Automatização de Tarefas com Bash e Cron Jobs

M.Sc. Ana Luiza Martins Karl

Professora Assistente – Curso de Análise e Desenvolvimento de Sistemas

Automatização de Tarefas com Bash e Cron Jobs

bit.ly/notionBash
github.com/aluizakarl/mc_bash-cron

- Técnica em Telecomunicações/ TV Digital – CEFET Petrópolis
- Bacharel em Biomedicina – Universidade Católica de Petrópolis (2016)
- Mestre em Modelagem Computacional pelo Laboratório Nacional de Computação Científica (LNCC/MCTI, 2019)
- Tecnóloga em Tecnologia da Informação e Comunicação pela FAETERJ Petrópolis (2023)

- Doutoranda em Modelagem Computacional pelo LNCC/MCTI;
- Bolsista de Desenvolvimento Tecnológico (DTI-A) na Universidade Estadual do Norte Fluminense (UENF)
- Professora Assistente – Curso de Análise e Desenvolvimento de Sistemas – UNIFESO
  - Disciplina de Redes de Computadores
  - Disciplina de Raciocínio Lógico e Matemático
  - Disciplina de Inteligência Artificial no Desenvolvimento Web*

* Disciplina optativa, em construção

Motivação do curso

Experiência com sistemas Linux desde 2009

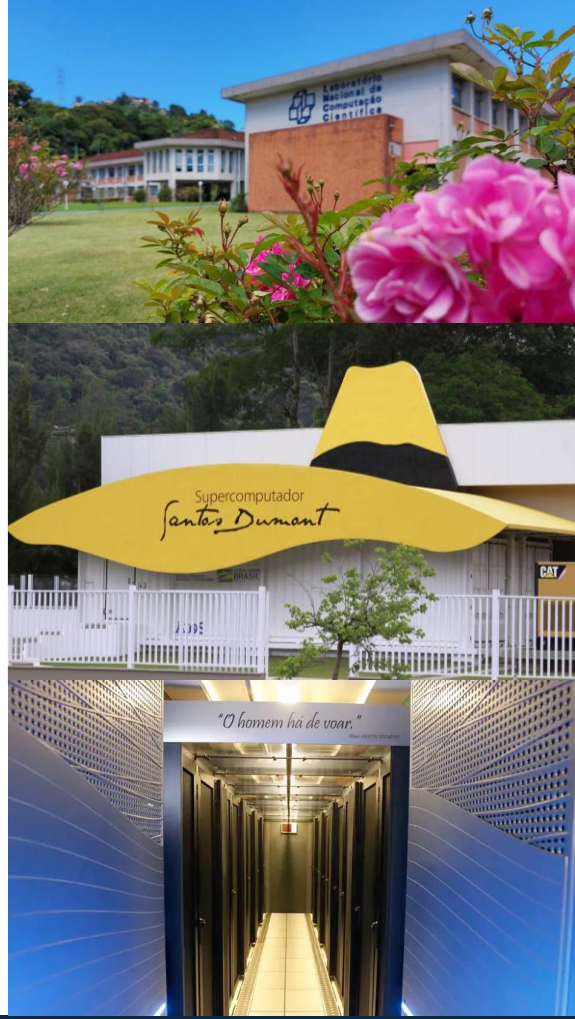
- Aprendizado de programação;
- Desenvolvimento e uso de softwares científicos;
- Paralelização de tarefas usando clusters e supercomputadores;
- Automatização de tarefas e análises científicas;

Supercomputador Santos Dumont

LNCC, Petrópolis

- Máquina mais eficiente da América Latina.
- Está entre os 100 supercomputadores mais rápidos do mundo (89º posição).
- Capacidade instalada de processamento na ordem de 20 Petaflop/s (20 x 10^{15} *float-point operations per second*).

lncc.br ◆ sdumont.lncc.br



SUMÁRIO

SUMÁRIO

Introdução

Linux

Linux é um sistema operacional de código aberto, conhecido por sua flexibilidade, segurança e desempenho.

