

PROGRAMAÇÃO PARA AUTOMAÇÃO DE SERVIDORES LINUX

Apostila prática de Bash Shell Script

Curso Técnico em Redes de Computadores • Debian • Do algoritmo ao trabalho real do técnico de infraestrutura

Revisão breve de lógica de programação + laboratórios progressivos + projeto integrador.

Apresentação e orientações

Esta apostila complementa o estudo prévio de lógica de programação. O foco é criar scripts Bash que automatizem consultas, diagnósticos, monitoramento, relatórios e tarefas repetitivas em servidores Debian.

Não exige sudo nas práticas básicas.

Metodologia por unidade: conceito → exemplo → previsão do resultado → execução → alteração → exercícios

→ desafio. Antes de rodar, cada estudante deve escrever o que espera ver; depois, comparar a previsão com a saída real.

Regras de segurança: use sua conta sem sudo; trabalhe dentro de ~/lab_bash; não execute scripts desconhecidos; não altere /etc, /var/log ou serviços de produção; não use rm -rf; faça backups somente de

dados do laboratório. Scripts que usam rede dependem de conectividade e políticas locais.

Preparação única:

```
mkdir -p "$HOME/lab_bash"
cd "$HOME/lab_bash"
pwd
bash --version | head -n 1
command -v ping; command -v ip; command -v ss
```

Se ping, ip ou ss não estiverem instalados, informe ao professor. Não instale pacotes por conta própria em

servidor compartilhado. Execute sempre os exemplos na sua pasta.

Mapa de competências da ementa

Sequência, instruções e algoritmos: unidades 1–2. Constantes, variáveis, tipos e entrada/saída: 1–3.

Operadores aritméticos, relacionais e lógicos: 3–5. Repetições e arquivos de dados: 5–8. Ferramentas, casos

e ambientes: todas as unidades. Programação orientada: apresentação conceitual ao final; Bash é procedural e não orientado a objetos.

Parte I — Revisão rápida de lógica aplicada ao Linux

Algoritmo é uma sequência finita de instruções. Em Bash, executamos comandos de cima para baixo, guardamos texto e números em variáveis, tomamos decisões com if e repetimos tarefas com for e while. O

código de saída 0 normalmente indica sucesso; outro valor indica falha.

```
#!/bin/bash
nome="Servidor-LAB" # texto
limite=80           # número inteiro
echo "Alvo: $nome"
if (( limite >= 80 )); then
```

```
echo "Limite de alerta configurado"  
fi  
for item in cpu memoria disco; do  
echo "Verificar: $item"  
done
```

No Bash, variáveis normalmente armazenam texto; a aritmética inteira usa ((...)) ou \$((...)). Aspas em "\$variavel" evitam separação indevida de palavras. A primeira linha #!/bin/bash indica o interpretador.

Unidade 1 — Meu primeiro inventário de servidor

Objetivo profissional: Identificar rapidamente o ambiente em que o técnico está conectado.

Conceito e situação real

Um inventário reúne nome da máquina, usuário, kernel e data. São consultas sem alteração de sistema.

Preparação e execução

No terminal: nano nome_do_script.sh (cole o código, salve com Ctrl+O e saia com Ctrl+X). Execute com bash

nome_do_script.sh. Os nomes dos arquivos de cada unidade podem ser adaptados.

Script comentado / exemplo funcional

```
#!/bin/bash
# inventario.sh
echo "=== INVENTARIO DO SERVIDOR ==="
echo "Usuário: $(whoami)"
echo "Máquina: $(hostname)"
echo "Kernel: $(uname -r)"
echo "Data: $(date '+%d/%m/%Y %H:%M')"
echo "Diretorio: $HOME"
```

Entenda o resultado

\$(comando) captura a saída de outro comando. O script organiza várias consultas numa execução.

Experimente modificar

Acrescente a arquitetura com uname -m e a distribuição com cat /etc/os-release.

