Minicurso: Automatização de Tarefas com Bash e Cron Jobs

Apresentação

Instrutora: M.Sc. Ana Luiza Martins Karl

Este minicurso tem como objetivo introduzir aos alunos a automatizar tarefas usando o Bash (shell do Linux) e o agendador de tarefas `cron`, criando *scripts* reutilizáveis e eficientes. Os alunos aprenderão a manipular o terminal, escrever *scripts*, utilizar estruturas de controle e agendar tarefas automáticas em sistemas baseados em Unix.

1. Introdução ao Linux e ao Terminal

Um pouco de história

- Linux criado em 1991 por **Linus Torvalds**.
- Inspirado no UNIX.
- Código aberto, gratuito e mantido por uma grande comunidade.
- Também deu origem ao Git, sistema de controle de versão.

Por que aprender Linux em 2025?

- Presente em servidores, supercomputadores, nuvem, IoT, Android.
- Requisito essencial em carreiras de infraestrutura, dados, DevOps, segurança e desenvolvimento.
- Base para ferramentas modernas como Docker, Kubernetes e sistemas de CI/CD.
- Suporte robusto da comunidade, estabilidade, flexibilidade e segurança.

Distribuições Linux

- Variadas: Ubuntu, Debian, Fedora, Arch, CentOS, entre outras.

- Todas compartilham o **kernel** Linux, mas diferem em pacotes, filosofia e usabilidade.
- Escolha da distribuição depende do perfil do usuário (servidor, desktop, minimalista, etc).

---

2. Fundamentos do Terminal e Bash

O que é o terminal?

- Interface de linha de comando (CLI).
- Usado para interagir com o sistema por meio de texto.
- Ideal para tarefas rápidas, repetitivas ou administrativas.

Por que usar o terminal?

- Mais leve e eficiente que interfaces gráficas.
- Permite automação e maior controle.
- Essencial para administração de servidores, scripts, deploys e ambientes de desenvolvimento.
- Combinável com outras ferramentas por meio de pipes (`|`) e redirecionamentos (`>`, `>>`).

Estrutura de Diretórios Linux:

```
/     - Raiz do sistema
/home  - Diretório dos usuários
/bin   - Comandos essenciais
/etc   - Arquivos de configuração
/tmp   - Arquivos temporários
/var   - Logs e arquivos variáveis
/usr   - Aplicações e binários de usuário
```

Comandos básicos:

```
pwd       # Mostra o caminho do diretório atual
ls        # Lista arquivos e pastas
```

```
cd        # Navega entre diretórios
touch     # Cria arquivos vazios
mkdir     # Cria diretórios
rm        # Remove arquivos
mv        # Move ou renomeia arquivos
tail      # Mostra as últimas linhas de um arquivo
head      # Mostra as primeiras linhas
echo      # Imprime texto na tela
cat       # Exibe conteúdo de arquivos
chmod     # Altera permissões de acesso
```

`sudo`

- Executa comandos com privilégios de administrador (root).
- Necessário para tarefas como instalação de pacotes, edição de arquivos do sistema, iniciar/parar serviços.

```
sudo apt update             # Atualiza repositórios
sudo systemctl restart ssh  # Reinicia o serviço SSH
```

---

3. Lógica de Scripts com Bash

O que é um script?

- Um script é um arquivo de texto com comandos que serão executados na ordem em que aparecem.
- Ele automatiza tarefas que seriam feitas manualmente no terminal.
- Deve começar com a linha `#!/bin/bash` (chamada de *shebang*).

```
#!/bin/bash
echo "Fazendo backup..."
cp -r ~/Documentos ~/Backup
echo "Backup concluído!"
```

Variáveis

- Guardam informações reutilizáveis no script.

```
nome="Ana"
echo "Olá, $nome"
data=$(date)
echo "Hoje é: $data"
```

- Atribuição **sem espaços**.
- Sempre acessar com o símbolo `$`.

