

DevOps Modulo 5 - IT Talent

Escrito por: [Gabriel Oliveira dos Santos](#)

Github: <https://github.com/Hypothesis>

Shell Scripting

Shell é uma linguagem que é usado para a manipulação e uso de um computador, podendo ser Mac, Linux e ate Windows. Muito usado para chamada de sistemas, manipulação de pacotes e entre outras funções.

Scripts Shell



- Automatizar tarefas em um sistema UNIX
 - Aumenta a produtividade
 - Reduz o risco de erros
- Portabilidade
 - UNIX: Linux, macOS
 - Windows (emuladores)
- Linguagem de Programação
 - Condicionais, loops, variáveis, funções...

Em vez de fazer tarefas manualmente, voce pode usar os Scripts para automatizar esses processos.

Vantagens x Desvantagens

Operações no Sistema de Arquivo

Gerenciamento de Processos

Trabalhar com texto

Administração de Sistemas

Cálculos complexos

Dados complexos

Interface gráfica

Eles não têm muitas vantagens para cálculos e dados complexos, e aplicações que exijam uma interface gráfica, pois são executados pelos terminais. Porém a fácil implementação, a automação e o gerenciamento de processos fazem com que o Shell Scripting seja uma opção mais que essencial para o DevOps.

Interpretadores

bash

Mais popular

Roda em
praticamente
qualquer lugar

Uma boa escolha

zsh

Mais recursos

Personalizável com
plugins

Algumas diferenças
para o bash

sh

O shell UNIX padrão

O mais portátil

Menos recursos que
os outros shell

Interpretadores são como tipos de arquivos que são "armazenados" os scripts de comando usando Shell, tipos de SO podem ter tipos mais populares para seu sistema padrão.

Introdução a Variaveis

```
$ curso = "Linux"
curso: command not found
$ |
```

Quando queremos criar uma variavel em um comando Shell, nao se pode usar espaço entre a declaração, pois como no exemplo acima, o shell entende que "curso" é um comando, não uma variavel. Na imagem abaixo, é reconhecido como variavel, e nao um comando como antes.

```
$ curso = "Linux"
curso: command not found
$ curso="Linux"
$ |
```

E pra exibir o valor da nossa variavel, temos que usar "\$" juntamente com o echo, assim é exibido o valor da variavel.

```
$ echo $curso
Linux
$ |
```

Criando Scripts para Automação de Processos

Nano

```

GNU nano 6.2                                script.sh
ashfsadjf
sADF OSIAHJDFOIJ
ADF ASODJFKÇALKDSJFÇALKSDJF
SadfLkjascdfklj
|

[ Wrote 5 lines ]

^G Help      ^O Write Out  ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File  ^\ Replace   ^U Paste     ^J Justify   ^_ Go To Line

```

com o comando "nano" é possível ter um editor para a criação de scripting, assim é possível nomear o arquivo com Ctrl+O. Fácil uso, ideal para iniciantes.

Vim

```
moises@moite-moises-irt: /tmp %  
  
VIM - Vi IMproved  
  
version 8.2.1847  
by Bram Moolenaar et al.  
Modified by team+vim@tracker.debian.org  
Vim is open source and freely distributable  
  
Become a registered Vim user!  
type :help register<Enter> for information  
  
type :q<Enter> to exit  
type :help<Enter> or <F1> for on-line help  
type :help version8<Enter> for version info  
  
0,0-1 All
```

O editor Vim, já é um pouco mais avançado. Ele tem modos para o uso da sua aplicação, para entrar no modo de inserção, usasse Esc+i, para salvar nosso arquivo aperta Esc e digite ":w" mas é necessario colocar o nome do arquivo com Esc + ":saveas +nom do nosso arquivo", e para sair usasse Esc ":q".

