BY BOBBY ILIEV

Introduction to Bash Scripting

FOR DEVELOPERS

Mục lục

Giới thiệu về cuốn sách **8**
Về tác giả 9
Nhà tài trợ 10
Công cụ tạo PDF Ebook 12
Công cụ tạo ePub Ebook 13
Bìa sách 14
Giấy phép 15

Nhập môn Bash scripting **16**

Cấu trúc của Bash **18**

Bash Hello World **20**

Biến trong Bash **22**

Nhập dữ liệu từ người dùng trong Bash **26**

Chú thích trong Bash **28**

Tham số trong Bash **29**

Mảng trong Bash **32**
Cắt mảng (Array Slicing) 34
Cắt chuỗi 35

Biểu thức điều kiện trong Bash **37**
Kiểm tra file 38
Biểu thức kiểm tra chuỗi 40
Toán tử số học 42
Toán tử trạng thái kết thúc 44

Câu lệnh điều kiện trong Bash **45**
Câu lệnh If 46
Câu lệnh If Else 47
Câu lệnh Switch case 50
Kết luận 52

Vòng lặp trong Bash **53**
Vòng lặp For 54
Vòng lặp While 56
Vòng lặp Until 58
Continue và Break 59

Hàm trong Bash **62**

Debugging, testing và shortcuts **64**

Tạo lệnh bash tùy chỉnh **67**
Ví dụ 68
Lưu lại các thay đổi 70
Liệt kê tất cả các alias có sẵn 71
Kết luận 72

Viết script Bash đầu tiên của bạn **73**

Lên kế hoạch cho script 74
Viết script 75
Thêm chú thích 76
Thêm biến đầu tiên của bạn 77
Thêm hàm đầu tiên của bạn 78
Thử thách: Thêm các hàm mới 80
Script mẫu 81
Kết luận 83

Tạo menu tương tác trong Bash 84
Lên kế hoạch cho chức năng 85
Thêm một số màu sắc 87

Thêm menu 88
Kiểm thử script 90
Kết luận 93

Thực thi script BASH trên nhiều máy chủ từ xa 94
Điều kiện tiên quyết 95
Script BASH 96
Chạy Script trên tất cả các máy chủ 98
Kết luận 99

Làm việc với JSON trong BASH sử dụng jq 100
Lập kế hoạch cho script 101
Cài đặt jq 102
Phân tích cú pháp JSON với jq 104
Lấy phần tử đầu tiên với jq 106
Chỉ lấy giá trị cho khóa cụ thể 107
Sử dụng jq trong script BASH 108

Kết luận 111

Làm việc với API Cloudflare bằng Bash 112

Điều kiện tiên quyết 113

Thử thách - Yêu cầu của script 114

Script mẫu 115

Kết luận 117

Script BASH để tóm tắt các log truy cập của NGINX và Apache 118

Yêu cầu của script 119

Script mẫu 120

Chạy script 121

Tìm hiểu về kết quả đầu ra 122

Kết luận 123

Gửi email bằng Bash và SSMTP 124

Điều kiện tiên quyết 125

Cài đặt SSMTP 126

Cấu hình SSMTP 127

Gửi email bằng SSMTP 128

Gửi Tệp đính kèm với SSMTP (tùy chọn) 129

Kết luận 130

Tạo mật khẩu bằng Script Bash 131

:warning: Cảnh báo bảo mật 132

Tóm tắt script 133

Yêu cầu tiên quyết 134

Tạo mật khẩu ngẫu nhiên 135

Script 137
Script bản đầy đủ: 138
Kết luận 139
Được đóng góp bởi 140

Redirections(Chuyển hướng) trong Bash 141

Sự khác biệt giữa Pipe và Redirections 142

Redirection trong Bash 143
STDIN (Đầu vào Chuẩn) 144
STDOUT (Đầu ra chuẩn) 146
STDERR (Lỗi Tiêu Chuẩn) 148

Piping 150

HereDocument 152

HereString 154
Tóm tắt 155

Cài đặt tự động WordPress trên LAMP bằng BASH 156

Điều kiện tiên quyết 157

Lên kế hoạch cho các chức năng 158

Script 160

Script đầy đủ 167

Tóm tắt 171

Lời kết **172**

Giới thiệu về cuốn sách

- **Phiên bản này được xuất bản vào ngày 08/02/2024**

Đây là một hướng dẫn mã nguồn mở về lập trình Bash, giúp bạn học những kiến thức cơ bản về scripting Bash và bắt đầu viết các script Bash tuyệt vời để tự động hóa các tác vụ SysOps, DevOps và phát triển hàng ngày. Dù bạn là kỹ sư DevOps/SysOps, lập trình viên hay chỉ là người đam mê Linux, bạn đều có thể sử dụng các script Bash để kết hợp các lệnh Linux khác nhau và tự động hóa những công việc lặp đi lặp lại hàng ngày, từ đó tập trung vào những việc hiệu quả và thú vị hơn.

Hướng dẫn này phù hợp cho bất kỳ ai đang làm việc như một lập trình viên, quản trị hệ thống hoặc kỹ sư DevOps và muốn học những kiến thức cơ bản về lập trình Bash.

13 chương đầu tiên sẽ tập trung hoàn toàn vào xây dựng nền tảng vững chắc về lập trình Bash, sau đó các chương còn lại sẽ cung cấp cho bạn một số ví dụ và script thực tế.

Về tác giả

Tên tôi là Bobby Iliev, và tôi đã làm việc như một Kỹ sư DevOps Linux từ năm 2014. Tôi là một người yêu thích Linux và ủng hộ triết lý phong trào mã nguồn mở. Tôi luôn cố gắng làm những điều mà tôi chưa biết để học hỏi cách thực hiện chúng, và tôi tin tưởng vào việc chia sẻ kiến thức.

Tôi nghĩ điều quan trọng là luôn giữ tính chuyên nghiệp và bao quanh mình bằng những người tốt, làm việc chăm chỉ và đối xử tốt với mọi người. Bạn phải thể hiện ở mức độ cao hơn những người khác một cách nhất quán. Đó là dấu hiệu của một chuyên gia thực thụ.

Để biết thêm thông tin, vui lòng truy cập blog của tôi tại https://bobbyiliev.com, follow me on Twitter @bobbyiliev_ and YouTube.

Nhà tài trợ

Cuốn sách này được thực hiện nhờ sự hỗ trợ của những công ty tuyệt vời sau!

Materialize

Cơ sở dữ liệu luồng cho phân tích thời gian thực.

Materialize là một cơ sở dữ liệu phản ứng cung cấp cập nhật view theo lũy tiến. Materialize giúp các nhà phát triển dễ dàng xây dựng với dữ liệu luồng bằng SQL tiêu chuẩn.

DigitalOcean

DigitalOcean là một nền tảng dịch vụ đám mây cung cấp sự đơn giản mà các nhà phát triển yêu thích và các doanh nghiệp tin tưởng để chạy các ứng dụng sản xuất ở quy mô lớn.

Nó cung cấp các giải pháp tính toán, lưu trữ và mạng có tính khả dụng cao, bảo mật và có thể mở rộng, giúp các nhà phát triển xây dựng phần mềm tuyệt vời nhanh hơn.

Được thành lập vào năm 2012 với các văn phòng tại New York và Cambridge, MA, DigitalOcean cung cấp giá cả minh bạch và hợp lý, giao diện người dùng tường minh và một trong những thư viện tài nguyên mã nguồn mở lớn nhất hiện có.

Để biết thêm thông tin, vui lòng truy cập https://www.digitalocean.com or follow @digitalocean trên Twitter.

Nếu bạn mới sử dụng DigitalOcean, bạn có thể nhận được $200 tín dụng miễn phí và tạo các máy chủ riêng thông qua liên kết giới thiệu

này:

Tín dụng $200 miễn phí cho DigitalOcean

DevDojo

DevDojo là một nguồn tài nguyên để học tất cả mọi thứ về phát triển web và thiết kế web. Học trong giờ nghỉ trưa hoặc thức dậy và thưởng thức một tách cà phê cùng chúng tôi để học điều gì đó mới.

Tham gia cộng đồng nhà phát triển này, và chúng ta có thể cùng nhau học hỏi, xây dựng và phát triển.

Join DevDojo

Để biết thêm thông tin, vui lòng truy cập https://www.devdojo.com or follow @thedevdojo trên Twitter.

Công cụ tạo PDF Ebook

Ebook này được tạo bởi Ibis do Mohamed Said phát triển.

Ibis là một công cụ PHP giúp bạn viết sách điện tử bằng markdown.

Công cụ tạo ePub Ebook

Phiên bản ePub được tạo bởi Pandoc.

Bìa sách

Bìa của ebook này được tạo bằng Canva.com.

Nếu bạn cần tạo một đồ họa, poster, thiệp mời, logo, bài thuyết trình – hoặc bất cứ thứ gì trông đẹp mắt — hãy thử Canva.

Giấy phép

Lưu ý: Phần giấy phép MIT dưới đây được giữ nguyên bản tiếng Anh để đảm bảo tính chính xác về mặt pháp lý

Nhập môn Bash scripting

Chào mừng đến với hướng dẫn nhập môn Bash! Trong khóa học **siêu tốc về bash** này, bạn sẽ học các **kiến thức cơ bản về Bash** để có thể bắt đầu viết các script(kịch bản) Bash của riêng mình và tự động hóa các công việc hàng ngày.

Bash là một loại shell và ngôn ngữ dòng lệnh của Unix. Nó được sử dụng rộng rãi trên nhiều hệ điều hành khác nhau, và cũng là trình thông dịch lệnh mặc định trên hầu hết các hệ thống Linux.

Bash là viết tắt của Bourne-Again SHell. Giống như các shell khác, bạn có thể sử dụng Bash tương tác trực tiếp trong terminal(giao diện dòng lệnh) của bạn, và cũng, bạn có thể sử dụng Bash như bất kỳ ngôn ngữ lập trình nào khác để viết các script. Cuốn sách này sẽ giúp bạn học các kiến thức cơ bản về lập trình Bash bao gồm: biến Bash, dữ liệu nhập từ người dùng, chú thích, tham số, mảng, biểu thức điều kiện, câu lệnh điều kiện, vòng lặp, hàm, gỡ lỗi, và kiểm thử.

Các script Bash rất hiệu quả để tự động hóa các công việc lặp đi lặp lại và giúp bạn tiết kiệm thời gian. Ví dụ, hãy tưởng tượng làm việc trong một nhóm năm lập trình vên cho một dự án cần thiết lập môi trường phức tạp. Để chương trình hoạt động chính xác, mỗi lập trình viên phải tự cài đặt môi trường một cách thủ công. Đó là cùng một việc rất dài và phức tạp (cài đặt môi trường) được lặp lại ít nhất 5 lần. Đây là lúc bạn và các script Bash đến để giải cứu tình hình! Thay vào đó, bạn tạo một tệp văn chứa tất cả các hướng dẫn cần thiết và chia sẻ nó với đồng nghiệp. Và bây giờ, tất cả những gì họ cần làm là chạy script Bash đó và mọi thứ sẽ được thiết lập tự động cho họ.

Để viết các script Bash, bạn chỉ cần một terminal UNIX và một trình

soạn thảo văn bản như Sublime Text, VS Code, hay một trình soạn thảo dựa trên terminal như vim hoặc nano.

Cấu trúc của Bash

Hãy bắt đầu bằng tạo một tệp mới có dạng mở rộng `.sh`. Ví dụ, ta có thể tạo một tệp có tên `devdojo.sh`.

Để tạo tệp đó, sử dụng lệnh `touch`:

```
touch devdojo.sh
```

Hoặc sử dụng trình soạn thảo văn bản:

```
nano devdojo.sh
```

Để thực thi hay chạy một tệp bash script bằng trình thông dịch shell bash, dòng đầu tiên của tệp script phải chỉ định đường dẫn tuyệt đối đến tệp thực thi bash:

```
#!/bin/bash
```

Đây còn được gọi là Shebang.

Tất cả những gì shebang làm là hướng dẫn hệ điều hành chạy script với tệp thực thi `/bin/bash`.

Tuy nhiên, bash không phải lúc nào cũng nằm trong thư mục `/bin/bash`, đặc biệt là trên các hệ thống không phải Linux hoặc do được cài đặt như một gói tùy chọn. Do đó, bạn có thể muốn sử dụng:

```
#!/usr/bin/env bash
```

Cách này sẽ tìm kiếm tệp thực thi bash trong các thư mục được liệt kê ở biến môi trường PATH.

Bash Hello World

Khi đã tạo tệp `devdojo.sh` và chỉ định shebang bash ở dòng đầu tiên, chúng ta đã sẵn sàng để tạo script bash "Hello World" đầu tiên.

Để làm điều đó, mở lại file `devdojo.sh` và thêm đoạn sau vào sau dòng `#!/bin/bash`:

```
#!/bin/bash

echo "Hello World!"
```

Lưu file và thoát.

Sau đó, dùng lệnh sau để cho script có thể thực thi:

```
chmod +x devdojo.sh
```

Tiếp theo, thực thi tệp:

```
./devdojo.sh
```

Bạn sẽ thấy thông báo "Hello World" hiện ra trên màn hình.

Một cách khác để chạy script là:

```
bash devdojo.sh
```

Vì bash có thể được sử dụng dưới dạng tương tác, bạn có thể chạy lệnh

sau trực tiếp trong terminal và sẽ nhận được kết quả tương tự:

```
echo "Hello DevDojo!"
```

Việc tạo một script trở nên hữu ích khi cần kết hợp nhiều lệnh lại với nhau.

Biến trong Bash

Giống như các ngôn ngữ lập trình khác, bạn cũng có thể sử dụng biến trong lập trình Bash. Tuy nhiên, Bash không có kiểu dữ liệu, và một biến trong Bash có thể chứa cả số lẫn ký tự.

Để gán giá trị cho một biến, bạn chỉ cần sử dụng dấu `=`:

```
name="DevDojo"
```

> **Notice:** không được có khoảng trắng trước và sau dấu `=`.

Sau đó, để truy cập biến, bạn phải sử dụng dấu `$` và tham chiếu đến nó như sau:

```
echo $name
```

Việc đặt tên biến trong dấu ngoặc nhọn là không bắt buộc, nhưng được coi là một thói quen tốt, và bạn nên dùng khi có thể:

```
echo ${name}
```

Đoạn mã trên sẽ hiển thị: `DevDojo` vì đây là giá trị của biến `name` của chúng ta.

Tiếp theo, hãy cập nhật script `devdojo.sh` của chúng ta và thêm một biến vào đó.

Một lần nữa, bạn có thể mở tệp `devdojo.sh` bằng trình soạn thảo văn bản yêu thích của mình, ở đây tôi sử dụng nano để mở tệp:

```
nano devdojo.sh
```

Thêm biến `name` vào tệp, kèm theo một thông điệp chào mừng. Tệp của chúng ta giờ trông như thế này:

```
#!/bin/bash

name="DevDojo"

echo "Hi there $name"
```

Lưu lại và chạy tệp bằng lệnh dưới đây:

```
./devdojo.sh
```

Bạn sẽ thấy kết quả sau trên màn hình:

```
Hi there DevDojo
```

Dưới đây là tóm tắt về script được viết trong tệp:

- `#!/bin/bash` - Đầu tiên, chúng ta chỉ định shebang.
- `name=DevDojo` - Sau đó, định nghĩa một biến tên là `name` và gán giá trị cho nó.
- `echo "Hi there $name"` - Cuối cùng, chúng ta xuất nội dung của biến ra màn hình dưới dạng thông điệp chào mừng bằng cách sử dụng `echo`

Bạn cũng có thể thêm nhiều biến vào tệp như sau:

```
#!/bin/bash

name="DevDojo"
greeting="Hello"

echo "$greeting $name"
```

Lưu tệp và chạy:

```
./devdojo.sh
```

Bạn sẽ thấy kết quả sau trên màn hình:

```
Hello DevDojo
```

Lưu ý rằng bạn không nhất thiết phải thêm dấu chấm phẩy `;` ở cuối mỗi dòng. Có hay không cũng được, khá giống như các ngôn ngữ lập trình khác như JavaScript!

