

# Programação Shell Script

## Automatizando Rotinas no Linux

<http://www.udemy.com/programacao-shell-script>

# O Instrutor – Ricardo Prudenciato

- Mais de 15 anos de experiência em Linux
- Atuação em médias e grandes empresas
- Experiência como Administrador de Sistemas e Analista de Suporte Linux

Ricardo Prudenciato



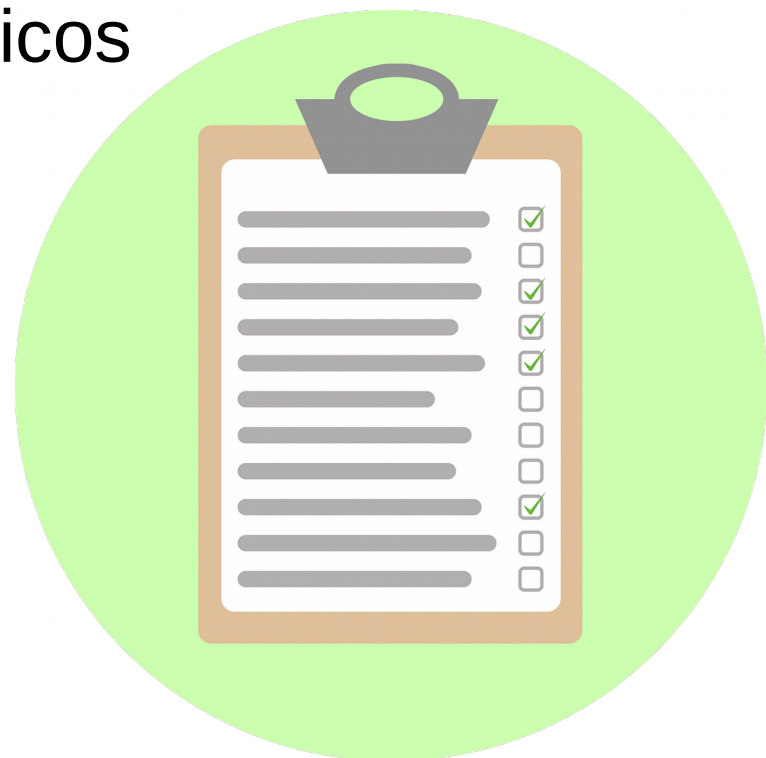
# O que você vai Aprender

- Criar um Shell Script do Zero
- Ser um Criador de Soluções
- Modificar, Melhorar e Corrigir Scripts
- Tornar-se um Profissional Linux Diferenciado



# Seções do Curso

- Revisão do Shell e Comandos Básicos
- Os Primeiros Passos
- Instruções Condicionais
- Instruções de Loop
- Uso de Funções
- Criando Logs e Enviando E-mails
- Encontrando Erros no Código



# Para quem é o curso?

- Profissionais que atuam em ambientes Linux ou Unix
- Administradores de Sistema
- Analistas de Suporte
- Profissionais e Estudantes que desejam aprimorar seus conhecimentos em Linux



# O que preciso para fazer o curso?

- Recomendável conhecimento de comandos básicos de Linux
- Um ambiente Linux/Unix para realizar os exercícios.

Obrigado e nos vemos no Curso!