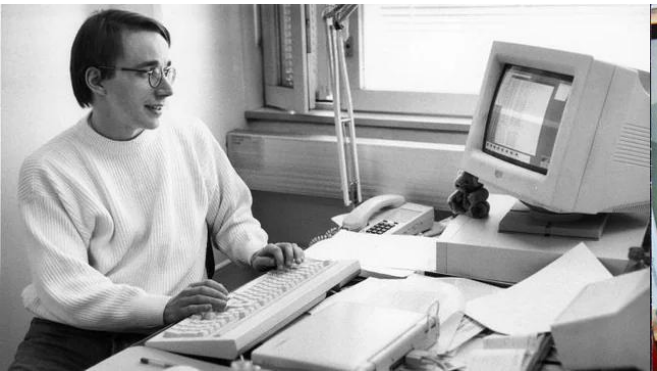
- É código aberto.
- É encontrado em diversas distribuições;
- É famoso por sua estabilidade e segurança;
- Possui uma grande e ativa comunidade de desenvolvedores;

Linux

O Linux surgiu em 1991, criado por Linus Torvalds.

- Inspirado no Unix (sistema robusto e estável usado em universidades);
- Objetivo: criar um sistema aberto, colaborativo e acessível a todos.

Curiosidade: Linus também criou o sistema de controle de versionamento git.



Linux

Graças a grande e ativa comunidade de desenvolvimento, o Linux é hoje um dos sistemas mais usados no mundo.

- 96% dos servidores de nuvem rodam Linux (AWS, Google, Cloud, Azure, etc.)
- O Android, usado por bilhões de pessoas, é baseado no Linux
- Ferramentas de automação, DevOps, containers e CI/CD rodam majoritariamente em Linux.
- Satélites, estações espaciais e equipamentos científicos utilizam distribuições Linux customizadas.
- Profissionais de TI, infraestrutura, DevOps, segurança e dados precisam dominar o ambiente Linux para crescer na carreira.

É um sistema operacional poderoso e versátil essencial para diversas carreiras de TI.

Por que estudar Bash e o terminal?

O terminal pode parecer intimidador no começo, mas ele permite:

- Automatizar tarefas repetitivas.
- Manipular grandes volumes de dados com eficiência.
- Interagir diretamente com o sistema operacional.
- Criar scripts que economizam tempo e evitam erros humanos.
- Se destacar em entrevistas, estágios e vagas técnicas.

Se você pretende seguir na carreira de desenvolvimento, dados ou segurança, dominar o terminal é um diferencial. Se você pretende seguir para áreas como DevOps, suporte técnico, redes ou administração de sistemas, é essencial.

Linux

As distribuições são sistemas operacionais completos, incluindo kernel (núcleo), ferramentas e ambiente de desktop.

Fundamentos do terminal

Antes de começar...

Como utilizar o terminal Linux no Windows:

- WSL (tutorial de como instalar: https://learn.microsoft.com/pt-br/windows/wsl/install);
- MobaXterm (https://mobaxterm.mobatek.net/download.html);
- VSCode (https://code.visualstudio.com/);

Terminal Linux

Terminal = interface de linha de comando

Ambiente baseado em texto onde os usuários podem interagir com o sistema operacional através de comandos.