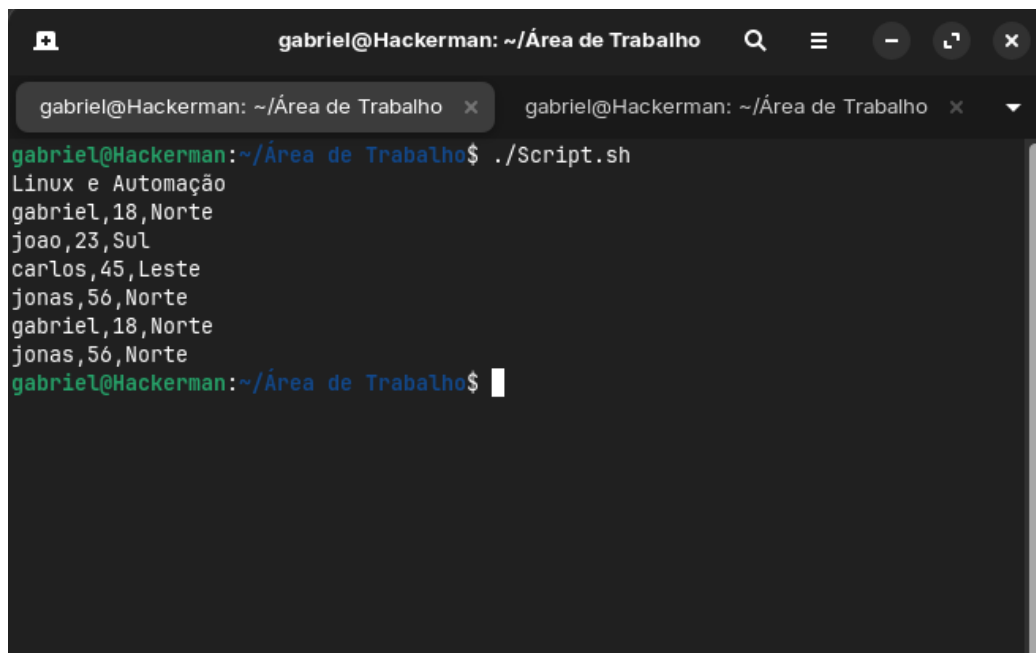
Comandos



```
gabriel@Hackerman: ~/Área de Trabalho
gabriel@Hackerman:~/Área de Trabalho$ head ah.txt
gabriel,18,Norte
joao,23,Sul
carlos,45,Leste
jonas,56,Norte
gabriel@Hackerman:~/Área de Trabalho$ grep Norte ah.txt
gabriel,18,Norte
jonas,56,Norte
gabriel@Hackerman:~/Área de Trabalho$
```

grep, faz uma busca inteligente de um padrao dentro de um arquivo. grep [padrao] [nome do nosso arquivo]. Na imagem acima o head ler tudo o que esta dentro do nosso arquivo txt, e o grep seleciona tudo que esta relacionado ao nosso padrao que queremos, que foi o "Norte".

Uma forma de fazer toda essa automação de forma mais rapida e simples, podemos fazer um Script.sh e executar tudo isso em apenas um comando. Assim vóde muda a permissao do arquivo para ser executado com "chmod u+x nome_do_arquivo.sh", e o u é para que esse arquivo so seja acessado por o nosso usuario.



```
gabriel@Hackerman: ~/Área de Trabalho
gabriel@Hackerman: ~/Área de Trabalho$ ./Script.sh
Linux e Automação
gabriel,18,Norte
joao,23,Sul
carlos,45,Leste
jonas,56,Norte
gabriel,18,Norte
jonas,56,Norte
gabriel@Hackerman:~/Área de Trabalho$
```

Hashbang



```
gabriel@Hackerman: ~/Área de Trabalho/shell
GNU nano 6.2 script2.sh *
#!/bin/bash
```

Hashbang é a primeira linha em que determina qual interpretador deve ser usado para interpretar nossa script. No exemplo assim, dizemos que o bash é o interpretador padrão para o nosso script



```
gabriel@Hackerman: ~/Área de Trabalho/shell
GNU nano 6.2 script2.sh
#!/bin/bash
echo "Script com interpretador bash"
```

e ao executar essa script, vamos exibir um texto na tela , mas o interpretador dessa scripting é o bash.

```
gabriel@Hackerman: ~/Área de Trabalho/shell
gabriel@Hackerman:~/Área de Trabalho/shell$ chmod u+x script2.sh
gabriel@Hackerman:~/Área de Trabalho/shell$ ./script2.sh
Script com interpretador bash
gabriel@Hackerman:~/Área de Trabalho/shell$
```