**THANK YOU**

ありがとうございました **MERCI**

**DANKE** धन्यवाद

شُكراً **OBRIGADO**

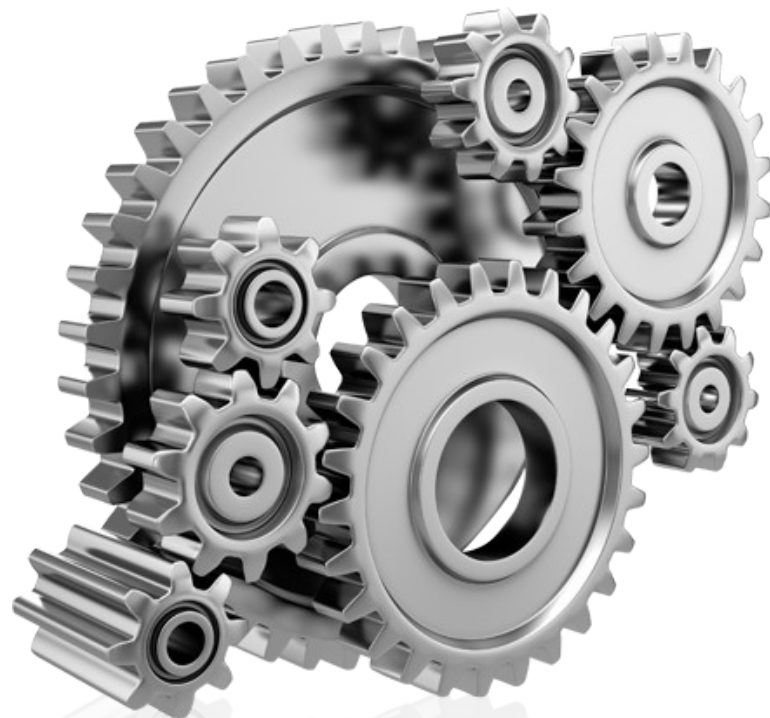
# O que é um Shell Script

- Uma sequência lógica de comandos e instruções no Shell
- Além de comandos são utilizadas variáveis, instruções lógicas e condicionais



# Funções do Shell Script

- Automatizar e Agilizar Atividades
- Ferramentas de Apoio ao Administrador ou Analista



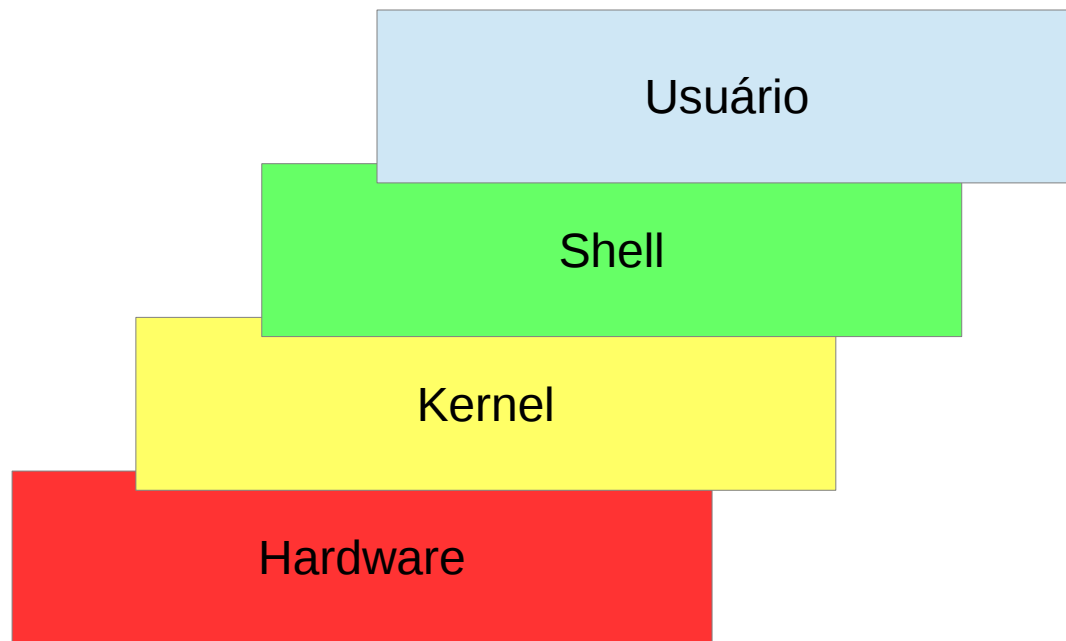
# Importância para o Profissional

- Aumento de Produtividade
- Criador de Soluções
- Reconhecimento
- Legado



# Shell

- Interface entre o usuário e o sistema Unix/Linux
- Interpretador de Instruções



# Tipos de Shell

- Bourne Shell (sh): Shell original e padrão do Unix
- Bourne-Again Shell (**bash**): Shell padrão no Linux  
Compatível com sh mas com melhorias. Também  
Incorpora recursos do csh e ksh
- Korn Shell (ksh): Também uma melhoria do bourne shell
- C Shell (csh): Traz para o shell recursos da linguagem C

# Instruções Condicionais

Instruções utilizadas para a execução condicional de comandos

# Instruções Condicionais - If

```
if <comando-condição>  
then  
    comando1  
    comando2  
    comando3  
fi
```

# Instruções Condicionais - If

```
if <comando-condição>  
then  
    comando1  
else  
    comando2  
fi
```

# Instruções Condicionais - If

```
if <comando-condição>  
then  
    comando1  
elif <comando-condição>  
then  
    comando2  
else  
    comando3  
fi
```