Bạn cũng có thể thêm biến trong Command Line bên ngoài script Bash và chúng có thể được đọc như các tham số:

```
./devdojo.sh Bobby buddy!
```

Script này nhận vào hai tham số `Bobby` và `buddy!` được phân tách bằng khoảng trắng. Trong tệp `devdojo.sh`, chúng ta có:

```
#!/bin/bash

echo "Hello there" $1
```

`$1` là tham số đầu vào trước tiên (`Bobby`) trong Command Line. Tương tự, có thể có nhiều tham số đầu vào khác và tất cả đều được tham chiếu bằng dấu `$` và thứ tự tương ứng của chúng. Điều này có nghĩa là `buddy!` được tham chiếu bằng `$2`. Một cách hữu ích khác để đọc biến là `$@`, nó sẽ đọc tất cả các tham số đầu vào.

Vậy bây giờ hãy thay đổi tệp `devdojo.sh` để hiểu rõ hơn:

```
#!/bin/bash

echo "Hello there" $1

# $1 : first parameter

echo "Hello there" $2

# $2 : second parameter

echo "Hello there" $@

# $@ : all
```

Sẽ có kết quả:

```
./devdojo.sh Bobby buddy!
```

Would be the following:

```
Hello there Bobby
Hello there buddy!
Hello there Bobby buddy!
```

Nhập dữ liệu từ người dùng trong Bash

Với script trước đó, chúng ta đã định nghĩa một biến và hiển thị giá trị của biến đó trên màn hình bằng lệnh `echo $name`.

Bây giờ, hãy yêu cầu người dùng nhập dữ liệu. Để làm điều này, mở tệp bằng trình soạn thảo văn bản yêu thích của bạn và cập nhật script như sau:

```
#!/bin/bash

echo "What is your name?"
read name

echo "Hi there $name"
echo "Welcome to DevDojo!"
```

Đoạn mã trên sẽ yêu cầu người dùng nhập dữ liệu và sau đó lưu trữ dữ liệu đó dưới dạng chuỗi/văn bản trong một biến.

Sau đó, chúng ta có thể sử dụng biến này và in ra một thông điệp cho người dùng.

Kết quả của script trên sẽ như sau:

- Đầu tiên, chạy script:

```
./devdojo.sh
```

- Sau đó, bạn sẽ được yêu cầu nhập tên:

```
What is your name?
Bobby
```

- Sau khi đã nhập tên, chỉ cần nhấn Enter, và bạn sẽ nhận được kết quả như sau:

```
Hi there Bobby
Welcome to DevDojo!
```

Để làm gọn mã, ta có thể thay đổi câu lệnh `echo` đầu tiên bằng `read -p`. Lệnh `read` khi được sử dụng với tùy chọn `-p` sẽ hiển thị một thông điệp trước khi yêu cầu người dùng nhập dữ liệu:

```
#!/bin/bash

read -p "What is your name? " name

echo "Hi there $name"
echo "Welcome to DevDojo!"
```

Hãy nhớ tự mình thử nghiệm điều này nhé!

Chú thích trong Bash

Giống như các ngôn ngữ lập trình khác, bạn có thể thêm chú thích vào script của mình. Chú thích được sử dụng để ghi chú cho bản thân trong code.

Để làm như vậy trong Bash, bạn cần thêm ký hiệu # ở đầu dòng. Chú thích sẽ không bao giờ được hiển thị trên màn hình.

Đây là một ví dụ về chú thích:

```
# This is a comment and will not be rendered on the screen
```

Hãy thêm một số chú thích vào script của chúng ta:

```
#!/bin/bash

# Ask the user for their name

read -p "What is your name? " name

# Greet the user
echo "Hi there $name"
echo "Welcome to DevDojo!"
```

Chú thích là cách tốt để mô tả một số chức năng phức tạp hơn trực tiếp trong script của bạn, để người khác có thể dễ dàng tìm hiểu code của bạn.

Tham số trong Bash

Bạn có thể tham số (argument) số vào shell script khi thực thi. Để truyền tham số, bạn chỉ cần viết nó ngay sau tên script. Ví dụ:

```
./devdojo.com your_argument
```

Trong script, chúng ta có thể sử dụng `$1` để tham chiếu đến tham số đầu tiên mà ta đã chỉ định.

Nếu ta truyền đối số thứ hai, nó sẽ có sẵn dưới dạng `$2` và cứ tiếp tục như vậy.

Hãy tạo một script ngắn có tên `arguments.sh` để làm ví dụ:

```
#!/bin/bash

echo "Argument one is $1"
echo "Argument two is $2"
echo "Argument three is $3"
```

Lưu tệp và đặt quyền thực thi:

```
chmod +x arguments.sh
```

Sau đó chạy tệp và truyền **3** tham số:

```
./arguments.sh dog cat bird
```

Kết quả đầu ra bạn sẽ nhận được là:

```
Argument one is dog
Argument two is cat
Argument three is bird
```

Để tham chiếu tất cả các đối số, bạn có thể sử dụng `$@`:

```
#!/bin/bash

echo "All arguments: $@"
```

Nếu bạn chạy lại script một lần nữa:

```
./arguments.sh dog cat bird
```

Bạn sẽ nhận được kết quả đầu ra sau:

```
All arguments: dog cat bird
```

Một điều khác bạn cần ghi nhớ là `$0` được sử dụng để script tham chiếu đến chính nó.

Đây là cách tuyệt vời để tự hủy tệp nếu bạn cần hoặc chỉ để lấy tên của script.

Ví dụ, hãy tạo một script in ra tên của tệp và xóa tệp sau đó:

```
#!/bin/bash

echo "The name of the file is: $0 and it is going to be self-deleted."

rm -f $0
```

Bạn cần cẩn thận với việc tự xóa và đảm bảo rằng bạn đã sao lưu script của mình trước khi tự xóa nó.

Mảng trong Bash

Nếu bạn từng lập trình, có lẽ bạn đã quen thuộc với mảng.

Nhưng nếu bạn không phải là một lập trình viên, thì điều chính mà bạn cần biết là không giống như biến, mảng có thể chứa nhiều giá trị dưới một tên.

Bạn có thể khởi tạo một mảng bằng cách gán các giá trị được phân tách bằng dấu cách và được bao quanh bởi `()`. Ví dụ:

```
my_array=("value 1" "value 2" "value 3" "value 4")
```

Để truy cập các phần tử trong mảng, bạn cần tham chiếu bằng chỉ số số của chúng.

> **Notice:** hãy nhớ rằng bạn cần sử dụng dấu ngoặc nhọn.

- Truy cập một phần tử duy nhất, cách này sẽ hiển thị: `value 2`

```
echo ${my_array[1]}
```

- Cách này sẽ trả về phần tử cuối cùng: `value 4`

```
echo ${my_array[-1]}
```

- Như với tham số command line, dùng `@` sẽ trả về tất cả các phần

tử trong mảng, như sau: `value 1 value 2 value 3 value 4`

```
echo ${my_array[@]}
```

- Thêm dấu thăng (#) trước mảng sẽ xuất ra tổng số phần tử trong mảng, trong trường hợp của chúng ta là 4:

```
echo ${#my_array[@]}
```

Hãy đảm bảo bạn kiểm tra và thực hành điều này với các giá trị khác nhau.

Cắt mảng (Array Slicing)

Mặc dù Bash không thực sự hỗ trợ cắt mảng, bạn có thể đạt được kết quả tương tự bằng cách kết hợp chỉ index(chỉ mục) mảng và cắt chuỗi:

```
#!/bin/bash

array=("A" "B" "C" "D" "E")

# Print entire array
echo "${array[@]}" # Output: A B C D E

# Access a single element
echo "${array[1]}" # Output: B

# Print a range of elements (requires Bash 4.0+)
echo "${array[@]:1:3}" # Output: B C D

# Print from an index to the end
echo "${array[@]:3}" # Output: D E
```

Khi làm việc với mảng, luôn sử dụng `[@]` để tham chiếu đến tất cả các phần tử, và bao quanh phép mở rộng tham số bằng dấu ngoặc kép để giữ nguyên khoảng trắng trong các phần tử mảng.

Cắt chuỗi

Trong Bash, bạn có thể trích xuất các phần của chuỗi bằng cách cắt. Cú pháp cơ bản là:

```
${string:start:length}
```

Trong đó:

- `start` là chỉ số bắt đầu (bắt đầu từ 0)
- `length` là số ký tự tối đa cần trích xuất

Hãy xem một số ví dụ:

```
#!/bin/bash

text="ABCDE"

# Extract from index 0, maximum 2 characters
echo "${text:0:2}" # Output: AB

# Extract from index 3 to the end
echo "${text:3}" # Output: DE

# Extract 3 characters starting from index 1
echo "${text:1:3}" # Output: BCD

# If length exceeds remaining characters, it stops at the end
echo "${text:3:3}" # Output: DE (only 2 characters available)
```

Lưu ý rằng số thứ hai trong ký hiệu cắt đại diện cho độ dài tối đa của chuỗi con được trích xuất, không phải là chỉ số kết thúc. Điều này khác với một số ngôn ngữ lập trình khác như Python. Trong Bash, nếu bạn chỉ định một độ dài vượt quá cuối chuỗi, nó sẽ đơn giản dừng lại ở cuối chuỗi mà không gây ra lỗi.

Ví dụ:

```
text="Hello, World!"

# Extract 5 characters starting from index 7
echo "${text:7:5}" # Output: World

# Attempt to extract 10 characters starting from index 7
# (even though only 6 characters remain)
echo "${text:7:10}" # Output: World!
```

Trong ví dụ thứ hai, mặc dù chúng ta yêu cầu 10 ký tự, Bash chỉ trả về 6 ký tự có sẵn từ chỉ số 7 đến cuối chuỗi. Cách xử lý này sẽ hữu ích khi ta không chắc chắn về độ dài chính xác của chuỗi mà ta đang xử lý.

Biểu thức điều kiện trong Bash

Trong khoa học máy tính, câu điều kiện, biểu thức điều kiện và cấu trúc điều kiện là các tính năng của một ngôn ngữ lập trình, việc thực hiện các phép tính hoặc hành động khác nhau tùy thuộc vào việc điều kiện boolean do lập trình viên chỉ định mà sẽ được đánh giá là đúng hay sai.

Trong Bash, biểu thức điều kiện được sử dụng bởi lệnh phức hợp `[[` và lệnh tích hợp `[` để kiểm tra thuộc tính file và thực hiện so sánh chuỗi và số học.

Dưới đây là danh sách các biểu thức điều kiện phổ biến nhất trong Bash. Bạn không cần phải ghi nhớ chúng. Hãy tham khảo lại danh sách này bất cứ khi nào bạn cần!

Kiểm tra file

- Đúng nếu file tồn tại.

```
[[ -a ${file} ]]
```

- Đúng nếu tập tin tồn tại và là block special file.

```
[[ -b ${file} ]]
```

- Đúng nếu tập tin tồn tại và là character special file.

```
[[ -c ${file} ]]
```

- Đúng nếu tập tin tồn tại và là thư mục.

```
[[ -d ${file} ]]
```

- Đúng nếu tập tin tồn tại.

```
[[ -e ${file} ]]
```

- Đúng nếu tập tin tồn tại và là tập tin thông thường.

```
[[ -f ${file} ]]
```

- Đúng nếu tập tin tồn tại và là symbolic link.

```
[[ -h ${file} ]]
```

• Đúng nếu tập tin tồn tại và có quyền đọc.

```
[[ -r ${file} ]]
```

• Đúng nếu tập tin tồn tại và có kích thước lớn hơn không.

```
[[ -s ${file} ]]
```

• Đúng nếu tập tin tồn tại và có quyền ghi.

```
[[ -w ${file} ]]
```

• Đúng nếu tập tin tồn tại và có quyền thực thi.

```
[[ -x ${file} ]]
```

• Đúng nếu tập tin tồn tại và là symbolic link.

```
[[ -L ${file} ]]
```

Biểu thức kiểm tra chuỗi

- Đúng nếu biến shell varname đã được gán giá trị.

```
[[ -v ${varname} ]]
```

- Đúng nếu độ dài của chuỗi là không.

```
[[ -z ${string} ]]
```

- Đúng nếu độ dài của chuỗi khác không.

```
[[ -n ${string} ]]
```

- Đúng nếu hai chuỗi bằng nhau. = nên được sử dụng với lệnh test để tuân thủ theo POSIX. Khi sử dụng với lệnh [[, điều này thực hiện so khớp theo mẫu như mô tả ở trên (Compound Commands - Lệnh phức hợp).

```
[[ ${string1} == ${string2} ]]
```

- Đúng nếu hai chuỗi không bằng nhau.

```
[[ ${string1} != ${string2} ]]
```

- Đúng nếu string1 sắp xếp trước string2 theo thứ tự từ điển.

```
[[ ${string1} < ${string2} ]]
```

- Đúng nếu string1 sắp xếp sau string2 theo thứ tự từ điển.

```
[[ ${string1} > ${string2} ]]
```

Toán tử số học

- Trả về đúng nếu các số **bằng nhau**

```
[[ ${arg1} -eq ${arg2} ]]
```

- Trả về đúng nếu các số **không bằng nhau**

```
[[ ${arg1} -ne ${arg2} ]]
```

- Trả về đúng nếu arg1 **nhỏ hơn** arg2

```
[[ ${arg1} -lt ${arg2} ]]
```

- Trả về đúng nếu arg1 **nhỏ hơn hoặc bằng** arg2

```
[[ ${arg1} -le ${arg2} ]]
```

- Trả về đúng nếu arg1 **lớn hơn** arg2

```
[[ ${arg1} -gt ${arg2} ]]
```

- Trả về đúng nếu arg1 **lớn hơn hoặc bằng** arg2

```
[[ ${arg1} -ge ${arg2} ]]
```

Lưu ý, arg1 và arg2 có thể là số nguyên dương hoặc âm.

Giống như các ngôn ngữ lập trình khác, bạn có thể sử dụng điều kiện

AND & OR:

```
[[ test_case_1 ]] && [[ test_case_2 ]] # And
[[ test_case_1 ]] || [[ test_case_2 ]] # Or
```

Toán tử trạng thái kết thúc

- trả về đúng nếu lệnh đã thành công mà không có lỗi nào

```
[[ $? -eq 0 ]]
```

- trả về đúng nếu lệnh không thành công hoặc có lỗi

```
[[ $? -gt 0 ]]
```

Câu lệnh điều kiện trong Bash

Trong phần trước, chúng ta đã tìm hiểu một số biểu thức điều kiện phổ biến nhất. Bây giờ chúng ta có thể sử dụng chúng với các câu lệnh điều kiện tiêu chuẩn như `if`, `if-else` và `switch case`.

Câu lệnh If

Cấu trúc của câu lệnh `if` trong Bash như sau:

```
if [[ some_test ]]
then
    <commands>
fi
```

Dưới đây là một ví dụ nhanh yêu cầu bạn nhập tên nếu bạn để trống:

```
#!/bin/bash

# Bash if statement example

read -p "What is your name? " name

if [[ -z ${name} ]]
then
echo "Please enter your name!"
fi
```

Câu lệnh If Else

Với câu lệnh `if-else`, bạn có thể chỉ định một hành động trong trường hợp điều kiện trong câu lệnh `if` không khớp. Chúng ta có thể kết hợp điều này với các biểu thức điều kiện từ phần trước như sau:

```
#!/bin/bash

# Bash if statement example

read -p "What is your name? " name

if [[ -z ${name} ]]
then
echo "Please enter your name!"
else
echo "Hi there ${name}"
fi
```

Bạn có thể sử dụng câu lệnh if trên với tất cả các biểu thức điều kiện từ các chương trước:

```
#!/bin/bash

admin="devdojo"

read -p "Enter your username? " username

# Check if the username provided is the admin

if [[ "${username}" == "${admin}" ]] ; then
echo "You are the admin user!"
else
echo "You are NOT the admin user!"
fi
```

Dưới đây là một ví dụ khác về câu lệnh `if` sẽ kiểm tra `ID người dùng`

hiện tại và không cho phép chạy script với tư cách người dùng `root`:

```
#!/bin/bash

if (( $EUID == 0 )); then
    echo "Please do not run as root"
    exit
fi
```

Nếu bạn đặt ở đầu script, nó sẽ thoát trong trường hợp EUID là 0 và sẽ không thực thi phần còn lại của script. Điều này đã được thảo luận trên the DigitalOcean community forum.