Exercícios

1. Explique a ordem das instruções.
2. Registre o hostname e o kernel encontrados.
3. Faça o script imprimir uma linha de separação ao final.

Desafio

Gere um inventário que inclua o tempo ligado do sistema com uptime -p.

Como conferir

Compare as saídas com os comandos executados isoladamente.

Unidade 2 — Entrada, variáveis e validação

Objetivo profissional: Receber parâmetros para um diagnóstico sem editar o código.

Conceito e situação real

read recebe entrada do teclado. Uma variável vazia pode produzir um teste sem sentido; portanto, validamos antes de usar.

Preparação e execução

No terminal: nano nome_do_script.sh (cole o código, salve com Ctrl+O e saia com Ctrl+X). Execute com bash

nome_do_script.sh. Os nomes dos arquivos de cada unidade podem ser adaptados.

Script comentado / exemplo funcional

```
#!/bin/bash
# identifica.sh

read -r -p "Informe o nome do equipamento: " equipamento

if [[ -z "$equipamento" ]]; then
    echo "Erro: nome nao informado"
    exit 1
fi

echo "Equipamento cadastrado para consulta: $equipamento"
echo "Operador: $(whoami)"
```

Entenda o resultado

-r impede interpretação de barras invertidas. -z testa se o texto está vazio. exit 1 informa erro ao sistema.

Experimente modificar

Acrescente uma pergunta para o setor e exiba os dois campos.

Exercícios

1. Execute pressionando Enter sem informar nome.
2. Explique a diferença entre echo e read.
3. Use nomes com espaços para testar as aspas.

Desafio

Solicite nome e local do equipamento; só exiba o cadastro quando ambos forem preenchidos.

Unidade 3 — Condicionais e conectividade

Objetivo profissional: Verificar se um endereço responde a ICMP.

Conceito e situação real

ping testa resposta ICMP, não a disponibilidade de todos os serviços. Um firewall pode bloquear ping mesmo

com o host funcionando.

Preparação e execução

No terminal: nano nome_do_script.sh (cole o código, salve com Ctrl+O e saia com Ctrl+X). Execute com bash

nome_do_script.sh. Os nomes dos arquivos de cada unidade podem ser adaptados.

Script comentado / exemplo funcional

```
#!/bin/bash
# ping_alvo.sh

read -r -p "IP ou nome do destino: " alvo

if [[ -z "$alvo" || "$alvo" == -* ]]; then
    echo "Destino invalido"
    exit 1
fi

if ping -c 2 -W 2 -- "$alvo" >/dev/null 2>&1; then
```

```
echo "Resposta ICMP recebida de $alvo"  
else  
echo "Sem resposta ICMP de $alvo"  
fi
```

Entenda o resultado

if avalia o status do comando. &&/|| e [[...]] representam operadores lógicos. O redirecionamento descarta

saída para mostrar apenas o resumo.

Experimente modificar

Teste primeiro 127.0.0.1 e depois um endereço definido pelo professor.

Exercícios

1. Qual a função de -c 2?
2. Explique por que ausência de resposta não prova servidor desligado.
3. Troque a mensagem de falha por uma orientação de diagnóstico.

Desafio

Inclua a data e o destino no resultado; registre o status em arquivo dentro do laboratório.

Unidade 4 — Aritmética e alerta de disco

Objetivo profissional: Avisar quando a ocupação da partição raiz ultrapassar um limite.

Conceito e situação real

df -P / mostra ocupação em formato estável; awk extrai o percentual. A comparação é inteira, sem unidades.

Preparação e execução

No terminal: nano nome_do_script.sh (cole o código, salve com Ctrl+O e saia com Ctrl+X). Execute com bash

nome_do_script.sh. Os nomes dos arquivos de cada unidade podem ser adaptados.