# Instruções Condicionais - If

Comando test: Utilizado para criar uma condição

```
test <expressão>
```

# Instruções Condicionais - If

Testando valores numéricos:

Opção	Descrição
-eq	Igual (equal)
-ne	Diferente (not equal)
-gt	Maior que (greater than)
-ge	Maior ou igual que (greater equal)
-lt	Menor que (lower than)
-le	Menor ou igual que (lower equal)

# Instruções Condicionais - If

Testando Strings:

Opção	Descrição
=	Uma string igual a outra
!=	Uma string diferente da outra
-n	String não nula
-z	String nula

# Instruções Condicionais - If

Testando Arquivos:

Opção	Descrição
-f	É um arquivo
-d	É um diretório
-r	Tem permissão de leitura
-w	Tem permissão de escrita
-x	Tem permissão de execução
-s	Possui tamanho maior que zero

# Instruções Condicionais - If

Exemplos:

```
# test 50 -gt 100
```

```
# test "$VAR1" -eq 12
```

```
# test -f /tmp/teste
```

```
# test "$VAR1" = "$VAR2"
```

# Instruções Condicionais - If

Exemplos:

```
[ 50 -gt 100 ]
```

```
[ "$VAR1" -eq 12 ]
```

```
[ -f /tmp/teste ]
```

```
[ "$VAR1" = "$VAR2" ]
```

# Instruções Condicionais - If

Exemplos:

```
VAR1=12
```

```
if test "$VAR1" -gt 10
```

```
then
```

```
    echo sucesso
```

```
fi
```

# Instruções Condicionais - If

Exemplos:

```
VAR1=12
```

```
if [ "$VAR1" -gt 10 ]
```

```
then
```

```
    echo sucesso
```

```
fi
```



# Instruções Condicionais - If

Exemplos:

```
VAR1=12
```

```
if [ ! "$VAR1" -gt 10 ]      # Negação
```

```
then
```

```
    echo sucesso
```

```
fi
```

# Instruções Condicionais - If

Exemplos:

```
VAR1=12
```

```
if [ "$VAR1" -gt 10 -a "$VAR1" -lt 20 ] # AND
```

```
then
```

```
    echo sucesso
```

```
fi
```

# Instruções Condicionais - If

Exemplos:

```
VAR1=12
```

```
if [ "$VAR1" -gt 10 -o "$VAR1" -eq 5 ] # OR
```

```
then
```

```
    echo sucesso
```

```
fi
```

# Instruções Condicionais - case

```
case $valor in
    padrão1)
        comandos
        ;;
    padrão2)
        comandos
        ;;
    *)
        comandos
        ;;
esac
```

# Instruções Condicionais - case

```
case $opcao in
    1)
        echo "Opção Incluir"
        ;;
    2)
        echo "Opção Remover"
        ;;
    *)
        echo "Opção Inexistente"
        ;;
esac
```

# Instruções Condicionais - case

```
case $character in
    [0-9])
        echo "O caracter informado é um número"
        ;;
    [A-Z])
        echo "O caracter informado é uma letra maiúscula"
        ;;
    [a-z])
        echo "O caracter informado é uma letra minúscula"
        ;;
esac
```

# Instruções de Loop

Instruções utilizadas para a execução de uma série de comandos em ciclos

# Instruções de Loop - for

```
for variavel in valor1 valor2 ... valorN
do
    comando1
    comando2
    ...
done
```



# Instruções de Loop - for

```
for numero in 1 2 3 4 5  
do  
    echo "O número atual é $numero"  
done
```

# Instruções de Loop - for

```
for arquivo in alunos*  
do  
    echo "O arquivo atual é $arquivo"  
done
```

# Instruções de Loop - for

```
for sequencia in $(seq 5 10)
do
    echo "O número é $sequencia"
done
```

# Instruções de Loop - for

```
for sequencia in {5..10}  
do  
    echo "O número é $sequencia"  
done
```

# Instruções de Loop - for

```
for sequencia in $(seq 1 5 50)
do
    echo "O número é $sequencia"
done
```

# Instruções de Loop - for

```
for sequencia in {1..50..5}
do
    echo "O número é $sequencia"
done
```

# Instruções de Loop - for

```
for i in $(cat arquivo.txt)
do
    echo "A valor atual é $i"
done
```

# Instruções de Loop - for

```
for (( i=5 ; i <= 20 ; i++ ))  
do  
    echo "O número é $i"  
done
```



# Instruções de Loop - while

```
while <comando-condição>  
do  
    comando1  
    comando2  
    ...  
done
```

# Instruções de Loop - while

```
x=1  
while [ $x -le 20 ]  
do  
    echo O valor atual é $x  
    x=$((expr $x + 1))  
done
```