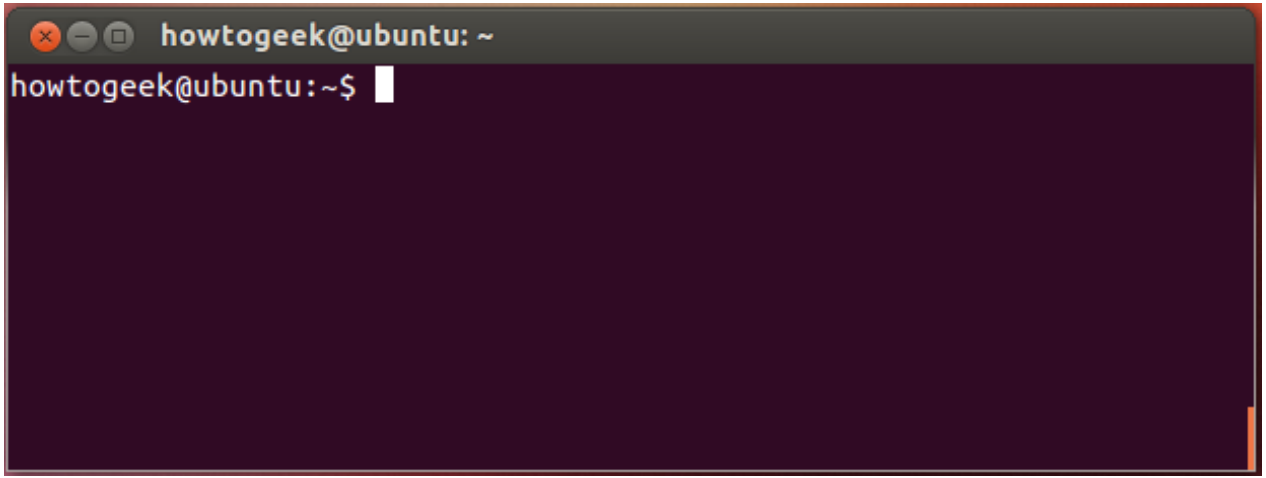
- Gerenciamento e manipulação de arquivos;
- Execução de programas
- Configuração do sistema;

⚠️ Muito usado em servidores, ambientes de desenvolvimento, automação e administração de sistemas.

Terminal Linux

Terminal = interface de linha de comando

Ambiente baseado em texto onde os usuários podem interagir com o sistema operacional através de comandos.



- O Bash (do inglês, *Bourne Again Shell*) é o shell* mais comum dos sistemas Linux;
- Possui características de uma linguagem de programação de alto nível;
- Oferece recursos de edição de linha de comando, como histórico de comandos, auto-completar e sugestões;
- Altamente personalizável;
- Portabilidade: é compatível com uma variedade de sistemas operacionais Unix;

* O shell é o interpretador que processa os comandos digitados no terminal.
Outros interpretadores shell: sh, zsh, fish, etc.

Por que usar o terminal?

- Mais rápido e poderoso que interfaces gráficas para várias tarefas.
- Permite automação de tarefas com scripts.
- Essencial para trabalhar com servidores, repositórios Git, contêineres, CI/CD, etc.
- Workflows modernos (Git, Docker, Kubernetes, etc.) são dirigidos por linha de comando.
- Depuração e *troubleshooting* rápidos: ver processos, portas, uso de memória ou logs sem abrir apps pesados.
- Documentação espontânea: o histórico (history > steps.sh) vira prova de auditoria.

Estrutura de diretórios Linux

Todas as distribuições Linux seguem uma forma padrão de organizar os arquivos, chamada de FHS (do inglês, *Filesystem Hierarchy Standard*).

/ - raiz - o principal diretório do sistema.