Bạn cũng có thể kiểm tra nhiều điều kiện với câu lệnh `if`. Trong ví dụ này, chúng ta muốn đảm bảo rằng người dùng không phải là người dùng admin cũng không phải là người dùng root để đảm bảo script không có khả năng gây ra quá nhiều thiệt hại. Chúng ta sẽ sử dụng toán tử `or` trong ví dụ này, được ký hiệu bằng `||`. Điều này có nghĩa là một trong hai điều kiện cần phải đúng. Nếu chúng ta sử dụng toán tử `and` là `&&` thì cả hai điều kiện đều phải đúng.

```
#!/bin/bash

admin="devdojo"

read -p "Enter your username? " username

# Check if the username provided is the admin

if [[ "${username}" != "${admin}" ]] && [[ $EUID != 0 ]] ;
then
    echo "You are not the admin or root user, but please be
safe!"
else
    echo "You are the admin user! This could be very
destructive!"
fi
```

Nếu bạn có nhiều điều kiện và kịch bản, thì có thể sử dụng câu lệnh `elif` với các câu lệnh `if` và `else`.

```
#!/bin/bash

read -p "Enter a number: " num

if [[ $num -gt 0 ]] ; then
    echo "The number is positive"
elif [[ $num -lt 0 ]] ; then
    echo "The number is negative"
else
    echo "The number is 0"
fi
```

Câu lệnh Switch case

Giống như trong các ngôn ngữ lập trình khác, bạn có thể sử dụng câu lệnh `case` để đơn giản hóa các điều kiện phức tạp khi có nhiều lựa chọn khác nhau. Vì vậy, thay vì sử dụng một vài câu lệnh `if` và `if-else`, bạn có thể sử dụng một câu lệnh `case` duy nhất.

Cú pháp câu lệnh `case` trong Bash trông như thế này:

```
case $some_variable in

  pattern_1)
    commands
    ;;

  pattern_2| pattern_3)
    commands
    ;;

  *)
    default commands
    ;;
esac
```

Tóm tắt nhanh về cấu trúc:

- Tất cả các câu lệnh `case` bắt đầu bằng từ khóa `case`.
- Trên cùng một dòng với từ khóa `case`, bạn cần chỉ định một biến hoặc một biểu thức theo sau là từ khóa `in`.
- Sau đó, bạn có các mẫu `case`, trong đó bạn cần sử dụng `)` để xác định phần cuối của mẫu.
- Bạn có thể chỉ định nhiều mẫu được phân tách bằng dấu pipe: `|`.
- Sau mẫu, bạn chỉ định các lệnh mà bạn muốn được thực thi trong trường hợp mẫu khớp với biến hoặc biểu thức mà bạn đã chỉ định.
- Tất cả các mệnh đề phải được kết thúc bằng cách thêm `;;` ở cuối.
- Bạn có thể có một câu lệnh mặc định bằng cách thêm `*` làm mẫu.

- Để đóng câu lệnh `case`, sử dụng từ khóa `esac` (case viết ngược lại).

Dưới đây là một ví dụ về câu lệnh `case` trong Bash:

```
#!/bin/bash

read -p "Enter the name of your car brand: " car

case $car in

  Tesla)
    echo -n "${car}'s car factory is in the USA."
    ;;

  BMW | Mercedes | Audi | Porsche)
    echo -n "${car}'s car factory is in Germany."
    ;;

  Toyota | Mazda | Mitsubishi | Subaru)
    echo -n "${car}'s car factory is in Japan."
    ;;

  *)
    echo -n "${car} is an unknown car brand"
    ;;

esac
```

Với script này, chúng ta yêu cầu người dùng nhập tên của một thương hiệu xe hơi như Tesla, BMW, Mercedes, v.v.

Sau đó, với câu lệnh `case`, chúng ta kiểm tra tên thương hiệu và nếu nó khớp với bất kỳ mẫu nào của chúng ta, chúng ta sẽ in ra vị trí của nhà máy.

Nếu tên thương hiệu không khớp với bất kỳ câu lệnh `case` nào của chúng ta, chúng ta sẽ in ra một thông báo mặc định: "là một thương hiệu xe không xác định".

Kết luận

Bạn nên thử chỉnh sửa script và thử nghiệm một chút để có thể luyện tập những gì bạn vừa học trong hai chương vừa qua!

Để xem thêm ví dụ về câu lệnh `case` trong Bash, hãy xem chương 16, chúng ta sẽ tạo một menu tương tác trong Bash sử dụng câu lệnh `case` để xử lý đầu vào của người dùng.

Vòng lặp trong Bash

Tương tự như các ngôn ngữ khác, vòng lặp rất tiện lợi trong lập trình. Với Bash, bạn có thể sử dụng vòng lặp `for`, vòng lặp `while`, và vòng lặp `until`.

Vòng lặp For

Dưới đây là cấu trúc của một vòng lặp for:

```
for var in ${list}
do
    your_commands
done
```

Ví dụ:

```
#!/bin/bash

users="devdojo bobby tony"

for user in ${users}
do
    echo "${user}"
done
```

Giải thích ngắn gọn về ví dụ trên:

- Đầu tiên, chúng ta xác định một danh sách người dùng và lưu giá trị vào biến `$users`.
- Sau đó, chúng ta bắt đầu vòng lặp `for` với từ khóa `for`.
- Tiếp theo, chúng ta định nghĩa một biến mới đại diện cho từng phần tử trong danh sách. Trong trường hợp này, chúng ta định nghĩa biến `user`, đại diện cho mỗi người dùng từ biến `$users`.
- Sau đó, chúng ta sử dụng từ khóa `in` theo sau là danh sách mà chúng ta sẽ lặp qua.
- Ở dòng tiếp theo, chúng ta sử dụng từ khóa `do`, cho biết những gì chúng ta sẽ thực hiện trong mỗi lần lặp.
- Tiếp đó, chúng ta chỉ định các lệnh mà chúng ta muốn chạy.
- Cuối cùng, chúng ta kết thúc vòng lặp bằng từ khóa `done`.

Bạn cũng có thể sử dụng `for` để xử lý một chuỗi các số. Ví dụ, đây là cách lặp từ 1 đến 10:

```
#!/bin/bash

for num in {1..10}
do
    echo ${num}
done
```

Vòng lặp While

Cấu trúc của vòng lặp while khá giống với vòng lặp `for`:

```
while [ your_condition ]
do
    your_commands
done
```

Dưới đây là một ví dụ về vòng lặp `while`:

```
#!/bin/bash

counter=1
while [[ $counter -le 10 ]]
do
    echo $counter
    ((counter++))
done
```

Đầu tiên, chúng ta khai báo biến đếm và đặt giá trị là `1`, sau đó trong vòng lặp, chúng ta tăng biến đếm bằng câu lệnh: `((counter++))`. Bằng cách này, chúng ta đảm bảo rằng vòng lặp sẽ chỉ chạy 10 lần và không chạy vô hạn. Vòng lặp sẽ kết thúc ngay khi biến đếm trở thành 10, như chúng ta đã đặt trong điều kiện: `while [[ $counter -le 10 ]]`.

Hãy tạo một script yêu cầu người dùng nhập tên và không cho phép dữ liệu đầu vào trống:

```
#!/bin/bash

read -p "What is your name? " name

while [[ -z ${name} ]]
do
    echo "Your name can not be blank. Please enter a valid
name!"
    read -p "Enter your name again? " name
done

echo "Hi there ${name}"
```

Bây giờ, nếu bạn chạy script trên và chỉ nhấn Enter mà không cung cấp đầu vào, vòng lặp sẽ chạy lại và yêu cầu bạn nhập tên nhiều lần cho đến khi bạn thực sự cung cấp một đầu vào.

Vòng lặp Until

Sự khác biệt giữa vòng lặp `until` và `while` là vòng lặp `until` sẽ thực hiện các lệnh trong vòng lặp cho đến khi điều kiện trở thành đúng.

Cấu trúc:

```
until [[ your_condition ]]
do
    your_commands
done
```

Ví dụ:

```
#!/bin/bash

count=1
until [[ $count -gt 10 ]]
do
    echo $count
    ((count++))
done
```

Continue và Break

Giống như các ngôn ngữ khác, bạn cũng có thể sử dụng `continue` và `break` trong các script bash của mình:

- `continue` yêu cầu script bash dừng lần lặp hiện tại của vòng lặp và bắt đầu lần lặp tiếp theo.

Cú pháp của câu lệnh continue như sau:

```
continue [n]
```

Tham số [n] là tùy chọn và có thể lớn hơn hoặc bằng 1. Khi [n] được cung cấp, vòng lặp bao bọc thứ n sẽ được tiếp tục. continue 1 tương đương với continue.

```
#!/bin/bash

for i in 1 2 3 4 5
do
    if [[ $i -eq 2 ]]
    then
        echo "skipping number 2"
        continue
    fi
    echo "i is equal to $i"
done
```

Chúng ta cũng có thể sử dụng lệnh continue tương tự như lệnh break để kiểm soát nhiều vòng lặp.

- `break` yêu cầu script bash kết thúc vòng lặp ngay lập tức.

Cú pháp của câu lệnh break có dạng như sau:

```
break [n]
```

[n] là đối số tùy chọn và phải lớn hơn hoặc bằng 1. Khi [n] được cung cấp, vòng lặp bao bọc thứ n sẽ được thoát ra. break 1 tương đương với break.

Ví dụ:

```
#!/bin/bash

num=1
while [[ $num -lt 10 ]]
do
    if [[ $num -eq 5 ]]
    then
        break
    fi
    ((num++))
done
echo "Loop completed"
```

Chúng ta cũng có thể sử dụng lệnh break với nhiều vòng lặp. Nếu chúng ta muốn thoát khỏi vòng lặp đang làm việc hiện tại, dù là vòng lặp trong hay ngoài, chúng ta chỉ cần sử dụng break. Nhưng nếu chúng ta đang ở vòng lặp trong và muốn thoát khỏi vòng lặp ngoài, chúng ta sử dụng break 2.

Ví dụ:

```
#!/bin/bash

for (( a = 1; a < 10; a++ ))
do
    echo "outer loop: $a"
    for (( b = 1; b < 100; b++ ))
    do
        if [[ $b -gt 5 ]]
        then
            break 2
        fi
        echo "Inner loop: $b "
    done
done
```

Script bash sẽ bắt đầu với a=1 và sẽ chuyển sang vòng lặp trong. Khi đến b=5, nó sẽ thoát khỏi vòng lặp ngoài. Chúng ta có thể chỉ sử dụng break thay vì break 2 để thoát khỏi vòng lặp trong và xem cách nó ảnh hưởng đến kết quả đầu ra.

Hàm trong Bash

Hàm là một cách tuyệt vời để tái sử dụng mã. Cấu trúc của một hàm trong bash khá giống với hầu hết các ngôn ngữ khác:

```
function function_name() {
    your_commands
}
```

Bạn cũng có thể bỏ từ khóa `function` ở đầu, và hàm vẫn hoạt động:

```
function_name() {
    your_commands
}
```

Tôi thích đặt từ khóa `function` ở đó để dễ đọc hơn. Nhưng đó là vấn đề sở thích cá nhân.

Ví dụ về một hàm "Hello World!":

```
#!/bin/bash

function hello() {
    echo "Hello World Function!"
}

hello
```

> **Notice:** Một điều cần nhớ là bạn không nên thêm dấu ngoặc đơn khi gọi hàm.

Truyền tham số cho một hàm hoạt động tương tự như cách truyền tham số cho một script:

```
#!/bin/bash

function hello() {
    echo "Hello $1!"
}

hello DevDojo
```

Các hàm nên có chú thích đề cập đến mô tả, biến toàn cục, đối số, đầu ra, và giá trị trả về, nếu có

```
########################################
# Description: Hello function
# Globals:
#   None
# Arguments:
#   Single input argument
# Outputs:
#   Value of input argument
# Returns:
#   0 if successful, non-zero on error.
########################################
function hello() {
    echo "Hello $1!"
}
```

Trong một vài chương tiếp theo, chúng ta sẽ sử dụng hàm rất nhiều!

Debugging, testing và shortcuts

Để gỡ lỗi các script bash, bạn có thể sử dụng tùy chọn `-x` khi thực thi script:

```
bash -x ./your_script.sh
```

Hoặc bạn có thể thêm `set -x` trước dòng cụ thể mà bạn muốn gỡ lỗi. Lệnh `set -x` kích hoạt một chế độ của shell, trong đó tất cả các lệnh được thực thi sẽ được in ra terminal.

Một cách khác để kiểm tra script của bạn là sử dụng công cụ tuyệt vời sau đây:

https://www.shellcheck.net/

Chỉ cần sao chép và dán mã của bạn vào hộp văn bản, công cụ sẽ đưa ra một số gợi ý về cách cải thiện script của bạn.

Bạn cũng có thể chạy công cụ này trực tiếp trong terminal:

https://github.com/koalaman/shellcheck

Nếu bạn thích công cụ này, hãy nhớ gắn sao cho nó trên GitHub và đóng góp!

Là một SysAdmin/DevOps, tôi dành phần lớn thời gian làm việc trên terminal. Dưới đây là những phím tắt ưa thích giúp tôi thực hiện công việc nhanh hơn khi viết script Bash hoặc khi làm việc trên terminal.

Hai phím tắt dưới đây đặc biệt hữu ích khi bạn có một lệnh rất dài.

- Xóa mọi thứ từ con trỏ đến cuối dòng:

```
Ctrl + k
```

- Xóa mọi thứ từ con trỏ đến đầu dòng:

```
Ctrl + u
```

- Xóa một từ phía trước con trỏ:

```
Ctrl + w
```

- Tìm kiếm ngược trong lịch sử lệnh. Đây có lẽ là phím tắt mà tôi sử dụng nhiều nhất. Nó thực sự tiện lợi và tăng tốc quy trình làm việc của tôi rất nhiều:

```
Ctrl + r
```

- Xóa màn hình, tôi sử dụng phím tắt này thay vì gõ lệnh `clear`:

```
Ctrl + l
```

- Dừng hiển thị đầu ra trên màn hình:

```
Ctrl + s
```

- Cho phép hiển thị đầu ra trên màn hình trong trường hợp đã bị dừng bởi `Ctrl + s`:

```
Ctrl + q
```

• Hủy lệnh đang thực thi:

```
Ctrl + c
```

• Đưa lệnh hiện tại vào chế độ chạy nền:

```
Ctrl + z
```

Tôi sử dụng những phím tắt này thường xuyên hàng ngày, và nó tiết kiệm cho tôi rất nhiều thời gian.

Nếu bạn nghĩ rằng tôi đã bỏ sót phím tắt nào, hãy thoải mái tham gia thảo luận trên diễn đàn cộng đồng DigitalOcean!

Tạo lệnh bash tùy chỉnh

Là một lập trình viên hoặc quản trị hệ thống, bạn có thể phải dành nhiều thời gian làm việc trên terminal. Tôi luôn tìm cách tối ưu hóa các tác vụ lặp đi lặp lại.

Một cách để làm điều đó là viết các script bash ngắn hoặc tạo các lệnh tùy chỉnh, còn được gọi là alias. Ví dụ, thay vì phải gõ một lệnh dài mỗi lần sử dụng, bạn có thể tạo một phím tắt cho nó.

Ví dụ

Hãy bắt đầu với tình huống sau: là một quản trị hệ thống, bạn có thể phải kiểm tra các kết nối đến web server thường xuyên, vì vậy tôi sẽ sử dụng lệnh `netstat` làm ví dụ.