# Instruções de Loop - while

```
while [ $(ps axu | wc -l) -lt 300 ]  
do  
    echo "Tudo OK"  
    sleep 30  
done
```

# Instruções de Loop - while

```
while ls /var/lock/processo.lock > /dev/null  
do  
    echo "Processo em Execução"  
    sleep 30  
done
```

# Instruções de Loop - until

```
until <comando-condição>  
do  
    comando1  
    comando2  
    ...  
done
```

# Instruções de Loop - until

```
x=1
```

```
until [ $x -eq 20 ]
```

```
do
```

```
    echo O valor atual é $x
```

```
    x=$(expr $x + 1)
```

```
done
```

# Instruções de Loop - until

```
until [ $(ps axu | wc -l) -ge 300 ]  
do  
    echo "Tudo OK"  
    sleep 30  
done
```

# Instruções de Loop - until

```
until ls /var/lock/processo.lock > /dev/null  
do  
    echo "Aguardando Processo..."  
    sleep 30  
done
```



# Instruções de Loop - break

`break` - Utilizado para sair do loop

# Instruções de Loop - break

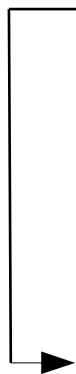
```
while ls /var/lock/processo.lock > /dev/null  
do
```

```
    if [ $(date +%H) -gt 18 ]  
    then
```

```
        break
```

```
    fi  
    echo "Processo em Execução"  
    sleep 30
```

```
done
```

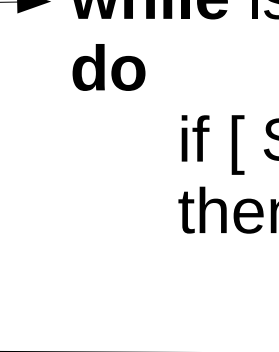


# Instruções de Loop - continue

`continue` - Utilizado para voltar ao início do loop

# Instruções de Loop - continue

```
► while ls /var/lock/processo.lock > /dev/null
do
    if [ $(date +%H) -eq 18 ]
    then
        sleep 3600
        continue
    fi
    echo "Processo em Execução"
    sleep 30
done
```



# Functions

**Função:** Trecho de código que exerce uma rotina específica dentro do script.

# Functions – Utilidade

- Evita a repetição excessiva de código
- Reduz o tamanho final do script
- Facilita a Manutenção

# Functions – Características

- Podem utilizar parâmetros
- Podem utilizar variáveis globais ou locais
- Devem ser definidas antes de serem chamadas
- Podem ser utilizados códigos de retorno

# Functions – Sintaxe Definição

```
function nome-funcao () {  
    Comandos  
}
```

```
nome-funcao () {  
    Comandos  
}
```



# Functions – Chamando funções

nome-funcao

nome-funcao par1 par2

VAR1=\$(nome-funcao)

# Functions – Variáveis

- Global = Visível por todo o código (padrão)
- Local = Visível apenas na função

```
local VAR1="Shell Script"
```

# Functions – Return Code

- Mesmo princípio do Exit Code
- Definida pela instrução “return”
- Acessada por \$?

```
return 10  
echo $?
```

# Parabéns!!!

"Aprender é a coisa mais inteligente que se pode fazer."

Miguel Esteves Cardoso



# Tudo o que Aprendemos...

- Como criar e executar shell scripts;
- Formas de receber entradas do usuário;
- Como declarar e utilizar variáveis;
- Instruções condicionais e de loop;
- Como criar logs e mandar e-mails;
- Como encontrar erros no código;
- E muito mais...

# O que fizemos...

- Geração de Relatórios;
- Automatização de rotinas de backup;
- Monitoração e Alarme de Recursos do Sistema
  - Processos
  - CPU
  - Memória (Swap)
  - Espaço em Disco

# Isso é Tudo?



# Avaliação





Muito Obrigado!!!

**THANK YOU**

ありがとうございました **MERCI**

**DANKE** धन्यवाद

شُكراً **OBRIGADO**