podemos usar `/bin/bash` ou até `/bin/sh`, para saber se o interpretador é `bash` ou `sh`, podemos usar um método. Arrays não funcionam no `sh`, mas funcionam no `bash`, que dá erro.

```
GNU nano 6.2 script2.sh
#!/bin/sh

array=(1 2 3)
echo "Script com interpretador bash"
echo ${array[0]}
```

```
gabriel@Hackerman:~/Área de Trabalho/shell$ ./script2.sh
./script2.sh: 3: Syntax error: "(" unexpected
gabriel@Hackerman:~/Área de Trabalho/shell$
```

se usarmos no `/bin/bash`, nosso código funciona.

```
GNU nano 6.2 script2.sh *
#!/bin/bash

array=(1 2 3)
echo "Script com interpretador bash"
echo ${array[0]}
```



```
gabriel@Hackerman: ~/Área de Trabalho/shell
gabriel@Hackerman:~/Área de Trabalho/shell$ ./script2.sh
./script2.sh: 3: Syntax error: "(" unexpected
gabriel@Hackerman:~/Área de Trabalho/shell$ nano script2.sh
gabriel@Hackerman:~/Área de Trabalho/shell$ ./script2.sh
Script com interpretador bash
1
gabriel@Hackerman:~/Área de Trabalho/shell$
```

Permissões

```
moises@localhost ShellScripting$ ls
script.sh vendas.csv
moises@localhost ShellScripting$ ls -l
total 8
-rw-r--r--. 1 moises moises 24 abr 1 20:45 script.sh
-rw-r--r--. 1 moises moises 311 abr 1 20:42 vendas.csv
moises@localhost ShellScripting$
```

Permissões de arquivos e diretórios são essenciais para a segurança, e é básico esse sistema de segurança do Linux. Quando começa com um "-", é um arquivo (um diretório começa com dr), em que -rw (read and write) para o root, o -r (only read) para o grupo e os outros apenas read. Essa é a nomenclatura e forma de se ler o (-rw -r - -r - -), é possível ver que nosso usuário não pode executar nosso script, então usamos o comando "chmod u+x script.sh" (o + adiciona uma permissão e - remove uma permissão), assim temos a permissão para executar para o nosso root.

Condicional e Loop no Shell

-Condicional

```
$ if mkdir teste; then echo "ok"; else echo "erro"; fi
```

Queremos criar uma pasta com nome teste, se verdadeiro mostre ok, senão mostre erro. A sintaxe começa com if e termina com fi, em que cada

condicional é separado por ;

```
$ if mkdir teste; then echo "ok"; else echo "erro"; fi
ok
$ ls
teste
$
```

Assim mostrando que a pasta foi exibida com sucesso, e se rodarmos o código novamente dará erro, pois já existe uma pasta com esse nome, exibindo "erro" na tela e erro de mkdir.

```
$ if mkdir teste; then echo "ok"; else echo "erro"; fi
mkdir: cannot create directory 'teste': File exists
erro
$
```

```
GNU nano 6.2 script.sh
#!/usr/bin/env bash

echo "O seu primeiro argumento foi $1"

if [[ ! $1 ]]; then
    echo "Erro: não foi encontrado o argumento"
    exit 1
fi

[ Read 8 lines ]
^G Help      ^O Write Out ^W Where Is  ^K Cut
^X Exit      ^R Read File ^\ Replace   ^I Paste
```

```
GNU nano 6.2 script.sh *
#!/usr/bin/env bash

if [[ $1 == "X" ]]
then
    echo "Voce digitou X. Não gostamos de X"
    exit 1
fi

^G Help      ^O Write Out ^W Where Is  ^K Cut
^X Exit      ^R Read File ^\ Replace   ^I Paste
```

```
$ ./script.sh 4
$ ./script.sh y
$ ./script.sh x
$ ./script.sh X
Voce digitou X. Não gostamos de X
$ |
```

programa para testar se uma variavel foi passada para nosso script, passando o argumento X de comparação. Com numero nao usamos "=", usamos -eq.

```
GNU nano 6.2 script.sh *
#!/usr/bin/env bash

if [[ $1 -eq 5 ]]
then
    echo "Voce digitou $1. Não gostamos de $1"
    exit 1
fi

^G Help      ^O Write Out ^W Where Is  ^K Cut
^X Exit      ^R Read File ^\ Replace   ^I Paste
```

```
$ ./script.sh 4
$ ./script.sh y
$ ./script.sh x
$ ./script.sh X
Voce digitou X. Não gostamos de X
$ nano script.sh
$ ./script.sh X
$ ./script.sh 4
$ ./script.sh 5
Voce digitou 5. Não gostamos de 5
$
```