Thông thường, khi truy cập vào một server đang gặp vấn đề với kết nối ở cổng 80 hoặc 443, tôi sẽ kiểm tra xem có dịch vụ nào đang lắng nghe trên các cổng đó không và số lượng kết nối đến các cổng đó.

Lệnh `netstat` sau đây sẽ cho chúng ta biết có bao nhiêu kết nối TCP hiện tại trên cổng 80 và 443:

```
netstat -plant | grep '80\|443' | grep -v LISTEN | wc -l
```

Đây là một lệnh khá dài, việc gõ nó mỗi lần có thể tốn thời gian về lâu dài, đặc biệt là khi bạn muốn lấy thông tin nhanh chóng.

Để tránh điều đó, chúng ta có thể tạo một alias. Thay vì gõ toàn bộ lệnh, chúng ta chỉ cần gõ một lệnh ngắn. Ví dụ, giả sử chúng ta muốn có thể gõ `conn` (viết tắt của connections) và nhận được cùng thông tin. Tất cả những gì chúng ta cần làm trong trường hợp này là chạy lệnh sau:

```
alias conn="netstat -plant | grep '80\|443' | grep -v LISTEN |
wc -l"
```

Bằng cách này, chúng ta đang tạo một alias có tên `conn`, về cơ bản là một 'phím tắt' cho lệnh `netstat` dài của chúng ta. Bây giờ nếu bạn chỉ cần chạy `conn`:

```
conn
```

Bạn sẽ nhận được cùng kết quả như khi chạy lệnh `netstat` dài. Bạn có thể sáng tạo hơn và thêm một số thông báo như sau:

```
alias conn="echo 'Total connections on port 80 and 443:' ;
netstat -plant | grep '80\|443' | grep -v LISTEN | wc -l"
```

Bây giờ nếu bạn chạy `conn`, bạn sẽ nhận được kết quả sau:

```
Total connections on port 80 and 443:
12
```

Tuy nhiên, nếu bạn đăng xuất và đăng nhập lại, alias của bạn sẽ bị mất. Trong bước tiếp theo, bạn sẽ thấy cách làm cho nó tồn tại lâu dài trên hệ thống.

Lưu lại các thay đổi

Để làm cho sự thay đổi bằng script tồn tại lâu dài trên hệ thống, chúng ta cần thêm lệnh `alias` vào file profile của shell.

Mặc định trên Ubuntu, đây sẽ là file `~/.bashrc`, đối với các hệ điều hành khác, có thể là `~/.bash_profile`. Mở file bằng trình soạn thảo yêu thích của bạn:

```
nano ~/.bashrc
```

Đi đến cuối file và thêm dòng sau:

```
alias conn="echo 'Total connections on port 80 and 443:' ;
netstat -plant | grep '80\|443' | grep -v LISTEN | wc -l"
```

Lưu và thoát.

Bằng cách này, ngay cả khi bạn đăng xuất và đăng nhập lại, thay đổi của bạn sẽ được giữ lại và bạn sẽ có thể chạy lệnh bash tùy chỉnh của mình.

Liệt kê tất cả các alias có sẵn

Để liệt kê tất cả các alias có sẵn cho shell hiện tại của bạn, bạn chỉ cần chạy lệnh sau:

```
alias
```

Điều này sẽ hữu ích trong trường hợp bạn thấy một số hành vi lạ với một số lệnh.

Kết luận

Đây là một cách để tạo các lệnh bash tùy chỉnh hoặc alias bash.

Tất nhiên, bạn cũng có thể viết một script bash và thêm script đó vào thư mục `/usr/bin`, nhưng điều này sẽ không hoạt động nếu bạn không có quyền root hoặc sudo, trong khi với alias, bạn có thể làm điều đó mà không cần quyền root.

> **Notice:** Nội dung này được đăng tải lần đầu trên DevDojo.com

Viết script Bash đầu tiên của bạn

Hãy cùng tổng hợp những gì chúng ta đã học được và tạo ra script Bash đầu tiên!

Lên kế hoạch cho script

Ví dụ, chúng ta sẽ viết một script thu thập một số thông tin hữu ích về máy chủ như:

- Mức sử dụng ổ đĩa hiện tại
- Mức sử dụng CPU hiện tại
- Mức sử dụng RAM hiện tại
- Kiểm tra chính xác phiên bản Kernel

Bạn có thể tùy chỉnh script bằng cách thêm hoặc bớt chức năng để phù hợp với nhu cầu của mình.

Viết script

Việc đầu tiên bạn cần làm là tạo một file mới với phần mở rộng `.sh`. Tôi sẽ tạo một file có tên `status.sh` vì script này sẽ cho chúng ta biết trạng thái của máy chủ.

Sau khi đã tạo file, mở nó bằng trình soạn thảo văn bản yêu thích của bạn.

Như chúng ta đã học ở chương 1, ở dòng đầu tiên của script Bash, chúng ta cần chỉ định cái gọi là Shebang:

```
#!/bin/bash
```

Tất cả những gì shebang làm là hướng dẫn hệ điều hành chạy script bằng thực thi /bin/bash.

Thêm chú thích

Tiếp theo, như đã thảo luận ở chương 6, hãy bắt đầu bằng việc thêm một số chú thích để mọi người có thể dễ dàng hiểu script này dùng để làm gì. Để làm điều đó, ngay sau shebang bạn có thể thêm như sau:

```
#!/bin/bash

# Script that returns the current server status
```

Thêm biến đầu tiên của bạn

Sau đó, hãy áp dụng những gì chúng ta đã học ở chương 4 và thêm một số biến mà chúng ta có thể muốn sử dụng xuyên suốt script.

Để gán giá trị cho một biến trong bash, bạn chỉ cần sử dụng dấu `=`. Ví dụ, hãy lưu tên máy chủ của chúng ta vào một biến để có thể sử dụng lại sau này:

```
server_name=$(hostname)
```

Bằng cách sử dụng `$()`, chúng ta yêu cầu bash thực sự thực thi lệnh và sau đó gán giá trị cho biến của chúng ta.

Bây giờ nếu chúng ta in ra biến đó, chúng ta sẽ thấy tên máy chủ hiện tại:

```
echo $server_name
```

Thêm hàm đầu tiên của bạn

Như bạn đã biết sau khi đọc chương 12, để tạo một hàm trong bash, bạn cần sử dụng cấu trúc sau:

```
function function_name() {
    your_commands
}
```

Hãy tạo một hàm trả về mức sử dụng bộ nhớ hiện tại trên máy chủ của chúng ta:

```
function memory_check() {
    echo ""
        echo "The current memory usage on ${server_name} is: "
        free -h
        echo ""
}
```

Giải thích nhanh về hàm:

- `function memory_check() {` - đây là cách chúng ta định nghĩa hàm
- `echo ""` - ở đây chúng ta chỉ in một dòng mới
- `echo "The current memory usage on ${server_name} is: "` - ở đây chúng ta in một thông báo ngắn và biến `$server_name`
- `}` - cuối cùng đây là cách chúng ta kết thúc hàm

Sau khi hàm đã được định nghĩa, để gọi nó, chỉ cần sử dụng tên của hàm:

```
# Định nghĩa hàm
function memory_check() {
    echo ""
        echo "The current memory usage on ${server_name} is: "
        free -h
        echo ""
}

# Gọi hàm
memory_check
```

Thử thách: Thêm các hàm mới

Trước khi xem đáp án, hãy thử thách bản thân bạn sử dụng hàm ở trên và tự viết một vài hàm.

Các hàm nên thực hiện những việc sau:

- Mức sử dụng ổ đĩa hiện tại
- Mức sử dụng CPU hiện tại
- Mức sử dụng RAM hiện tại
- Kiểm tra phiên bản Kernel chính xác

Bạn có thể sử dụng Google nếu không chắc chắn về các lệnh cần dùng để lấy thông tin đó.

Khi bạn đã sẵn sàng, hãy cuộn xuống và kiểm tra cách chúng tôi đã làm và so sánh kết quả!

Lưu ý rằng có nhiều cách đúng để thực hiện điều này!

Script mẫu

Đây là kết quả cuối cùng sẽ trông như thế này:

```
#!/bin/bash

##
# BASH script that checks:
# - Memory usage
# - CPU load
# - Number of TCP connections
# - Kernel version
##

server_name=$(hostname)

function memory_check() {
    echo ""
        echo "Memory usage on ${server_name} is: "
        free -h
        echo ""
}

function cpu_check() {
    echo ""
        echo "CPU load on ${server_name} is: "
    echo ""
        uptime
    echo ""
}

function tcp_check() {
    echo ""
        echo "TCP connections on ${server_name}: "
    echo ""
        cat  /proc/net/tcp | wc -l
    echo ""
}

function kernel_check() {
    echo ""
        echo "Kernel version on ${server_name} is: "
        echo ""
```

```
        uname -r
    echo ""
}

function all_checks() {
        memory_check
        cpu_check
        tcp_check
        kernel_check
}

all_checks
```

Kết luận

Lập trình bash thật tuyệt vời! Cho dù bạn là kỹ sư DevOps/SysOps, lập trình viên, hay chỉ là người đam mê Linux, bạn đều có thể sử dụng các script Bash để kết hợp các lệnh Linux khác nhau và tự động hóa các tác vụ hàng ngày nhàm chán và lặp đi lặp lại, để bạn có thể tập trung vào những việc hiệu quả và thú vị hơn!

> **Notice:** Nội dung này được đăng tải lần đầu trên DevDojo.com

Tạo menu tương tác trong Bash

Trong hướng dẫn này, tôi sẽ chỉ cho bạn cách tạo một menu đa lựa chọn trong Bash để người dùng có thể chọn hành động cần thực hiện!

Chúng ta sẽ tái sử dụng một số đoạn mã từ chương trước, vì vậy hãy đảm bảo bạn đã đọc nó nếu chưa.

Lên kế hoạch cho chức năng

Hãy bắt đầu bằng việc xem xét các chức năng chính của script:

- Kiểm tra mức sử dụng Ổ cứng hiện tại
- Kiểm tra mức sử dụng CPU hiện tại
- Kiểm tra mức sử dụng RAM hiện tại
- Kiểm tra phiên bản Kernel chính xác

Trong trường hợp bạn chưa có sẵn, đây là script:

```
#!/bin/bash


##
# BASH menu script that checks:
# - Memory usage
# - CPU load
# - Number of TCP connections
# - Kernel version
##


server_name=$(hostname)

function memory_check() {
    echo ""
        echo "Memory usage on ${server_name} is: "
        free -h
        echo ""
}

function cpu_check() {
    echo ""
        echo "CPU load on ${server_name} is: "
    echo ""
        uptime
    echo ""
}

function tcp_check() {
```

```
    echo ""
        echo "TCP connections on ${server_name}: "
    echo ""
        cat  /proc/net/tcp | wc -l
    echo ""
}

function kernel_check() {
    echo ""
        echo "Kernel version on ${server_name} is: "
        echo ""
        uname -r
    echo ""
}

function all_checks() {
        memory_check
        cpu_check
        tcp_check
        kernel_check
}
```

Sau đó, chúng ta sẽ xây dựng một menu cho phép người dùng chọn hàm cần thực thi.

Tất nhiên, bạn có thể điều chỉnh các hàm hoặc thêm hàm mới tùy theo nhu cầu.

Thêm một số màu sắc

Để làm cho menu dễ đọc và dễ nắm bắt hơn, chúng ta sẽ thêm một số hàm màu.

Thêm các hàm màu sau vào đầu script của bạn:

```
##
# Color  Variables
##
green='\e[32m'
blue='\e[34m'
clear='\e[0m'

##
# Color Functions
##

ColorGreen(){
        echo -ne $green$1$clear
}
ColorBlue(){
        echo -ne $blue$1$clear
}
```

Bạn có thể sử dụng các hàm màu như sau:

```
echo -ne $(ColorBlue 'Some text here')
```

Đoạn mã trên sẽ xuất ra chuỗi `Some text here` và nó sẽ có màu xanh dương!

Thêm menu

Cuối cùng, để thêm menu, chúng ta sẽ tạo một hàm riêng biệt với cấu trúc switch case cho các tùy chọn menu:

```
menu(){
echo -ne "
My First Menu
$(ColorGreen '1)') Memory usage
$(ColorGreen '2)') CPU load
$(ColorGreen '3)') Number of TCP connections
$(ColorGreen '4)') Kernel version
$(ColorGreen '5)') Check All
$(ColorGreen '0)') Exit
$(ColorBlue 'Choose an option:') "
        read a
        case $a in
                1) memory_check ; menu ;;
                2) cpu_check ; menu ;;
                3) tcp_check ; menu ;;
                4) kernel_check ; menu ;;
                5) all_checks ; menu ;;
                        0) exit 0 ;;
                        *) echo -e $red"Wrong option."$clear;
WrongCommand;;
        esac
}
```

Giải thích nhanh về mã

Đầu tiên, chúng ta chỉ hiển thị các tùy chọn menu với một số màu sắc:

```
echo -ne "
My First Menu
$(ColorGreen '1)') Memory usage
$(ColorGreen '2)') CPU load
$(ColorGreen '3)') Number of TCP connections
$(ColorGreen '4)') Kernel version
$(ColorGreen '5)') Check All
$(ColorGreen '0)') Exit
$(ColorBlue 'Choose an option:') "
```

Sau đó, chúng ta đọc câu trả lời của người dùng và lưu trữ nó trong một biến có tên `$a`:

```
        read a
```

Cuối cùng, chúng ta có một cấu trúc switch case kích hoạt các hàm khác nhau tùy thuộc vào giá trị của `$a`:

```
        case $a in
                1) memory_check ; menu ;;
                2) cpu_check ; menu ;;
                3) tcp_check ; menu ;;
                4) kernel_check ; menu ;;
                5) all_checks ; menu ;;
                0) exit 0 ;;
                *) echo -e $red"Wrong option."$clear;
WrongCommand;;
        esac
```

Cuối cùng, chúng ta cần gọi hàm menu để thực sự in ra menu:

```
# Gọi hàm menu
menu
```

Kiểm thử script

Và bây giờ, script của chúng ta sẽ như thế này:

```
#!/bin/bash

##
# BASH menu script that checks:
#   - Memory usage
#   - CPU load
#   - Number of TCP connections
#   - Kernel version
##

server_name=$(hostname)

function memory_check() {
    echo ""
        echo "Memory usage on ${server_name} is: "
        free -h
        echo ""
}

function cpu_check() {
    echo ""
        echo "CPU load on ${server_name} is: "
    echo ""
        uptime
    echo ""
}

function tcp_check() {
    echo ""
        echo "TCP connections on ${server_name}: "
    echo ""
        cat  /proc/net/tcp | wc -l
    echo ""
}

function kernel_check() {
    echo ""
        echo "Kernel version on ${server_name} is: "
        echo ""
```

```
        uname -r
    echo ""
}

function all_checks() {
        memory_check
        cpu_check
        tcp_check
        kernel_check
}

##
# Color  Variables
##
green='\e[32m'
blue='\e[34m'
clear='\e[0m'

##
# Color Functions
##

ColorGreen(){
        echo -ne $green$1$clear
}
ColorBlue(){
        echo -ne $blue$1$clear
}

menu(){
echo -ne "
My First Menu
$(ColorGreen '1)') Memory usage
$(ColorGreen '2)') CPU load
$(ColorGreen '3)') Number of TCP connections
$(ColorGreen '4)') Kernel version
$(ColorGreen '5)') Check All
$(ColorGreen '0)') Exit
$(ColorBlue 'Choose an option:') "
        read a
        case $a in
                1) memory_check ; menu ;;
                2) cpu_check ; menu ;;
                3) tcp_check ; menu ;;
                4) kernel_check ; menu ;;
                5) all_checks ; menu ;;
```

```
                0) exit 0 ;;
                *) echo -e $red"Wrong option."$clear;
WrongCommand;;
        esac
}

# Call the menu function
menu
```

Để kiểm tra script, hãy tạo một file mới với phần mở rộng .sh, ví dụ: menu.sh và sau đó chạy nó:

```
bash menu.sh
```

Kết quả hiển thị sẽ trông như sau:

```
My First Menu
1) Memory usage
2) CPU load
3) Number of TCP connections
4) Kernel version
5) Check All
0) Exit
Choose an option:
```

Bạn sẽ có thể chọn các tùy chọn khác nhau từ danh sách và mỗi số sẽ gọi một hàm khác nhau từ script:

Kết luận

Giờ đây bạn đã biết cách tạo một menu Bash và triển khai nó trong các script của mình để người dùng có thể chọn các giá trị khác nhau!