/home/ - diretórios dos usuários

/bin/ - programas/comandos essenciais

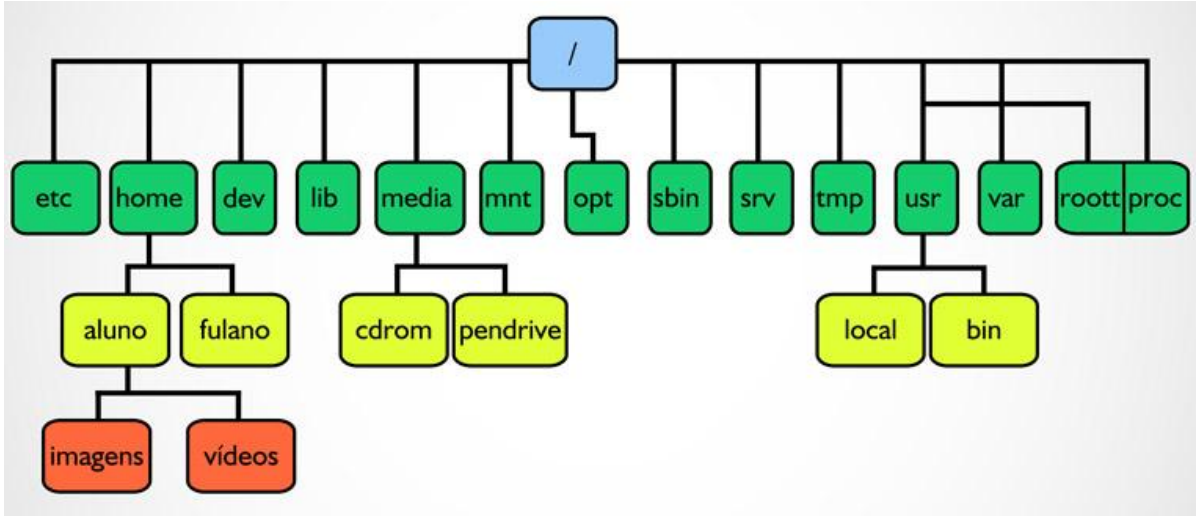
/etc/ - arquivos de configuração

/var/ - arquivos variáveis (logs, spool)

/tmp/ - arquivos temporários

Estrutura de diretórios Linux

Todas as distribuições Linux seguem uma forma padrão de organizar os arquivos, chamada de FHS (do inglês, *Filesystem Hierarchy Standard*).



Conceito de caminhos

Baseado na estrutura de diretórios, temos o conceito de caminhos no Linux:

Caminho absoluto: começa com /

Ex: /home/luiza/documentos

Caminho relativo: baseado na pasta atual

Ex: . /imagens

. . /projetos/amostras

Atalhos:

. – pasta atual

. . – pasta anterior

~ – pasta do diretório

Comandos

Comandos são programas executáveis residentes em /bin/

Ex: `ls`, `cd`, `echo`, `rm`, `cp`, `mkdir`, etc.

Os comandos possuem sintaxe própria e podem ter parâmetros e opções:

```
comando [opções] [parâmetros]
```

Ex: `ls -l /home/`

O comando `man` permite a consulta a documentação dos comandos padrões.

Ex: `man ls`

⚠️ O terminal Linux é *case sensitive*: distingue minúsculas e maiúsculas

Segurança

Alguns comandos exigem privilégios de superusuário (administrador).

O comando `sudo` permite que usuários comuns executem comandos com privilégios de administrador (root) — mas de forma controlada e temporária.

Para executar comandos como superusuário:

```
sudo comando [opções] [parâmetros]
```

⚠️ O `sudo` é necessário para instalar, atualizar e remover pacotes, alterar arquivos do sistema, criar usuários, alterar permissões, formatar discos, montar dispositivos, etc.

⚠️ ⚠️ ⚠️ Use o **sudo** somente quando necessário, e com cautela.

Lógica de *scripts*

O que é um *script*?

Um script é um conjunto de comandos organizados por uma lógica e armazenados em um arquivo de texto, que pode ser executado de forma sequencial por um interpretador.

```
#! /bin/bash

echo "Fazendo backup..."
cp -r ~/Documentos ~/media/storage/Backup
echo "Backup realizado com sucesso!"
```

Se eu salvo esses comandos em um arquivo backup.sh, consigo executar o script através de:

```
bash backup.sh
```

Ou, dando a permissão através do `chmod`:

```
chmod +x backup.sh
./backup.sh
```

O que é um *script*?

Ou seja, um script é um programa simples que automatiza tarefas.

Os scripts:

- São escritos em linguagem interpretada (como Bash, Python, etc.);
- Podem conter condições, laços, variáveis e funções;
- Servem para automatizar processos manuais e repetitivos;
- Facilitam a reprodutibilidade e reduzem erros humanos;

As variáveis em bash

Variáveis são nomes simbólicos para espaços de memória dentro de um script, programa ou sessão do terminal.