Entrada e saída do usuário

- Permite interatividade no script.

```
echo "Digite seu nome:"
read nome
echo "Olá, $nome!"
```

Estruturas de controle de fluxo: `if`

Permite decisões baseadas em condições:

```
if [ $idade -ge 18 ]; then
  echo "Maior de idade"
else
  echo "Menor de idade"
fi
```

Estruturas de controle de fluxo: Operadores

Numéricos: `-eq` (igual), `-ne` (diferente), `-lt`, `-le`, `-gt`, `-ge`

Strings: `=`, `!=`, `-z` (vazio), `-n` (não vazio)

Arquivos: `-e` (existe), `-f` (arquivo), `-d` (diretório), `-s` (não vazio)

Estruturas de controle de fluxo: `for`

Itera sobre listas:

```
for i in 1 2 3; do
  echo "Item: $i"
done
```

Com arquivos:

```
for arq in *.txt; do
  echo "Processando $arq"
done
```

Estruturas de controle de fluxo: `while`

Executa enquanto uma condição for verdadeira:

```
contador=1
while [ $contador -le 5 ]; do
  echo "Contador: $contador"
  ((contador++))
done
```

Parâmetros na execução

Permite passar argumentos ao script:

```
#!/bin/bash
echo "Olá, $1! Bem-vindo ao curso de $2."
```

Execução:

```
bash boas_vindas.sh Ana Bash
```

---

4. Agendamento com `cron`

O que é o cron?

- Sistema de agendamento de tarefas recorrentes no Linux.
- Executa comandos ou scripts em horários predefinidos.

Sintaxe do `crontab`

```
* * * * * comando
| | | | |
| | | | └── dia da semana (0–6)
| | | └──── mês (1–12)
| | └────── dia do mês (1–31)
| └──────── hora (0–23)
└────────── minuto (0–59)
```

Exemplo de agendamento

```
0 8 * * * /home/aluno/scripts/backup.sh
```

> Executa o script todos os dias às 08h da manhã.

Para editar seu crontab:

```
crontab -e
```

4: Prática

Exercício 1: Crie um script `boasvindas.sh` que:

- Peça nome e idade;
- Informe se é maior de idade;
- Conte até a idade com `for`.

Exercício 2: Crie um script `limpar_logs.sh` que:

- Apague arquivos `.log` com mais de 7 dias:

```
find /var/log -name "*.log" -mtime +7 -exec rm -f {} \;
```

- Agende com `cron` para rodar diariamente:

```
0 3 * * * /home/usuario/limpar_logs.sh
```

Exercício 3: Crie um script que conte quantos arquivos `.txt` existem em uma pasta e exiba o total.

```
#!/bin/bash
quantidade=$(ls *.txt | wc -l)
echo "Existem $quantidade arquivos .txt"
```

Exercício 4: Crie um script que leia um arquivo linha por linha e imprima as linhas com numeração.

```
#!/bin/bash
contador=1
while read linha; do
  echo "$contador: $linha"
  ((contador++))
done < arquivo.txt
```

❓ Dúvidas Frequentes

1. O que acontece se eu esquecer os espaços dentro dos colchetes em `if`?

Erro! Os colchetes são comandos em Bash e exigem espaços: `if [ $x -eq 1 ]`

2. Posso usar `for` com intervalos?

Sim, usando `seq`: `for i in $(seq 1 10)`

3. Como saber se meu script está executando?

Adicione `echo` ou use `set -x` para debug.

4. Meu `cron` não executa o script, o que pode ser?

Verifique permissões, caminhos absolutos e se o script é executável (`chmod +x`).

5. Posso usar `cron` para scripts em Python ou outros?

Sim, desde que o interpretador esteja instalado e o caminho esteja correto no script (`#!/usr/bin/python3`, por exemplo).

Conclusão

- Scripts Bash são ferramentas poderosas para automatizar tarefas.

- O terminal oferece controle total sobre o sistema.
- `cron` é essencial para agendamentos confiáveis e manutenção de rotinas.

Se precisar de ajuda, consulte:

- GNU Bash Manual
- Crontab Guru
- Use `man comando` no terminal para ajuda integrada