> **Notice:** Nội dung này được đăng tải lần đầu trên DevDojo.com

Thực thi script BASH trên nhiều máy chủ từ xa

Bất kỳ lệnh nào bạn có thể chạy từ dòng lệnh đều có thể được sử dụng trong một script bash. Script được dùng để chạy một chuỗi các lệnh. Bash có sẵn mặc định trên các hệ điều hành Linux và macOS.

Hãy xem xét một tình huống giả định khi bạn cần thực thi một script BASH trên nhiều máy chủ từ xa, nhưng bạn không muốn sao chép thủ công script đó vào từng máy chủ, rồi lại phải đăng nhập vào từng máy chủ riêng lẻ và chỉ sau đó mới thực thi được script.

Tất nhiên bạn có thể sử dụng một công cụ như Ansible, nhưng hãy cùng tìm hiểu cách thực hiện việc này bằng Bash!

Điều kiện tiên quyết

Trong ví dụ này, tôi sẽ sử dụng 3 máy chủ Ubuntu từ xa được triển khai trên DigitalOcean. Nếu bạn chưa có tài khoản DigitalOcean, bạn có thể đăng ký và nhận $200 tín dụng miễn phí thông qua link giới thiệu này:

https://m.do.co/c/2a9bba940f39

Sau khi có tài khoản DigitalOcean, hãy tiến hành triển khai 3 droplet.

Tôi đã tạo sẵn 3 máy chủ Ubuntu:

Tôi sẽ đặt các địa chỉ IP của những máy chủ đó vào một file `servers.txt` mà tôi sẽ sử dụng để duyệt qua bằng script Bash của chúng ta.

Nếu bạn mới làm quen với DigitalOcean, bạn có thể làm theo các bước tạo Droplet tại đây:

- Cách tạo một Droplet từ Bảng điều khiển DigitalOcean

Bạn cũng có thể xem video hướng dẫn cách thực hiện cài đặt ban đầu cho máy chủ:

- Cách thực hiện cài đặt ban đầu cho máy chủ với Ubuntu

Hoặc tốt hơn nữa, bạn có thể làm theo bài viết này để tự động hóa quá trình cài đặt ban đầu cho máy chủ bằng Bash

- Tự động hóa cài đặt ban đầu cho máy chủ Ubuntu 18.04 bằng Bash

Với 3 máy chủ mới đã sẵn sàng, chúng ta có thể tập trung vào việc chạy script Bash trên tất cả chúng chỉ bằng một lệnh duy nhất!

Script BASH

Tôi sẽ tái sử dụng script demo từ chương trước với một vài thay đổi nhỏ. Nó chỉ đơn giản thực hiện một số kiểm tra như mức sử dụng bộ nhớ hiện tại, tải CPU hiện tại, số lượng kết nối TCP và phiên bản kernel.`

```
#!/bin/bash

##
# BASH script that checks the following:
# - Memory usage
# - CPU load
# - Number of TCP connections
# - Kernel version
##

##
# Memory check
##
server_name=$(hostname)

function memory_check() {
echo "#######"
echo "The current memory usage on ${server_name} is: "
free -h
echo "#######"
}


function cpu_check() {
echo "#######"
echo "The current CPU load on ${server_name} is: "
echo ""
uptime
echo "#######"
}

function tcp_check() {
echo "#######"
echo "Total TCP connections on ${server_name}: "
echo ""
```

```
cat /proc/net/tcp | wc -l
echo "#######"
}

function kernel_check() {
echo "#######"
echo "The exact Kernel version on ${server_name} is: "
echo ""
uname -r
echo "#######"
}

function all_checks() {
memory_check
cpu_check
tcp_check
kernel_check
}

all_checks
```

Sao chép đoạn mã trên và thêm nó vào một file có tên `remote_check.sh`. Bạn cũng có thể lấy script từ ở đây.

Chạy Script trên tất cả các máy chủ

Bây giờ chúng ta đã có script và các máy chủ sẵn sàng, đồng thời đã thêm các máy chủ đó vào file servers.txt, chúng ta có thể chạy lệnh sau để duyệt qua tất cả các máy chủ và thực thi script từ xa mà không cần sao chép script vào từng máy chủ và kết nối riêng lẻ đến từng máy chủ.

```
for server in $(cat servers.txt) ; do ssh your_user@${server}
'bash -s' < ./remote_check.sh ; done
```

Vòng lặp for này sẽ duyệt qua từng máy chủ trong file servers.txt và sau đó chạy lệnh sau cho mỗi mục trong danh sách:

```
ssh your_user@the_server_ip 'bash -s' < ./remote_check.sh
```

Bạn sẽ nhận được kết quả đầu ra như sau:

Kết luận

Đây chỉ là một ví dụ đơn giản về cách thực thi một script đơn giản trên nhiều máy chủ mà không cần sao chép script vào từng máy chủ và không cần truy cập riêng lẻ vào từng máy chủ.

Tất nhiên, bạn có thể chạy một script phức tạp hơn và trên nhiều máy chủ hơn nữa.

Nếu bạn quan tâm đến việc tự động hóa, tôi khuyên bạn nên xem trang tài nguyên Ansible trên website của DigitalOcean:

Ansible Resources

> **Notice:** Nội dung này được đăng tải lần đầu trên DevDojo

Làm việc với JSON trong BASH sử dụng jq

Công cụ dòng lệnh `jq` là một trình xử lý **JSON** nhẹ và linh hoạt cho command-line. Nó rất hữu ích để phân tích cú pháp đầu ra JSON trong BASH.

Một trong những điểm tuyệt vời của `jq` là nó được viết bằng portable C và không có phụ thuộc runtime. Tất cả những gì bạn cần làm là tải xuống một tệp nhị phân duy nhất hoặc sử dụng trình quản lý gói như apt và cài đặt nó chỉ với một lệnh duy nhất.

Lập kế hoạch cho script

Để minh họa trong hướng dẫn này, tôi sẽ sử dụng một REST API bên ngoài trả về kết quả JSON đơn giản được gọi là QuizAPI:

https://quizapi.io/

Nếu bạn muốn theo dõi, hãy đảm bảo có được một API key miễn phí tại đây:

https://quizapi.io/clientarea/settings/token

QuizAPI miễn phí cho các nhà phát triển.

Cài đặt jq

Có nhiều cách để cài đặt `jq` trên hệ thống của bạn. Một trong những cách đơn giản nhất là sử dụng trình quản lý gói tùy thuộc vào hệ điều hành của bạn.

Dưới đây là danh sách các lệnh bạn cần sử dụng tùy thuộc vào hệ điều hành của mình:

- Cài đặt jq trên Ubuntu/Debian:

```
sudo apt-get install jq
```

- Cài đặt jq trên Fedora:

```
sudo dnf install jq
```

- Cài đặt jq trên openSUSE:

```
sudo zypper install jq
```

- Cài đặt jq trên Arch:

```
sudo pacman -S jq
```

- Cài đặt trên Mac với Homebrew:

```
brew install jq
```

- Cài đặt trên Mac với MacPort:

```
port install jq
```

Nếu bạn đang sử dụng hệ điều hành khác, tôi đề nghị bạn xem tài liệu chính thức tại đây để biết thêm thông tin:

https://stedolan.github.io/jq/download/

Sau khi cài đặt jq, bạn có thể kiểm tra phiên bản hiện tại bằng cách chạy lệnh này:

```
jq --version
```

Phân tích cú pháp JSON với jq

Sau khi đã cài đặt `jq` và có API Key của QuizAPI, bạn có thể phân tích cú pháp đầu ra JSON của QuizAPI trực tiếp trong terminal.

Trước tiên, hãy tạo một biến lưu trữ API Key của bạn:

```
API_KEY=YOUR_API_KEY_HERE
```

Để lấy một số đầu ra từ một trong các endpoint của QuizAPI, bạn có thể sử dụng lệnh curl:

```
curl
"https://quizapi.io/api/v1/questions?apiKey=${API_KEY}&limit=1
0"
```

Để có đầu ra cụ thể hơn, bạn có thể sử dụng Trình tạo URL QuizAPI tại đây:

https://quizapi.io/api-config

Sau khi chạy lệnh curl, đầu ra bạn nhận được sẽ trông giống như thế này:

Điều này có thể khá khó đọc, nhưng nhờ công cụ dòng lệnh jq, tất cả những gì chúng ta cần làm là chuyển hướng lệnh curl tới jq và chúng ta sẽ thấy một đầu ra JSON được định dạng đẹp mắt:

```
curl
"https://quizapi.io/api/v1/questions?apiKey=${API_KEY}&limit=1
0" | jq
```

Lưu ý phần `| jq` ở cuối.

Trong trường hợp này, đầu ra bạn nhận được sẽ trông giống như thế này:

Bây giờ, trông nó đẹp hơn nhiều! Công cụ dòng lệnh jq đã định dạng đầu ra cho chúng ta và thêm một số màu sắc đẹp mắt!

Lấy phần tử đầu tiên với jq

Giả sử chúng ta chỉ muốn lấy phần tử đầu tiên từ đầu ra JSON, để làm điều đó chúng ta chỉ cần chỉ định chỉ mục mà chúng ta muốn xem với cú pháp sau:

```
jq .[0]
```

Bây giờ, nếu chúng ta chạy lại lệnh curl và chuyển hướng đầu ra tới jq .[0] như thế này:

```
curl
"https://quizapi.io/api/v1/questions?apiKey=${API_KEY}&limit=1
0" | jq.[0]
```

Bạn sẽ chỉ nhận được phần tử đầu tiên và đầu ra sẽ trông như thế này:

Chỉ lấy giá trị cho khóa cụ thể

Đôi khi bạn có thể chỉ muốn lấy giá trị của một khóa cụ thể, giả sử trong ví dụ của chúng ta, QuizAPI trả về danh sách các câu hỏi cùng với câu trả lời, mô tả và v.v. nhưng nếu bạn chỉ muốn lấy các Câu hỏi mà không có thông tin bổ sung thì sao?

Điều này khá đơn giản với `jq`, tất cả những gì bạn cần làm là thêm khóa sau lệnh jq, vì vậy nó sẽ trông giống như thế này:

```
jq .[].question
```

Chúng ta phải thêm `.[]` vì QuizAPI trả về một mảng và bằng cách chỉ định `.[]`, chúng ta nói với jq rằng chúng ta muốn lấy giá trị .question cho tất cả các phần tử trong mảng.

Đầu ra bạn nhận được sẽ trông giống như thế này:

Như bạn có thể thấy, bây giờ chúng ta chỉ nhận được các câu hỏi mà không có phần còn lại của các giá trị.

Sử dụng jq trong script BASH

Hãy tiếp tục và tạo một script bash ngắn để xuất ra những thông tin sau cho chúng ta:

- Chỉ lấy câu hỏi đầu tiên từ đầu ra
- Lấy tất cả các câu trả lời cho câu hỏi đó
- Gán các câu trả lời cho các biến
- In ra câu hỏi và các câu trả lời
- Để làm điều đó, tôi đã tổng hợp script sau:

> **Notice:** đảm bảo thay đổi phần API_KEY bằng khóa QuizAPI thực tế của bạn:

```
#!/bin/bash

##
# Make an API call to QuizAPI and store the output in a
variable
##
output=$(curl
'https://quizapi.io/api/v1/questions?apiKey=API_KEY&limit=10'
2>/dev/null)

##
# Get only the first question
##
output=$(echo $output | jq .[0])

##
# Get the question
##
question=$(echo $output | jq .question)

##
# Get the answers
##

answer_a=$(echo $output | jq .answers.answer_a)
answer_b=$(echo $output | jq .answers.answer_b)
answer_c=$(echo $output | jq .answers.answer_c)
answer_d=$(echo $output | jq .answers.answer_d)

##
# Output the question
##

echo "
Question: ${question}

A) ${answer_a}
B) ${answer_b}
C) ${answer_c}
D) ${answer_d}

"
```

Nếu bạn chạy script, bạn sẽ nhận được đầu ra sau:

Chúng ta thậm chí có thể làm tốt hơn bằng cách làm cho nó tương tác để chúng ta có thể thực sự chọn câu trả lời trực tiếp trong terminal của mình.

Đã có sẵn một script bash thực hiện điều này bằng cách sử dụng QuizAPI và `jq`:

Bạn có thể xem script đó tại đây:

- https://github.com/QuizApi/QuizAPI-BASH/blob/master/quiz.sh

Kết luận

Công cụ dòng lệnh `jq` là một công cụ tuyệt vời giúp bạn có khả năng làm việc với JSON trực tiếp trong terminal BASH của mình.

Bằng cách đó, bạn có thể dễ dàng tương tác với tất cả các loại REST API khác nhau bằng BASH.

Để biết thêm thông tin, bạn có thể xem tài liệu chính thức tại đây:

- https://stedolan.github.io/jq/manual/

Và để biết thêm thông tin về **QuizAPI**, bạn có thể xem tài liệu chính thức tại đây:

- https://quizapi.io/docs/1.0/overview

> **Notice:** Nội dung này được đăng tải lần đầu trên DevDojo.com

Làm việc với API Cloudflare bằng Bash

Tôi lưu trữ tất cả các trang web của mình trên các Droplet của **DigitalOcean** và sử dụng Cloudflare làm nhà cung cấp CDN. Một trong những lợi ích khi sử dụng Cloudflare là nó giúp giảm lưu lượng truy cập tổng thể đến máy chủ của bạn và cũng ẩn địa chỉ IP thực của máy chủ đằng sau CDN của họ.

Tính năng yêu thích cá nhân của tôi ở Cloudflare là bảo vệ DDoS miễn phí. Nó đã cứu các máy chủ của tôi nhiều lần khỏi các cuộc tấn công DDoS khác nhau. Họ có một API rất tiện dụng mà bạn có thể sử dụng để bật và tắt tính năng bảo vệ DDoS một cách dễ dàng.

Chương này sẽ là một bài tập thực hành! Tôi thách thức bạn viết một đoạn script bash ngắn để tự động bật và tắt tính năng bảo vệ DDoS của Cloudflare cho máy chủ của bạn khi cần thiết!

Điều kiện tiên quyết

Trước khi làm theo hướng dẫn này, hãy thiết lập tài khoản Cloudflare và chuẩn bị sẵn website của bạn. Nếu bạn không chắc cách thực hiện, bạn có thể làm theo các bước sau đây: Tạo tài khoản Cloudflare và thêm một trang web.

Sau khi có tài khoản Cloudflare, hãy đảm bảo bạn có được các thông tin sau:

- Một tài khoản Cloudflare
- Khóa API Cloudflare
- ID vùng (Zone ID) Cloudflare

Ngoài ra, hãy đảm bảo curl đã được cài đặt trên máy chủ của bạn:

```
curl --version
```

Nếu curl chưa được cài đặt, bạn cần chạy lệnh sau:

- Đối với RedHat/CentOS:

```
yum install curl
```

- Đối với Debian/Ubuntu:

```
apt-get install curl
```

Thử thách - Yêu cầu của script

Script cần giám sát mức sử dụng CPU trên máy chủ của bạn và nếu mức sử dụng CPU tăng cao dựa trên số lượng vCPU, nó sẽ tự động kích hoạt tính năng bảo vệ DDoS của Cloudflare thông qua API Cloudflare.

