note that we can run `printenv` to print every environment variables.

`PATH` is one of the most important and it is interesting to understand why: when you want to run a Python script, you do not use the full path to Python executable. You only use `Python3`. This is because the `PATH` environment variable contains the directory where this executable is (`/usr/bin`). When you type Python3, Linux will look into the directories inside the `PATH`. If it exists, it will run it, else it will return a `command not found error`.

To define an environment variable, we use `export`:

Run the following command

```
# creating an environment variable
export MY_ENVIRONMENT_VARIABLE=my_value

# trying to retrieve the value with printenv
printenv | grep MY_ENVIRONMENT_VARIABLE
```

Run the following command and notice the difference

```
# creating a variable
MY_VARIABLE=my_value

# trying to retrieve the value with printenv
printenv | grep MY_VARIABLE

# printing the value of MY_VARIABLE
echo $MY_VARIABLE
```

Be aware that this variable is not eternal and will be removed if you close this session.

Exit the current session with `exit`, reconnect and print the content of `MY_ENVIRONMENT_VARIABLE`

Run the following command

```
# creating an environment variable
export MY_ENVIRONMENT_VARIABLE=my_value

# trying to retrieve the value with printenv
printenv | grep MY_ENVIRONMENT_VARIABLE

# deleting environment variable
unset MY_ENVIRONMENT_VARIABLE

# trying to retrieve the value with printenv
printenv | grep MY_ENVIRONMENT_VARIABLE
```

To define a variable that will be kept through sessions, you could define it in the `~/.bashrc` that is run everytime the console is opened.

Other tools

Possibilities are unlimited with Linux but some tools are more useful than others. In this part, we are going to see some of them.

apt

`apt` is a repository of programs in Debian distributions. It can be use to install new programs. The main command to know are:

- `apt update`: to update repository information
- `apt upgrade`: to upgrade programs according to the information inside the repository
- `apt install program_name`: to install a program `program_name`

For example, if we want to use `cowsay`, we need to install it:

Run the following command

```
cowsay "Hello world"
```

You will receive a `command not found error`.

To install `cowsay`, run the following command

```
sudo apt update
sudo apt install -y cowsay
```

`cowsay` is now installed: try `cowsay "Hello world"`. This program is not very useful but we have seen how to install a program with `apt`. Note that in older versions of Ubuntu, `apt` is called `apt-get` so you may find a lot of documentation referring to `apt-get`.

wget

`wget` is a very interesting tool, installed by default, that will let you download files from the internet. You can use it followed by the url of the object: here the object is a simple json file with random data.

```
# retrieving data from the internet
wget https://assets-datascientest.s3.eu-west-1.amazonaws.com/de/resources/data.json

# changing the name of the file
wget https://assets-datascientest.s3.eu-west-1.amazonaws.com/de/resources/data.json -O my_file.json

# avoiding using cache
wget https://assets-datascientest.s3.eu-west-1.amazonaws.com/de/resources/data.json --no-cache
```

tar

`tar` is a utility program that allows you to create archives or to decompress archives. Archives are basically mulitple files grouped as one. Some are compressed and other not.

Abdullah ÇAY

```
# creating a directory
mkdir my_folder

# creating files
touch my_folder/file1 my_folder/file2

# creating an archive
tar cvf my_folder.tar my_folder

# moving the archive
mv my_folder.tar ./my_folder

# changing current directory
cd my_folder

# decompressing the archive
tar xvf ./my_folder.tar

# looking at the structure
ls -R ..
```

In this example, we have created an archive and decompressed it in another folder. We can see how this can be useful to manipulate multiple files as one. Here is the meaning of the flags:

- `c` : creates the archive
- `x` : expands the archive
- `f` : uses the file given as an argument
- `v` : uses verbose mode

date

The basic usage of `date` is to provide the date of the system at the current time. It has a lot of options that you can see with `date --help`

> Try `date`

curl

`curl` is not installed by default but is very useful to make HTTP requests over the internet. It is so useful that we have installed it for you on you machine. You can pass the url and the HTTP method that you want to use:

```
# GET query on https://www.example.com
curl -X GET https://www.example.com

# GET query on https://jsonplaceholder.typicode.com/ API
curl -X GET https://jsonplaceholder.typicode.com/users/1
```

Validate