-Loop

```
GNU nano 6.2 loop.sh *

for nome in João Maria Fátima
do
    echo "O nome é $nome"
done

^G Help      ^O Write Out  ^W Where Is
^X Exit      ^R Read File  ^\ Replace

$ ./loop.sh
O nome é João
O nome é Maria
O nome é Fátima
$ |
```

Docker



Docker

Eliminar atrito nos ambiente de desenvolvimento e automatizar processos do nosso projeto.O Docker permite que possamos implementar aplicativos dentro de containers virtuais, assim é possível rodar aplicativos específicos de um SO em outro tipo de Linux, Windows e até Mac-OS.

O que é o Docker

Um gerenciador de contêineres

- virtualização leve (*sistemas host e convidado compartilham o mesmo kernel*)
- baseado em namespaces e cgroups do Linux

Virtualizar a nível de sistema operacional trazendo um ambiente de testes mais próximos do cliente. Um container Docker é um pacote de software com todas as dependências necessárias para executar um aplicativo específico. Todas as configurações e instruções para iniciar ou parar containers são ditadas pela imagem do Docker. Sempre que um usuário executa uma imagem, um novo container é criado.

O que é o Docker

Eficiência e economia de recursos

- imagens imutáveis
- implantação instantânea
- adequado para micro-serviços (*um processo, um contêiner*)

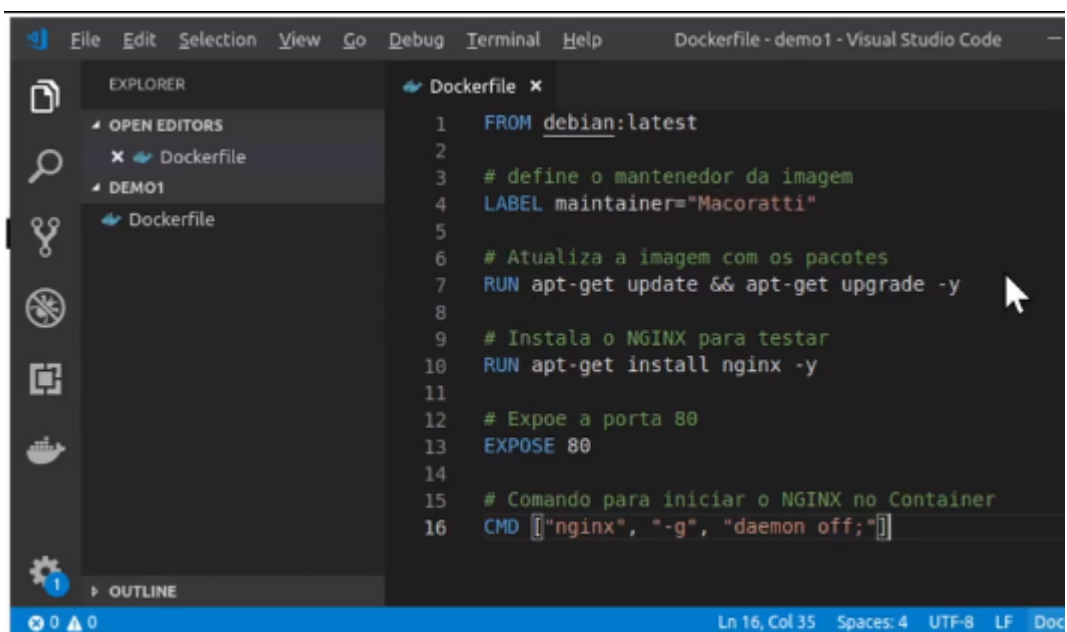
Inicialização de dockers de forma mais rápida, imagens de virtualização que não impedem o outro de executar, sem erros de dependencias de virtualiação para um com outro, além de ser adequado para micro-serviços.