Các tính năng chính của script nên bao gồm:

- Kiểm tra tải CPU trên máy chủ
- Trong trường hợp CPU tăng đột biến, script sẽ kích hoạt một cuộc gọi API đến Cloudflare và bật tính năng bảo vệ DDoS cho vùng được chỉ định
- Sau khi tải CPU trở lại bình thường, script sẽ tắt tùy chọn "I'm under attack" và đặt lại về trạng thái bình thường

Script mẫu

Tôi đã chuẩn bị sẵn một script demo mà bạn có thể sử dụng làm tài liệu tham khảo. Tuy nhiên, tôi khuyến khích bạn hãy thử viết script của riêng mình trước, và chỉ xem script của tôi sau đó!

Để tải script, chỉ cần chạy lệnh sau:

```
wget
https://raw.githubusercontent.com/bobbyiliev/cloudflare-ddos-p
rotection/main/protection.sh
```

Mở script bằng trình soạn thảo văn bản yêu thích của bạn:

```
nano protection.sh
```

Và cập nhật các thông tin sau với chi tiết Cloudflare của bạn:

```
CF_CONE_ID=YOUR_CF_ZONE_ID
CF_EMAIL_ADDRESS=YOUR_CF_EMAIL_ADDRESS
CF_API_KEY=YOUR_CF_API_KEY
```

Sau đó, hãy cấp quyền thực thi cho script:

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

Cuối cùng, thiết lập 2 công việc Cron để chạy mỗi 30 giây. Để chỉnh sửa crontab của bạn, hãy chạy:

```
crontab -e
```

Và thêm nội dung sau:

```
* * * * * /path-to-the-script/cloudflare/protection.sh
* * * * * ( sleep 30 ; /path-to-the-
script/cloudflare/protection.sh )
```

Lưu ý rằng bạn cần thay đổi đường dẫn đến script với đường dẫn thực tế nơi bạn đã lưu script.

Kết luận

Đây là một giải pháp khá đơn giản và tiết kiệm, một trong những nhược điểm của script là nếu máy chủ của bạn không phản hồi do bị tấn công, script có thể không được kích hoạt.

Tất nhiên, một cách tiếp cận tốt hơn sẽ là sử dụng hệ thống giám sát như Nagios và dựa trên thống kê từ hệ thống giám sát đó, bạn có thể kích hoạt script, nhưng thử thách viết script này có thể là một bài tập tốt!

Đây là một tài liệu tham khảo tuyệt vời khác về cách sử dụng API Discord và gửi thông báo đến kênh Discord của bạn bằng script Bash:

Cách sử dụng Discord Webhooks để nhận thông báo về trạng thái trang web của bạn trên Ubuntu 18.04

> **Notice:** Nội dung này được đăng lần đầu trên DevDojo

Script BASH để tóm tắt các log truy cập của NGINX và Apache

Một trong những việc đầu tiên tôi thường làm khi thấy mức sử dụng CPU cao trên một số máy chủ Linux là kiểm tra danh sách process bằng lệnh top hoặc htop. Nếu thấy có nhiều tiến trình Apache hoặc Nginx, tôi sẽ nhanh chóng kiểm tra các access log để xác định nguyên nhân gây ra hoặc đang gây ra mức sử dụng CPU đột ngột tăng trên máy chủ, hoặc để tìm hiểu xem có hoạt động độc hại nào đang diễn ra không.

Đôi khi việc đọc log có thể khá khó khăn vì log có thể rất lớn và việc xem xét thủ công có thể mất nhiều thời gian. Hơn nữa, định dạng log thô có thể gây nhầm lẫn cho những người ít kinh nghiệm.

Giống như chương trước, chương này cũng sẽ là một thử thách! Bạn cần viết một script bash ngắn để tóm tắt toàn bộ log truy cập mà không cần cài đặt thêm bất kỳ phần mềm nào.

Yêu cầu của script

Script BASH này cần phân tích và tóm tắt các log truy cập của bạn, cung cấp những thông tin hữu ích như:

- 20 trang có nhiều yêu cầu POST nhất
- 20 trang có nhiều yêu cầu GET nhất
- 20 địa chỉ IP hàng đầu và vị trí địa lý của chúng

Script mẫu

Tôi đã chuẩn bị sẵn một script demo mà bạn có thể tham khảo. Tuy nhiên, tôi khuyến khích bạn thử viết trước, sau đó mới xem script của tôi!

Để tải script, bạn có thể clone repository bằng lệnh sau:

```
git clone
https://github.com/bobbyiliev/quick_access_logs_summary.git
```

Hoặc chạy lệnh sau để tải script về thư mục hiện tại:

```
wget
https://raw.githubusercontent.com/bobbyiliev/quick_access_logs
_summary/master/spike_check
```

Script này không thay đổi gì trên hệ thống của bạn, nó chỉ đọc nội dung của log truy cập và tóm tắt lại. Tuy nhiên, sau khi tải về, hãy đảm bảo tự kiểm tra nội dung.

Chạy script

Sau khi tải script về, bạn chỉ cần làm cho nó có quyền thực thi và chạy nó.

Để làm điều đó, hãy chạy lệnh sau để cấp quyền thực thi cho script:

```
chmod +x spike_check
```

Sau đó chạy script:

```
./spike_check /path/to/your/access_log
```

Hãy đảm bảo thay đổi đường dẫn tới file bằng đường dẫn thực tế đến log truy cập của bạn. Ví dụ, nếu bạn đang sử dụng Apache trên một máy chủ Ubuntu, lệnh chính xác sẽ như sau:

```
./spike_check /var/log/apache2/access.log
```

Nếu bạn đang sử dụng Nginx, lệnh sẽ gần như giống hệt, chỉ khác đường dẫn đến log truy cập của Nginx:

```
./spike_check /var/log/nginx/access.log
```

Tìm hiểu về kết quả đầu ra

Khi bạn chạy script, có thể mất một lúc tùy thuộc vào kích thước của log.

Kết quả đầu ra bạn sẽ thấy sẽ trông giống như thế này:

Về cơ bản, trong trường hợp này, chúng ta có thể thấy rằng chúng ta đã nhận được 16 yêu cầu POST đến file xmlrpc.php, file này thường được những kẻ tấn công sử dụng để cố gắng khai thác các trang web WordPress bằng cách dùng các tổ hợp tên người dùng và mật khẩu khác nhau.

Trong trường hợp cụ thể này, đây không phải là một cuộc tấn công brute force lớn, nhưng nó cung cấp cho chúng ta một dấu hiệu sớm và chúng ta có thể hành động để ngăn chặn một cuộc tấn công lớn hơn trong tương lai.

Chúng ta cũng có thể thấy có một vài địa chỉ IP từ Nga truy cập vào trang web của chúng ta, vì vậy trong trường hợp bạn không mong đợi bất kỳ lưu lượng truy cập nào từ Nga, bạn có thể muốn chặn những địa chỉ IP đó.

Kết luận

Đây là một ví dụ về một script BASH đơn giản cho phép bạn nhanh chóng tóm tắt các log truy cập và xác định xem có điều gì độc hại đang diễn ra hay không.

Tất nhiên, bạn cũng có thể muốn xem xét thủ công các log, nhưng đây là một thử thách tốt để thử tự động hóa việc này bằng Bash!

> **Notice:** Nội dung này được đăng lần đầu trên DevDojo

Gửi email bằng Bash và SSMTP

SSMTP là một công cụ giúp chuyển email từ máy tính hoặc máy chủ đến một máy chủ mail đã được cấu hình.

SSMTP không phải là một máy chủ email và không nhận email hay quản lý hàng đợi.

Một trong những mục đích chính của nó là chuyển tiếp email tự động (như cảnh báo hệ thống) từ máy của bạn đến một địa chỉ email bên ngoài.

Điều kiện tiên quyết

Bạn cần những thứ sau để có thể hoàn thành hướng dẫn này:

- Quyền truy cập vào máy chủ Ubuntu 18.04 với tư cách người dùng không phải root có đặc quyền sudo và tường lửa đang hoạt động. Để thiết lập những điều này, vui lòng tham khảo Hướng dẫn Thiết lập Máy chủ Ban đầu cho Ubuntu 18.04

- Một máy chủ SMTP cùng với tên người dùng và mật khẩu SMTP. Điều này cũng hoạt động với máy chủ SMTP của Gmail, hoặc bạn có thể thiết lập máy chủ SMTP riêng bằng cách làm theo các bước từ hướng dẫn Cách Cài đặt và Cấu hình Postfix làm Máy chủ SMTP Chỉ Gửi trên Ubuntu 16.04

Cài đặt SSMTP

Để cài đặt SSMTP, trước tiên bạn cần cập nhật bộ nhớ cache apt bằng lệnh:

```
sudo apt update
```

Sau đó chạy lệnh sau để cài đặt SSMTP:

```
sudo apt install ssmtp
```

Một thứ khác bạn cần cài đặt là `mailutils`, để làm điều đó hãy chạy lệnh sau:

```
sudo apt install mailutils
```

Cấu hình SSMTP

Bây giờ bạn đã cài đặt `ssmtp`, để cấu hình nó sử dụng máy chủ SMTP của bạn khi gửi email, bạn cần chỉnh sửa tệp cấu hình SSMTP.

Sử dụng trình soạn thảo văn bản ưa thích của bạn để mở tệp `/etc/ssmtp/ssmtp.conf`:

```
sudo nano /etc/ssmtp/ssmtp.conf
```

Bạn cần bao gồm cấu hình SMTP của mình:

```
root=postmaster
mailhub=<^>your_smtp_host.com<^>:587
hostname=<^>your_hostname<^>
AuthUser=<^>your_gmail_username@your_smtp_host.com<^>
AuthPass=<^>your_gmail_password<^>
FromLineOverride=YES
UseSTARTTLS=YES
```

Lưu tệp và thoát.

Gửi email bằng SSMTP

Sau khi cấu hình xong, để gửi email chỉ cần chạy lệnh sau:

```
echo "<^>Here add your email body<^>" | mail -s "<^>Here
specify your email subject<^>"
<^>your_recepient_email@yourdomain.com<^>
```

Bạn có thể chạy lệnh này trực tiếp trong terminal hoặc đưa vào script bash của bạn.

Gửi Tệp đính kèm với SSMTP (tùy chọn)

Nếu bạn cần gửi tệp đính kèm, bạn có thể sử dụng `mpack`.

Để cài đặt `mpack`, chạy lệnh sau:

```
sudo apt install mpack
```

Tiếp theo, để gửi email với tệp đính kèm, chạy lệnh sau:

```
mpack -s "<^>Your Subject here<^>" your_file.zip
<^>your_recepient_email@yourdomain.com<^>
```

Lệnh trên sẽ gửi email đến `<^>your_recepient_email@yourdomain.com<^>` với tệp `<^>your_file.zip<^>` đính kèm.

Kết luận

SSMTP là một cách tốt và đáng tin cậy để triển khai chức năng gửi email SMTP trực tiếp trong các script bash.

Để biết thêm thông tin về SSMTP, tôi khuyên bạn nên kiểm tra tài liệu chính thức tại đây.

> **Notice:** Nội dung này được đăng lần đầu trên diễn đàn cộng đồng DigitalOcean.

Tạo mật khẩu bằng Script Bash

Việc cần tạo một mật khẩu ngẫu nhiên để sử dụng cho việc cài đặt phần mềm hoặc đăng ký tài khoản trên các trang web là khá phổ biến.

Có nhiều cách để thực hiện điều này. Bạn có thể sử dụng một trình quản lý mật khẩu, nơi thường có tùy chọn tạo mật khẩu ngẫu nhiên, hoặc sử dụng một trang web có chức năng tạo mật khẩu hộ bạn.

Bạn cũng có thể sử dụng Bash trong terminal (dòng lệnh) để tạo nhanh một mật khẩu. Có nhiều cách để làm điều này và tôi sẽ giới thiệu một số phương pháp, để bạn có thể chọn cách phù hợp nhất với nhu cầu của mình.

:warning: Cảnh báo bảo mật

Script này chỉ nhằm mục đích luyện kỹ năng lập trình bash. Bạn có thể thử với các dự án đơn giản bằng BASH, nhưng bảo mật không phải là chuyện để đùa, vì vậy hãy đảm bảo không lưu mật khẩu dưới dạng văn bản thuần túy trong một tệp ở local hoặc ghi chúng ra giấy.

Tôi khuyên mọi người nên sử dụng các nhà cung cấp đáng tin cậy và an toàn để tạo và lưu trữ mật khẩu.

Tóm tắt script

Trước tiên, hãy để tôi tóm tắt nhanh những gì script của chúng ta sẽ làm:

1. Chúng ta sẽ có tùy chọn để chọn độ dài ký tự của mật khẩu khi script được thực thi.
2. Script sau đó sẽ tạo 5 mật khẩu ngẫu nhiên với độ dài đã được chỉ định ở bước 1.

Yêu cầu tiên quyết

Bạn cần một terminal bash và một trình soạn thảo văn bản. Bạn có thể sử dụng bất kỳ trình soạn thảo nào như vi, vim, nano hoặc Visual Studio Code.

Tôi chạy script này trên local trong laptop Linux của mình, nhưng nếu bạn đang sử dụng máy tính Windows, bạn có thể ssh vào bất kỳ máy chủ nào bạn chọn và thực thi script ở đó.

Mặc dù script khá đơn giản, việc có kiến thức cơ bản về lập trình BASH sẽ giúp bạn hiểu rõ hơn về script và cách nó hoạt động.

Tạo mật khẩu ngẫu nhiên

Một trong những lợi ích tuyệt vời của Linux là bạn có thể làm nhiều việc bằng các phương pháp khác nhau. Khi nói đến việc tạo một chuỗi ký tự ngẫu nhiên, cũng không ngoại lệ.

Bạn có thể sử dụng một số lệnh để tạo một chuỗi ký tự ngẫu nhiên. Tôi sẽ giới thiệu một vài cách và cung cấp một số ví dụ.

- Sử dụng lệnh `date`. Lệnh date sẽ xuất ra ngày và giờ hiện tại. Tuy nhiên, chúng ta cũng có thể thao tác thêm với đầu ra để sử dụng nó như một mật khẩu được tạo ngẫu nhiên. Chúng ta có thể mã hóa ngày bằng md5, sha hoặc chỉ cần chạy nó qua base64. Đây là một vài ví dụ:

```
date | md5sum
94cb1cdecfed0699e2d98acd9a7b8f6d -
```

sử dụng sha256sum:

```
date | sha256sum
30a0c6091e194c8c7785f0d7bb6e1eac9b76c0528f02213d1b6a5fbcc76cef
f4 -
```

sử dụng base64:

```
date | base64
0YHQsSDRj9C90YMgMzAgMTk6NTE6NDggRUVUIDIwMjEK
```

- Chúng ta cũng có thể sử dụng openssl để tạo các byte giả ngẫu nhiên và chạy đầu ra qua base64. Một ví dụ đầu ra sẽ là:

```
openssl rand -base64 10
9+soM9bt8mhdcw==
```

Hãy nhớ rằng openssl có thể chưa được cài đặt trên hệ thống của bạn, vì vậy có thể bạn sẽ cần cài đặt nó trước để sử dụng.

- Cách được ưa chuộng nhất là sử dụng bộ sinh số giả ngẫu nhiên - /dev/urandom vì nó được thiết kế cho hầu hết các mục đích mã hóa. Chúng ta cũng cần thao tác với đầu ra bằng `tr` để dịch nó. Một lệnh ví dụ là:

```
tr -cd '[:alnum:]' < /dev/urandom | fold -w10 | head -n 1
```

Với lệnh này, chúng ta lấy đầu ra từ /dev/urandom và dịch nó bằng `tr` trong khi sử dụng tất cả các chữ cái và chữ số và in ra số ký tự mong muốn.