No bash, as variáveis:

- Não têm tipo (tudo é string).
- A atribuição é sem espaços.
- Para acessar, é necessário o uso do $

⚠️ Para evitar problemas com espaços ou caracteres especiais, use aspas na atribuição

As variáveis em bash

Por exemplo:

```
nome="Ana Luiza Martins Karl"
echo "Usuário: $nome"
```

Também é possível armazenar o resultado de um comando em uma variável:

```
data=$(date)
echo “Hoje é: $data"
```

ou

```
data=`date`
echo “Hoje é: $data"
```

As variáveis em bash

Seja o arquivo alunosUnifeso.txt com seguinte conteúdo:

```
Ana Carolina
Felipe
Gustavo
João Guilherme
Maria Eduarda
Vitória
Thiago
```

O conteúdo desse arquivo pode ser armazenado em uma variável da mesma maneira que um valor único:

```
listaAlunos=$(cat alunosUnifeso.txt)
echo “$listaAlunos"
```

Estruturas de controle de fluxo

Condicional if:

Permite que um bloco de código seja executado somente se a condição for verdadeira.

Ex:

```
if [ condição ]; then
    # comandos se verdadeiro
elif [ outra condição ]; then
    # outro caso
else
    # caso contrário
fi
```

Estruturas de controle de fluxo

Condicional if:

Permite que um bloco de código seja executado somente se a condição for verdadeira.

Ex:

```
idade=17
if [ $idade -ge 18 ]; then
    echo "Maior de idade"
else
    echo "Menor de idade"
fi
```

⚠ Em bash, os colchetes [ ] são comandos. Use-os com espaço.

Estruturas de controle de fluxo

Operadores de comparação:

Operador	Significado	Exemplo
-eq	Igual	[5 -eq 5]
-ne	Diferente	[5 -ne 3]
-lt	Menor que	[3 -lt 10]
-le	Menor ou igual	[3 -le 3]
-gt	Maior que	[10 -gt 5]
-ge	Maior ou igual	[10 -ge 10]

Estruturas de controle de fluxo

Operadores de comparação:

Operador	Significado	Exemplo
= ou ==	Igual	["$nome" == "Ana"]
!=	Diferente	["$nome" != "João"]
-z	String vazia	[-z "$texto"]
-n	String não vazia	[-n "$texto"]

Estruturas de controle de fluxo

Operadores de comparação:

Operador	Verifica se...	Exemplo
-e	O arquivo existe	[-e arquivo.txt]
-f	É um arquivo comum	[-f arquivo.txt]
-d	É um diretório	[-d /home/aluno]
-s	Arquivo existe e não está vazio	[-s log.txt]
-r	Possui permissão de leitura	[-r dados.txt]
-w	Possui permissão de escrita	[-w dados.txt]
-x	Possui permissão de execução	[-x script.sh]

Estruturas de controle de fluxo

Condicional if:

Ex:

```
arquivo="relatorio.txt"

if [ -e "$arquivo" ]; then
    echo "O arquivo existe."
else
    echo "Arquivo não encontrado."
fi
```

Estruturas de controle de fluxo

Laço for:

O laço for é usado para repetir um conjunto de comandos várias vezes, percorrendo uma lista de valores (como números, nomes de arquivos ou resultados de comandos).

Sintaxe:

```
for variavel in $lista
do
    comandos
done
```

Estruturas de controle de fluxo

Laço for:

O laço for é usado para repetir um conjunto de comandos várias vezes, percorrendo uma lista de valores (como números, nomes de arquivos ou resultados de comandos).

Ex:

```
listaAlunos=$(cat alunosUnifeso.txt)

for nome in $listaAlunos
do
    echo "Olá, $nome!"
done
```

Estruturas de controle de fluxo

Laço for:

O laço for é usado para repetir um conjunto de comandos várias vezes, percorrendo uma lista de valores (como números, nomes de arquivos ou resultados de comandos).