O que é o Docker

Um sistema de build

- Imagens podem ser construídas a partir de código-fonte
- Usando uma DSL (Domain Specific Language) simples

DSL é uma linguagem própria para um framework, assim tem sua linguagem própria, como na figura abaixo.



```
1 FROM debian:latest
2
3 # define o mantenedor da imagem
4 LABEL maintainer="Macoratti"
5
6 # Atualiza a imagem com os pacotes
7 RUN apt-get update && apt-get upgrade -y
8
9 # Instala o NGINX para testar
10 RUN apt-get install nginx -y
11
12 # Expoe a porta 80
13 EXPOSE 80
14
15 # Comando para iniciar o NGINX no Container
16 CMD ["nginx", "-g", "daemon off;"]
```

O que é o Docker

Um conjunto de APIs REST

API do Engine (controla o Docker Engine)

API de Plugin (extende o engine → redes, armazenamento, autorização)

API de Registry (publicar/baixar imagens)

API do Swarm (gerencia um cluster de hosts Docker)

Padronização

Como o Docker Ajuda?

Padronização:

Ambientes reproduzíveis

Facilidade de configuração

Integração Contínua e Implantação Contínua

Colaboração

Escalabilidade

Ambientes Reproduzíveis: Com o docker é compactar todas as dependências de uma aplicação em um container, garantindo que a nossa aplicação seja executada do mesmo jeito em qualquer ambiente (SO).

Facilidade de Configuração: O Docker fornece uma maneira padronizada para definir e configurar ambientes de testes, desenvolvimento (etc) com arquivo de

configuração, como docker file e docker compose

CI/CD: A padronização do Docker, com ambientes consistentes e definidos por código, é mais fácil automatizar e padronizar nosso processo de construção, testes e desenvolvimentos.

Colaboração: Docker ajuda na na colaboração para com os desenvolvedores, pois é possível enivar containeres que são configurados, assim rodando na Máquina de cada desenvolvedor da mesma forma.

Escalabilidade: Simplifica a escalabilidade horizontal, assim adicionar novas instancias é mais fácil e simplificado.

Arquivamento

Como o Docker Ajuda?

Arquivamento:

- Código-fonte - Dockerfile – receita para reconstruir o ambiente do zero
- Binário – Imagem Docker – “Foto” imutável do software com o seu ambiente de execução – pode ser re-executada em qualquer momento posterior

Dockerfile é como uma receita de bolo, em que é dito como o container deve ser configurado para construir uma imagem, instalação de dependencias, escolha de imagens, variaveis de ambientes são configurações descritas no dockerfile. Assim padronizando o ambiente de execução do nosso ambinete.

Imagem Docker é uma “foto” de uma ambiente descrito no dockerfile, assim não depende nosso ambiente construído. Assim nossa imagem docker pode

ser executado em qualquer docker posteriormente, sem nenhuma barreira de dependência.

Imagem Docker

Uma **imagem** Docker é:

uma **“foto” imutável do sistema de arquivos**

Criado a partir de um dockerfile, essa imagem é imutável, assim a imagem não pode ser alterada. Assim facilidade de implantação e segurança, já que não pode ser construídos de novo (a partir dessa imagem).

Container Docker

Um **container** Docker é:

Um sistema de arquivos temporário:

- Em camadas sobre um s.a. imutável (imagem Docker)

Uma pilha de rede (como o OSI):

- Com seu próprio endereço privado (por padrão em 172.17.x.x)

Um grupo de processos:

- Um processo principal iniciado dentro do container
- Todos os sub-processos recebem um SIGKILL qdo o processo principal termina

Por ter essa camada de sistema de arquivos, esse container do Docker, é leve e consome pouca memória da nossa máquina. O que garante os isolamentos

dos containeres é o fato dos arquivos temporarios, que são executados na nossa imagem, são excluídos quando nosso container é removido do nosso Docker. Redes virtuais são criados nos containeres para que essas imagens (containeres) possam se conectar através da rede, assim cada container tem seu próprio endereçamento privado.