Script

Đầu tiên, chúng ta bắt đầu script với shebang. Chúng ta sử dụng nó để cho hệ điều hành biết trình thông dịch nào sẽ được sử dụng để phân tích phần còn lại của tệp.

```
#!/bin/bash
```

Sau đó, chúng ta có thể tiếp tục và yêu cầu người dùng nhập một số thông tin. Trong trường hợp này, chúng ta muốn biết mật khẩu cần có bao nhiêu ký tự:

```
# Ask user for password length
clear
printf "\n"
read -p "How many characters you would like the password to
have? " pass_length
printf "\n"
```

Tạo mật khẩu và sau đó in ra để người dùng có thể sử dụng.

```
# This is where the magic happens!
# Generate a list of 10 strings and cut it to the desired
value provided from the user

for i in {1..10}; do (tr -cd '[:alnum:]' < /dev/urandom | fold
-w${pass_length} | head -n 1); done

# Print the strings
printf "$pass_output\n"
printf "Goodbye, ${USER}\n"
```

Script bản đầy đủ:

```
#!/bin/bash
#=======================================
# Password generator with login option
#=======================================

# Ask user for the string length
clear
printf "\n"
read -p "How many characters you would like the password to
have? " pass_length
printf "\n"

# This is where the magic happens!
# Generate a list of 10 strings and cut it to the desired
value provided from the user

for i in {1..10}; do (tr -cd '[:alnum:]' < /dev/urandom | fold
-w${pass_length} | head -n 1); done

# Print the strings
printf "$pass_output\n"
printf "Goodbye, ${USER}\n"
```

Kết luận

Đây là cách bạn có thể sử dụng script bash đơn giản để tạo mật khẩu ngẫu nhiên.

:warning: **Như đã đề cập, hãy đảm bảo sử dụng mật khẩu mạnh để bảo vệ tài khoản của bạn. Ngoài ra, khi có thể, hãy sử dụng xác thực hai yếu tố vì điều này sẽ cung cấp lớp bảo mật bổ sung cho tài khoản của bạn.**

Mặc dù script hoạt động bình thường, nhưng nó cần người dùng sẽ cung cấp đầu vào theo yêu cầu. Để hạn chế vấn đề phát sinh, bạn nên làm một số kiểm tra nâng cao hơn đối với đầu vào của người dùng để đảm bảo script sẽ tiếp tục hoạt động tốt ngay cả khi đầu vào được cung cấp không phù hợp với nhu cầu của chúng ta.

Được đóng góp bởi

Alex Georgiev

Redirections(Chuyển hướng) trong Bash

Một người dùng Linux thành thạo cần có kiến thức tốt về pipe và chuyển hướng trong Bash. Đây là một thành phần thiết yếu của hệ thống và thường rất hữu ích trong lĩnh vực Quản trị Hệ thống Linux.

Khi bạn chạy một lệnh như `ls`, `cat`, v.v., bạn sẽ nhận được một số kết quả đầu ra trên terminal. Nếu bạn viết sai lệnh, truyền sai tùy chọn hoặc tham số dòng lệnh, bạn sẽ nhận được thông báo lỗi trên terminal. Trong cả hai trường hợp, bạn đều nhận được một số văn bản. Có thể nó trông giống như "chỉ là văn bản" đối với bạn, nhưng hệ thống xử lý văn bản này khác nhau. Định danh này được gọi là File Descriptor (fd).

Trong Linux, có 3 File Descriptor: **STDIN** (0); **STDOUT** (1) và **STDERR** (2).

- **STDIN** (fd: 0): Quản lý đầu vào trong terminal.
- **STDOUT** (fd: 1): Quản lý văn bản đầu ra trong terminal.
- **STDERR** (fd: 2): Quản lý văn bản lỗi trong terminal.

Sự khác biệt giữa Pipe và Redirections

Cả *pipe* và *redirections* đều chuyển hướng luồng dữ liệu `(file descriptor)` của quá trình đang được thực thi. Sự khác biệt chính là *redirections* xử lý với luồng tệp tin`(files stream)`, gửi luồng đầu ra đến một tệp hoặc gửi nội dung của một tệp cho trước vào luồng đầu vào của quá trình.

Mặt khác, một pipe kết nối hai lệnh bằng cách gửi luồng đầu ra của lệnh thứ nhất đến luồng đầu vào của lệnh thứ hai mà không cần chỉ định bất kỳ chuyển hướng nào.

Redirection trong Bash

STDIN (Đầu vào Chuẩn)

Khi bạn nhập văn bản đầu vào mà một lệnh yêu cầu, thì thực tế bạn đang nhập văn bản vào file descriptor **STDIN**. Chạy lệnh `cat` mà không có tham số dòng lệnh nào. Có vẻ như là quá trình đã tạm dừng nhưng thực tế `cat` đang yêu cầu **STDIN**. `cat` là một chương trình đơn giản và sẽ in ra văn bản được truyền vào **STDIN**. Tuy nhiên, bạn có thể mở rộng trường hợp sử dụng bằng cách chuyển hướng đầu vào cho các lệnh nhận **STDIN**.

Ví dụ với `cat`:

```
cat << EOF
Hello World!
How are you?
EOF
```

Lệnh này in ra văn bản đã cung cấp trên màn hình terminal:

```
Hello World!
How are you?
```

Điều tương tự có thể được thực hiện với các lệnh khác nhận đầu vào qua STDIN. Ví dụ, `wc`:

```
wc -l << EOF
Hello World!
How are you?
EOF
```

Tùy chọn `-l` với `wc` đếm số dòng. Đoạn mã bash này sẽ in số dòng ra màn hình terminal:

```
2
```

STDOUT (Đầu ra chuẩn)

Văn bản bình thường không lỗi trên màn hình terminal của bạn được in ra thông qua file descriptor **STDOUT**. **STDOUT** của một lệnh có thể được chuyển hướng vào một tệp, theo cách mà đầu ra của lệnh được ghi vào một tệp thay vì được in ra màn hình terminal. Điều này được thực hiện đơn giản với sự trợ giúp của các toán tử `>` và `>>`.

Ví dụ:

```
echo "Hello World!" > file.txt
```

Lệnh trên sẽ không in "Hello World" ra màn hình terminal, thay vào đó nó sẽ tạo một tệp có tên `file.txt` và ghi chuỗi "Hello World" vào đó. Điều này có thể được kiểm chứng bằng cách chạy lệnh `cat` trên tệp `file.txt`.

```
cat file.txt
```

Tuy nhiên, mỗi lần bạn chuyển hướng **STDOUT** của bất kỳ lệnh nào nhiều lần vào cùng một tệp, nó sẽ xóa nội dung hiện có của tệp để ghi nội dung mới.

Ví dụ:

```
echo "Hello World!" > file.txt
echo "How are you?" > file.txt
```

Khi chạy `cat` trên tệp `file.txt`:

```
cat file.txt
```

Bạn sẽ chỉ nhận được chuỗi "How are you?" được in ra.

```
How are you?
```

Điều này là do chuỗi "Hello World" đã bị ghi đè. Hành vi này có thể được tránh bằng cách sử dụng toán tử `>>`.

Ví dụ trên có thể được viết lại như sau:

```
echo "Hello World!" > file.txt
echo "How are you?" >> file.txt
```

Khi chạy `cat` trên tệp `file.txt`, bạn sẽ nhận được kết quả mong muốn.

```
Hello World!
How are you?
```

Ngoài ra, toán tử chuyển hướng cho **STDOUT** cũng có thể được viết là `1>`. Ví dụ:

```
echo "Hello World!" 1> file.txt
```

STDERR (Lỗi Tiêu Chuẩn)

Văn bản lỗi hiển thị trên màn hình terminal được in ra thông qua **STDERR** của lệnh. Ví dụ:

```
ls --hello
```

sẽ cho ra thông báo lỗi. Thông báo lỗi này chính là **STDERR** của lệnh.

STDERR có thể được chuyển hướng bằng toán tử `2>`.

```
ls --hello 2> error.txt
```

Lệnh này sẽ chuyển hướng thông báo lỗi vào tệp `error.txt` và ghi nó vào đó. Điều này có thể được kiểm chứng bằng cách chạy lệnh `cat` trên tệp `error.txt`.

Bạn cũng có thể sử dụng toán tử `2>>` cho **STDERR** tương tự như cách bạn đã sử dụng `>>` cho **STDOUT**.

Đôi khi, thông báo lỗi trong các Tập lệnh Bash có thể gây phiền phức. Bạn có thể chọn bỏ qua chúng bằng cách chuyển hướng thông báo lỗi đến tệp `/dev/null`. `/dev/null` là thiết bị giả lập(pseudo-device) sẽ nhận vào văn bản và sau đó loại bỏ nó ngay lập tức.

Ví dụ trên có thể được viết như sau để hoàn toàn bỏ qua văn bản lỗi:

```
ls --hello 2> /dev/null
```

Tất nhiên, bạn có thể chuyển hướng cả **STDOUT** và **STDERR** cho cùng một lệnh hoặc tập lệnh.

```
./install_package.sh > output.txt 2> error.txt
```

Cả hai cũng có thể được chuyển hướng vào cùng một tệp.

```
./install_package.sh > file.txt 2> file.txt
```

Có một cách ngắn gọn hơn để làm điều này.

```
./install_package.sh > file.txt 2>&1
```

Piping

Cho đến nay, chúng ta đã thấy cách chuyển hướng **STDOUT**, **STDIN** và **STDERR** đến và từ một tệp. Để nối đầu ra từ *(lệnh) của một chương trình làm đầu vào cho một (lệnh)* chương trình khác, bạn có thể sử dụng ký hiệu đường ống dọc `|`.

Ví dụ:

```
ls | grep ".txt"
```

Lệnh này sẽ liệt kê các tệp trong thư mục hiện tại và chuyển đầu ra cho lệnh *grep*, sau đó lọc đầu ra để chỉ hiển thị các tệp chứa chuỗi ".txt".

Cú pháp:

```
[time [-p]] [!] command1 [ | or |& command2 ] …
```

Bạn cũng có thể xây dựng các chuỗi lệnh tùy ý bằng cách nối chúng lại với nhau thông qua ống dẫn để đạt được kết quả tốt.

Ví dụ này tạo ra danh sách mọi người dùng sở hữu tệp trong một thư mục nhất định cùng với số lượng tệp và thư mục họ sở hữu:

```
ls -l /projects/bash_scripts | tail -n +2 | sed 's/\s\s*/ /g'
| cut -d ' ' -f 3 | sort | uniq -c
```

Kết quả:

```
8 anne
34 harry
37 tina
18 ryan
```

HereDocument

Ký hiệu << có thể được sử dụng để tạo một tệp tạm thời [heredoc] và chuyển hướng từ nó tại command line.

```
COMMAND << EOF
        ContentOfDocument
        ...
        ...
EOF
```

Lưu ý rằng EOF đại diện cho dấu phân cách (kết thúc tệp) của heredoc. Thực tế, chúng ta có thể sử dụng bất kỳ từ chữ số nào thay thế để biểu thị điểm bắt đầu và kết thúc của tệp. Ví dụ, đây là một heredoc hợp lệ:

```
cat << randomword1
        This script will print these lines on the terminal.
        Note that cat can read from standard input. Using this
heredoc, we can
        create a temporary file with these lines as it's
content and pipe that
        into cat.
randomword1
```

Phần dịch cho nội dung giữa 2 chữ randomword1: Script này sẽ in những dòng này ra terminal. Lưu ý rằng cat có thể đọc từ đầu vào chuẩn. Sử dụng heredoc này, chúng ta có thể tạo một tệp tạm thời với những dòng này làm nội dung và chuyển nó vào cat.

Về cơ bản, nó sẽ xuất hiện như thể nội dung của heredoc được chuyển vào lệnh thông qua ống dẫn. Điều này có thể làm cho tập lệnh rất gọn gàng nếu nhiều dòng cần được chuyển vào một chương trình.

Hơn nữa, chúng ta có thể gắn thêm các ống dẫn như sau:

```
cat << randomword1 | wc
        This script will print these lines on the terminal.
        Note that cat can read from standard input. Using this
heredoc, we can
        create a temporary file with these lines as it's
content and pipe that
        into cat.
randomword1
```

Phần dịch cho nội dung giữa 2 chữ randomword1: Tập lệnh này sẽ in những dòng này ra terminal. Lưu ý rằng cat có thể đọc từ đầu vào tiêu chuẩn. Sử dụng heredoc này, chúng ta có thể tạo một tệp tạm thời với những dòng này làm nội dung và chuyển nó vào cat.

Ngoài ra, các biến đã định nghĩa trước có thể được sử dụng bên trong heredocs.

HereString

Herestrings khá giống với heredocs nhưng sử dụng <<<. Chúng được sử dụng cho các chuỗi một dòng cần được chuyển vào một chương trình nào đó. Cách này trông gọn gàng hơn heredocs vì chúng ta không cần phải chỉ định dấu phân cách.

```
wc <<<"this is an easy way of passing strings to the stdin of
a program (here wc)"
```

Giống như heredocs, herestrings có thể chứa các biến.

Tóm tắt

Toán tử	Mô tả
>	Lưu đầu ra vào một tệp
>>	Nối thêm đầu ra vào một tệp
<	Đọc đầu vào từ một tệp
2>	Chuyển hướng thông báo lỗi
|	Gửi đầu ra từ một chương trình làm đầu vào cho chương trình khác
<<	Chuyển nhiều dòng vào một chương trình một cách gọn gàng
<<<	Chuyển một dòng duy nhất vào một chương trình một cách gọn gàng

Cài đặt tự động WordPress trên LAMP bằng BASH

Đây là một ví dụ về cài đặt đầy đủ LAMP và WordPress hoạt động trên bất kỳ máy tính nào dựa trên Debian.

Điều kiện tiên quyết

- Một máy tính dựa trên Debian (Ubuntu, Debian, Linux Mint, v.v.)

Lên kế hoạch cho các chức năng

Hãy bắt đầu lại bằng cách xem xét các chức năng chính của script:

Cài đặt LAMP

- Cập nhật trìnhpackage manager
- Cài đặtfirewall (ufw)
- Cho phép lưu lượng truy cập SSH, HTTP và HTTPS
- Cài đặt Apache2
- Cài đặt & Cấu hình MariaDB
- Cài đặt PHP và các plugin cần thiết
- Kích hoạt tất cả các mod Apache2 cần thiết

Thiết lập Virtual Host Apache

- Tạo một thư mục trong `/var/www`
- Cấu hình quyền truy cập cho thư mục
- Tạo tập tin `$domain` trong `/etc/apache2/sites-available` và thêm nội dung Virtualhost cần thiết
- Kích hoạt trang web
- Khởi động lại Apache2

Cấu hình SSL

- Tạo chứng chỉ OpenSSL
- Thêm chứng chỉ SSL vào tập tin `ssl-params.conf`
- Thêm cấu hình SSL vào tập tin Virtualhost
- Bật SSL
- Tải lại Apache2

Cấu hình Cơ sở dữ liệu

- Tạo cơ sở dữ liệu
- Tạo người dùng
- Cập nhật quyền

Cấu hình WordPress

- Cài đặt các plugin PHP cần thiết cho WordPress
- Cài đặt WordPress
- Thêm thông tin cần thiết vào tập tin `wp-config.php`

Và không chần chừ nữa, hãy bắt đầu viết script.