Ex:

```
for nome in “Ana Luiza Martins Karl”
do
    echo "Olá, $nome!"
done
```

Estruturas de controle de fluxo

Laço for:

O laço for é usado para repetir um conjunto de comandos várias vezes, percorrendo uma lista de valores (como números, nomes de arquivos ou resultados de comandos).

Ex:

```
for i in $(seq 1 5)
do
    echo "Número: $i"
done
```

Use o for para repetir comandos para vários arquivos ou diretórios, realizar operações em série (como backups, copias e renomeações) e automatizar testes com diferentes entradas.

Estruturas de controle de fluxo

Laço while:

Usado para repetir um bloco de comandos enquanto uma condição for verdadeira.

Sintaxe:

```
while [ condição ]
do
    comandos
done
```

Estruturas de controle de fluxo

Laço while:

Usado para repetir um bloco de comandos enquanto uma condição for verdadeira.

Ex:

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

Estruturas de controle de fluxo

Laço while:

Usado para repetir um bloco de comandos enquanto uma condição for verdadeira.

Ex:

```
resposta=""

while [ "$resposta" != "sair" ]
do
    echo "Digite algo (ou 'sair' para encerrar):"
    read resposta
done
```

Estruturas de controle de fluxo

Laço while:

Usado para repetir um bloco de comandos enquanto uma condição for verdadeira.

Ex:

```
while read linha
do
    echo "Linha do arquivo: $linha"
done < arquivo.txt
```

Entrada e Saída de dados

Em scripts bash também é possível interação com o usuário através de entrada e saída de dados:

Entrada de dados: a entrada de dados é dada pelo comando **read**

Saída de dados: a saída de dados é dada pelo comando **echo**

Ex:

```
echo "Digite seu nome:"
read nome
echo "Bem-vindo(a), $nome!"
```

Usando parâmetros

No Bash, é possível receber entradas diretamente como parâmetros da linha de execução do script. Isso é feito usando variáveis posicionais, como $1, $2, etc.

Ex:

```
#!/bin/bash

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

No terminal:

```
Bash boas_vindas.sh Luiza Bash
```

SUMÁRIO

Relembrando...

01. Introdução

- Breve história do Linux e seu criador Linus Tovarlds.
- Importância do Linux em servidores, nuvem, IoT e dispositivos móveis.
- Filosofia do software livre e da colaboração.

02. Bash e Terminal

- Terminal: interface para interação direta com o SO.
- Bash: interpretador de comando mais comum do Linux.
- Vantagens de usar o terminal: automação, controle e eficiência.

Relembrando...

03. Comandos básicos

- Navegação: `cd`, `ls`, `pwd`
- Manipulação de arquivos: `touch`, `mkdir`, `rm`, `mv`, `cp`
- Leitura e inspeção: `cat`, `less`, `head`, `tail`
- Permissão de acesso: `chmod`, `sudo`

04. Lógica de scripts

- Variáveis: armazenar e reutilizar informações.
- Condicional `if`: executar comandos baseado em condições
- Estruturas de repetição:
  - `for`: percorre listas de itens;
  - `while`: repete até que uma condição seja falsa
- Entrada por parâmetros

Introdução ao Cron

Por que agendar tarefas?

Em um sistema Linux, muitas atividades são repetitivas e rotineiras: fazer backups, limpar arquivos antigos, atualizar pacotes, monitorar recursos, enviar notificações...

Agora imagine ter que fazer tudo isso manualmente, todos os dias, nos horários certos...

- Tarefas repetitivas desperdiçam tempo.
- Tarefas noturnas nem sempre podem ser feitas manualmente.
- Tarefas programadas aumentam a confiabilidade e organização.

Agendar tarefas é útil quando:

- Você não pode estar presente no momento da execução.
- A tarefa precisa ocorrer em intervalos regulares (horários, dias, semanas).
- Você quer garantir que algo aconteça independente da memória humana.
- Seu sistema precisa realizar manutenção automática.

Ex: Backup automático de arquivos importantes, limpeza de diretórios temporários a cada domingo, atualização do sistema de madrugada, envio de lembretes de tarefas para o e-mail, coleta de dados de APIs para análise futura, etc.

Vantagens do Cron

- Já vem instalado na maioria das distribuições Linux.
- Leve, confiável e fácil de configurar.
- Executa qualquer comando ou script, em qualquer frequência desejada.

Em resumo, automatizar tarefas com `cron` significa ganhar tempo, evitar esquecimentos e aumentar a confiabilidade dos processos.

Crontab

O Crontab (ou crontable) é um arquivo de texto onde cada linha representa um agendamento de tarefa. Essas tarefas são executadas automaticamente pelo serviço cron, que roda em segundo plano no Linux.

⚠️ Cada usuário pode ter seu próprio crontab: ou seja, você pode criar tarefas específicas para sua conta, sem afetar outros usuários do sistema.

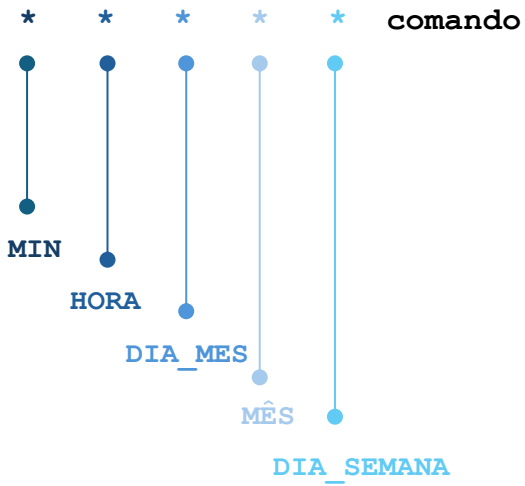
Crontab

O Crontab (ou crontable) é um arquivo de texto onde cada linha representa um agendamento de tarefa. Essas tarefas são executadas automaticamente pelo serviço cron, que roda em segundo plano no Linux.

⚠️ Cada usuário pode ter seu próprio crontab: ou seja, você pode criar tarefas específicas para sua conta, sem afetar outros usuários do sistema.

Estrutura do Crontab

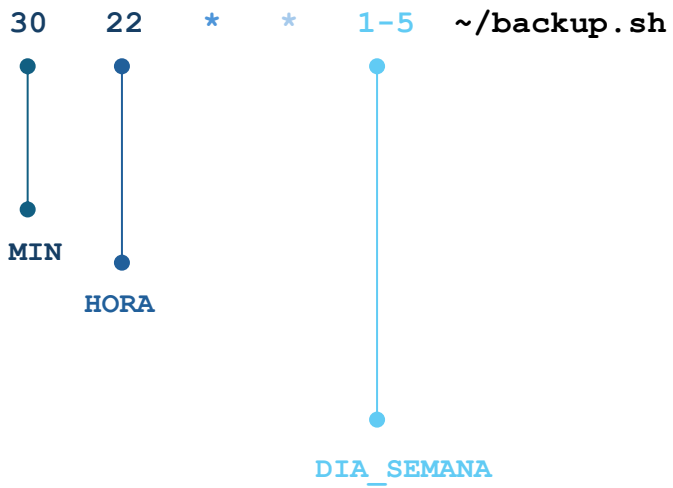
Uma linha no crontab segue este formato:



Estrutura do Crontab

Uma linha no crontab segue este formato:

Ex:



Essa linha executa o backup às 22:30 de segunda à sexta-feira.

Boas práticas

- Use sempre caminhos absolutos no comando (ex: /home/usuário/script.sh)
- Torne seu script executável: chmod +x script.sh
- Verifique sempre se o script roda fora do cron.
- Teste redirecionando a saída para um arquivo:

```
* * * * * /caminho/script.sh >> /home/usuario/saida.log 2>&1
```

Obrigada =)

M.Sc. Ana Luiza Martins Karl

Professora Assistente – Curso de Análise e Desenvolvimento de Sistemas

@: anamartins@unifeso.edu.br
github: aluizakarl