Script

Chúng ta bắt đầu bằng cách thiết lập các biến và yêu cầu người dùng nhập tên miền của họ:

```
echo 'Please enter your domain of preference without www:'
read DOMAIN
echo "Please enter your Database username:"
read DBUSERNAME
echo "Please enter your Database password:"
read DBPASSWORD
echo "Please enter your Database name:"
read DBNAME

ip=`hostname -I | cut -f1 -d' '`
```

Bây giờ chúng ta đã sẵn sàng để bắt đầu viết các hàm. Bắt đầu bằng cách tạo hàm `lamp_install()`. Bên trong nó, chúng ta sẽ cập nhật hệ thống, cài đặt ufw, cho phép lưu lượng truy cập SSH, HTTP và HTTPS, cài đặt Apache2, cài đặt MariaDB và PHP. Chúng ta cũng sẽ kích hoạt tất cả các mod Apache2 cần thiết.

```
lamp_install () {
        apt update -y
        apt install ufw
        ufw enable
        ufw allow OpenSSH
        ufw allow in "WWW Full"

        apt install apache2 -y
        apt install mariadb-server
        mysql_secure_installation -y
        apt install php libapache2-mod-php php-mysql -y
        sed -i "2d" /etc/apache2/mods-enabled/dir.conf
        sed -i "2i\\\tDirectoryIndex index.php index.html
index.cgi index.pl index.xhtml index.htm" /etc/apache2/mods-
enabled/dir.conf
        systemctl reload apache2
}
```

Tiếp theo, chúng ta sẽ tạo hàm `apache_virtualhost_setup()`. Bên trong nó, chúng ta sẽ tạo một thư mục trong `/var/www`, cấu hình quyền truy cập cho thư mục, tạo tập tin `$domain` trong `/etc/apache2/sites-available` và thêm nội dung Virtualhost cần thiết, kích hoạt trang web và khởi động lại Apache2.

```
apache_virtual_host_setup () {
        mkdir /var/www/$DOMAIN
        chown -R $USER:$USER /var/www/$DOMAIN

        echo "<VirtualHost *:80>" >> /etc/apache2/sites-
available/$DOMAIN.conf
        echo -e "\tServerName $DOMAIN" >> /etc/apache2/sites-
available/$DOMAIN.conf
        echo -e "\tServerAlias www.$DOMAIN" >>
/etc/apache2/sites-available/$DOMAIN.conf
        echo -e "\tServerAdmin webmaster@localhost" >>
/etc/apache2/sites-available/$DOMAIN.conf
        echo -e "\tDocumentRoot /var/www/$DOMAIN" >>
/etc/apache2/sites-available/$DOMAIN.conf
        echo -e '\tErrorLog ${APACHE_LOG_DIR}/error.log' >>
/etc/apache2/sites-available/$DOMAIN.conf
        echo -e '\tCustomLog ${APACHE_LOG_DIR}/access.log
combined' >> /etc/apache2/sites-available/$DOMAIN.conf
        echo "</VirtualHost>" >> /etc/apache2/sites-
available/$DOMAIN.conf
        a2ensite $DOMAIN
        a2dissite 000-default
        systemctl reload apache2

}
```

Tiếp theo, chúng ta sẽ tạo hàm `ssl_config()`. Bên trong hàm, chúng ta sẽ tạo chứng chỉ OpenSSL, thêm chứng chỉ SSL vào tập tin `ssl-params.conf`, thêm cấu hình SSL vào tập tin Virtualhost, kích hoạt SSL và tải lại Apache2.

```
ssl_config () {
        openssl req -x509 -nodes -days 365 -newkey rsa:2048 -
keyout /etc/ssl/private/apache-selfsigned.key -out
/etc/ssl/certs/apache-selfsigned.crt
        echo "SSLCipherSuite
EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH" >>
/etc/apache2/conf-available/ssl-params.conf
        echo "SSLProtocol All -SSLv2 -SSLv3 -TLSv1 -TLSv1.1"
>> /etc/apache2/conf-available/ssl-params.conf
        echo "SSLHonorCipherOrder On" >> /etc/apache2/conf-
available/ssl-params.conf
```

```
        echo "Header always set X-Frame-Options DENY" >>
/etc/apache2/conf-available/ssl-params.conf
        echo "Header always set X-Content-Type-Options
nosniff" >> /etc/apache2/conf-available/ssl-params.conf
        echo "SSLCompression off" >> /etc/apache2/conf-
available/ssl-params.conf
        echo "SSLUseStapling on" >> /etc/apache2/conf-
available/ssl-params.conf
        echo "SSLStaplingCache \"shmcb:logs/stapling-
cache(150000)\"" >> /etc/apache2/conf-available/ssl-
params.conf
        echo "SSLSessionTickets Off" >> /etc/apache2/conf-
available/ssl-params.conf
        cp /etc/apache2/sites-available/default-ssl.conf
/etc/apache2/sites-available/default-ssl.conf.bak
        sed -i "s/var\/www\/html/var\/www\/$DOMAIN/1"
/etc/apache2/sites-available/default-ssl.conf
        sed -i "s/etc\/ssl\/certs\/ssl-cert-
snakeoil.pem/etc\/ssl\/certs\/apache-selfsigned.crt/1"
/etc/apache2/sites-available/default-ssl.conf
        sed -i "s/etc\/ssl\/private\/ssl-cert-
snakeoil.key/etc\/ssl\/private\/apache-selfsigned.key/1"
/etc/apache2/sites-available/default-ssl.conf
        sed -i "4i\\\t\tServerName $ip" /etc/apache2/sites-
available/default-ssl.conf
        sed -i "22i\\\tRedirect permanent \"/\"
\"https://$ip/\"" /etc/apache2/sites-available/000-
default.conf
        a2enmod ssl
        a2enmod headers
        a2ensite default-ssl
        a2enconf ssl-params
        systemctl reload apache2
}
```

Tiếp theo, chúng ta sẽ tạo hàm `db_setup()`. Bên trong nó, chúng ta sẽ tạo cơ sở dữ liệu, tạo người dùng và cấp tất cả quyền cho người dùng.

```
db_config () {
        mysql -e "CREATE DATABASE $DBNAME;"
        mysql -e "GRANT ALL ON $DBNAME.* TO
'$DBUSERNAME'@'localhost' IDENTIFIED BY '$DBPASSWORD' WITH
GRANT OPTION;"
        mysql -e "FLUSH PRIVILEGES;"
}
```

Tiếp theo, chúng ta sẽ tạo hàm `wordpress_config()`. Bên trong nó, chúng ta sẽ tải xuống phiên bản WordPress mới nhất, giải nén nó vào thư mục `/var/www/$DOMAIN`, tạo tập tin `wp-config.php` và thêm nội dung cần thiết vào nó.

```
wordpress_config () {
        db_config

        apt install php-curl php-gd php-mbstring php-xml php-
xmlrpc php-soap php-intl php-zip -y
        systemctl restart apache2
        sed -i "8i\\\t<Directory /var/www/$DOMAIN/>"
/etc/apache2/sites-available/$DOMAIN.conf
        sed -i "9i\\\t\tAllowOverride All" /etc/apache2/sites-
available/$DOMAIN.conf
        sed -i "10i\\\t</Directory>" /etc/apache2/sites-
available/$DOMAIN.conf

        a2enmod rewrite
        systemctl restart apache2

        apt install curl
        cd /tmp
        curl -O https://wordpress.org/latest.tar.gz
        tar xzvf latest.tar.gz
        touch /tmp/wordpress/.htaccess
        cp /tmp/wordpress/wp-config-sample.php
/tmp/wordpress/wp-config.php
        mkdir /tmp/wordpress/wp-content/upgrade
        cp -a /tmp/wordpress/. /var/www/$DOMAIN
        chown -R www-data:www-data /var/www/$DOMAIN
        find /var/www/$DOMAIN/ -type d -exec chmod 750 {} \;
        find /var/www/$DOMAIN/ -type f -exec chmod 640 {} \;
        curl -s https://api.wordpress.org/secret-key/1.1/salt/
>> /var/www/$DOMAIN/wp-config.php
        echo "define('FS_METHOD', 'direct');" >>
/var/www/$DOMAIN/wp-config.php
        sed -i "51,58d" /var/www/$DOMAIN/wp-config.php
        sed -i "s/database_name_here/$DBNAME/1"
/var/www/$DOMAIN/wp-config.php
        sed -i "s/username_here/$DBUSERNAME/1"
/var/www/$DOMAIN/wp-config.php
        sed -i "s/password_here/$DBPASSWORD/1"
/var/www/$DOMAIN/wp-config.php
}
```

Và cuối cùng, chúng ta sẽ tạo hàm `execute()`. Bên trong nó, chúng ta sẽ gọi tất cả các hàm mà chúng ta đã tạo ở trên.

```
execute () {
        lamp_install
        apache_virtual_host_setup
        ssl_config
        wordpress_config
}
```

Và bây giờ, bạn đã có script và sẵn sàng để chạy nó. Và nếu bạn cần toàn bộ script, bạn có thể tìm thấy nó trong phần tiếp theo.

Script đầy đủ

```
#!/bin/bash

echo 'Please enter your domain of preference without www:'
read DOMAIN
echo "Please enter your Database username:"
read DBUSERNAME
echo "Please enter your Database password:"
read DBPASSWORD
echo "Please enter your Database name:"
read DBNAME

ip=`hostname -I | cut -f1 -d' '`

lamp_install () {
        apt update -y
        apt install ufw
        ufw enable
        ufw allow OpenSSH
        ufw allow in "WWW Full"

        apt install apache2 -y
        apt install mariadb-server
        mysql_secure_installation -y
        apt install php libapache2-mod-php php-mysql -y
        sed -i "2d" /etc/apache2/mods-enabled/dir.conf
        sed -i "2i\\\tDirectoryIndex index.php index.html
index.cgi index.pl index.xhtml index.htm" /etc/apache2/mods-
enabled/dir.conf
        systemctl reload apache2
}

apache_virtual_host_setup () {
        mkdir /var/www/$DOMAIN
        chown -R $USER:$USER /var/www/$DOMAIN

        echo "<VirtualHost *:80>" >> /etc/apache2/sites-
available/$DOMAIN.conf
        echo -e "\tServerName $DOMAIN" >> /etc/apache2/sites-
```

```
available/$DOMAIN.conf
        echo -e "\tServerAlias www.$DOMAIN" >>
/etc/apache2/sites-available/$DOMAIN.conf
        echo -e "\tServerAdmin webmaster@localhost" >>
/etc/apache2/sites-available/$DOMAIN.conf
        echo -e "\tDocumentRoot /var/www/$DOMAIN" >>
/etc/apache2/sites-available/$DOMAIN.conf
        echo -e '\tErrorLog ${APACHE_LOG_DIR}/error.log' >>
/etc/apache2/sites-available/$DOMAIN.conf
        echo -e '\tCustomLog ${APACHE_LOG_DIR}/access.log
combined' >> /etc/apache2/sites-available/$DOMAIN.conf
        echo "</VirtualHost>" >> /etc/apache2/sites-
available/$DOMAIN.conf
        a2ensite $DOMAIN
        a2dissite 000-default
        systemctl reload apache2

}


ssl_config () {
        openssl req -x509 -nodes -days 365 -newkey rsa:2048 -
keyout /etc/ssl/private/apache-selfsigned.key -out
/etc/ssl/certs/apache-selfsigned.crt
        echo "SSLCipherSuite
EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH" >>
/etc/apache2/conf-available/ssl-params.conf
        echo "SSLProtocol All -SSLv2 -SSLv3 -TLSv1 -TLSv1.1"
>> /etc/apache2/conf-available/ssl-params.conf
        echo "SSLHonorCipherOrder On" >> /etc/apache2/conf-
available/ssl-params.conf
        echo "Header always set X-Frame-Options DENY" >>
/etc/apache2/conf-available/ssl-params.conf
        echo "Header always set X-Content-Type-Options
nosniff" >> /etc/apache2/conf-available/ssl-params.conf
        echo "SSLCompression off" >> /etc/apache2/conf-
available/ssl-params.conf
        echo "SSLUseStapling on" >> /etc/apache2/conf-
available/ssl-params.conf
        echo "SSLStaplingCache \"shmcb:logs/stapling-
cache(150000)\"" >> /etc/apache2/conf-available/ssl-
params.conf
        echo "SSLSessionTickets Off" >> /etc/apache2/conf-
available/ssl-params.conf
        cp /etc/apache2/sites-available/default-ssl.conf
/etc/apache2/sites-available/default-ssl.conf.bak
```

```
        sed -i "s/var\/www\/html/var\/www\/$DOMAIN/1"
/etc/apache2/sites-available/default-ssl.conf
        sed -i "s/etc\/ssl\/certs\/ssl-cert-
snakeoil.pem/etc\/ssl\/certs\/apache-selfsigned.crt/1"
/etc/apache2/sites-available/default-ssl.conf
        sed -i "s/etc\/ssl\/private\/ssl-cert-
snakeoil.key/etc\/ssl\/private\/apache-selfsigned.key/1"
/etc/apache2/sites-available/default-ssl.conf
        sed -i "4i\\\t\tServerName $ip" /etc/apache2/sites-
available/default-ssl.conf
        sed -i "22i\\\tRedirect permanent \"/\"
\"https://$ip/\"" /etc/apache2/sites-available/000-
default.conf
        a2enmod ssl
        a2enmod headers
        a2ensite default-ssl
        a2enconf ssl-params
        systemctl reload apache2
}
db_config () {
        mysql -e "CREATE DATABASE $DBNAME;"
        mysql -e "GRANT ALL ON $DBNAME.* TO
'$DBUSERNAME'@'localhost' IDENTIFIED BY '$DBPASSWORD' WITH
GRANT OPTION;"
        mysql -e "FLUSH PRIVILEGES;"
}

wordpress_config () {
        db_config

        apt install php-curl php-gd php-mbstring php-xml php-
xmlrpc php-soap php-intl php-zip -y
        systemctl restart apache2
        sed -i "8i\\\t<Directory /var/www/$DOMAIN/>"
/etc/apache2/sites-available/$DOMAIN.conf
        sed -i "9i\\\t\tAllowOverride All" /etc/apache2/sites-
available/$DOMAIN.conf
        sed -i "10i\\\t</Directory>" /etc/apache2/sites-
available/$DOMAIN.conf

        a2enmod rewrite
        systemctl restart apache2

        apt install curl
        cd /tmp
        curl -O https://wordpress.org/latest.tar.gz
```

```
        tar xzvf latest.tar.gz
        touch /tmp/wordpress/.htaccess
        cp /tmp/wordpress/wp-config-sample.php
/tmp/wordpress/wp-config.php
        mkdir /tmp/wordpress/wp-content/upgrade
        cp -a /tmp/wordpress/. /var/www/$DOMAIN
        chown -R www-data:www-data /var/www/$DOMAIN
        find /var/www/$DOMAIN/ -type d -exec chmod 750 {} \;
        find /var/www/$DOMAIN/ -type f -exec chmod 640 {} \;
        curl -s https://api.wordpress.org/secret-key/1.1/salt/
>> /var/www/$DOMAIN/wp-config.php
        echo "define('FS_METHOD', 'direct');" >>
/var/www/$DOMAIN/wp-config.php
        sed -i "51,58d" /var/www/$DOMAIN/wp-config.php
        sed -i "s/database_name_here/$DBNAME/1"
/var/www/$DOMAIN/wp-config.php
        sed -i "s/username_here/$DBUSERNAME/1"
/var/www/$DOMAIN/wp-config.php
        sed -i "s/password_here/$DBPASSWORD/1"
/var/www/$DOMAIN/wp-config.php
}

execute () {
        lamp_install
        apache_virtual_host_setup
        ssl_config
        wordpress_config
}
```

Tóm tắt

Script thực hiện các công việc sau:

- Cài đặt LAMP
- Tạo một virtual host
- Cấu hình SSL
- Cài đặt WordPress
- Cấu hình WordPress

Với những điều đã nói, tôi hy vọng bạn thích ví dụ này. Nếu bạn có bất kỳ câu hỏi nào, vui lòng liên hệ trực tiếp với tôi tại @denctl.

Lời kết

Chúc mừng! Bạn vừa hoàn thành hướng dẫn cơ bản về Bash!

Nếu thấy hữu ích, hãy nhớ gắn sao cho dự án trên GitHub!

Nếu bạn có bất kỳ đề xuất cải tiến nào, đừng ngần ngại đóng góp pull request hoặc mở issue.

Trong cuốn sách nhập môn Bash này, chúng ta mới chỉ đi qua những kiến thức cơ bản, nhưng bạn đã có đủ hành trang để bắt đầu viết script và tự động hóa các tác vụ hàng ngày!

Bước tiếp theo, hãy thử viết script của riêng bạn và chia sẻ nó với cộng đồng! Đây là cách tốt nhất để học bất kỳ ngôn ngữ lập trình hay scripting mới nào!

Nếu cuốn sách này truyền cảm hứng cho bạn viết được những script Bash hay, đừng quên tweet về nó và gắn thẻ @bobbyiliev_ để chúng tôi có thể xem nhé!

Một lần nữa, xin chúc mừng bạn đã hoàn thành cuốn sách này!