Script comentado / exemplo funcional

```
#!/bin/bash

# alerta_disco.sh

limite=80

uso=$(df -P / | awk 'NR==2 {gsub(/%/, ""); print $5}')

if [[ ! "$uso" =~ ^[0-9]+$ ]]; then
    echo "Nao foi possivel obter o uso do disco"
    exit 1
fi

echo "Uso da raiz: ${uso}% | Limite: ${limite}%"

if (( uso >= limite )); then
    echo "ALERTA: verificar espaco disponivel"
else
    echo "Situacao dentro do limite"
fi
```

Entenda o resultado

A expressão regular confere que uso contém números. (()) realiza a comparação numérica.

Experimente modificar

Altere o limite para 20, 50 e 95 e observe as mudanças.

Exercícios

1. Explique a diferença entre $\geq$ e $>$.

2. Execute `df -h /` e compare a leitura.
3. Mostre o valor de uso sem o sinal de porcentagem.

Desafio

Permita informar o limite por argumento e rejeite valores fora de 0 a 100.

Unidade 5 — Repetições: vários hosts

Objetivo profissional: Automatizar a verificação de uma pequena lista de equipamentos.

Conceito e situação real

for percorre uma lista. Os endereços abaixo são exemplos reservados para documentação; substitua pelos

IPs autorizados do laboratório.

Preparação e execução

No terminal: `nano nome_do_script.sh` (cole o código, salve com `Ctrl+O` e saia com `Ctrl+X`). Execute com `bash`

`nome_do_script.sh`. Os nomes dos arquivos de cada unidade podem ser adaptados.

Script comentado / exemplo funcional

```
#!/bin/bash
# verifica_hosts.sh

hosts=("127.0.0.1" "192.0.2.10" "192.0.2.11")
for host in "${hosts[@]}; do
if ping -c 1 -W 1 -- "$host" >/dev/null 2>&1; then
echo "$host;RESPONDE"
else
echo "$host;SEM_RESPOSTA_ICMP"
fi
done
```

Entenda o resultado

Um array armazena vários valores; o loop repete o mesmo teste sem duplicar o código.

Experimente modificar

Substitua os exemplos por três IPs indicados pelo professor.

Exercícios

1. Conte quantas iterações ocorrem.
2. Ajuste a quantidade de tentativas.
3. Inclua uma mensagem de início e término.

Desafio

Conte quantos destinos responderam e quantos não responderam.

Unidade 6 — Arquivos de dados e logs

Objetivo profissional: Registrar o resultado de verificações para consulta posterior.

Conceito e situação real

>> acrescenta linhas a um arquivo; > sobrescreve. O log deve ficar na pasta do aluno, não em /var/log.

Preparação e execução

No terminal: nano nome_do_script.sh (cole o código, salve com Ctrl+O e saia com Ctrl+X). Execute com bash

nome_do_script.sh. Os nomes dos arquivos de cada unidade podem ser adaptados.

Script comentado / exemplo funcional

```
#!/bin/bash
# log_rede.sh
arquivo="$HOME/lab_bash/rede.csv"
read -r -p "Destino: " alvo
if [[ -z "$alvo" || "$alvo" == -* ]]; then
echo "Destino invalido"; exit 1
fi
status="SEM_RESPOSTA_ICMP"
if ping -c 1 -W 2 -- "$alvo" >/dev/null 2>&1; then
status="RESPONDE"
fi
printf '%s;%s;%s\n' "$(date '+%F %T')" "$alvo" "$status" >> "$arquivo"
echo "Registro gravado em $arquivo"
```

Entenda o resultado

O arquivo usa ponto e vírgula como separador. O registro preserva data, destino e resultado; um IP ou hostname informado pelo usuário não deve conter ponto e vírgula se for exportado como CSV.

Experimente modificar

Execute três vezes e consulte cat ~/lab_bash/rede.csv.

Exercícios

1. Qual a diferença entre > e >>?
2. Exiba apenas as últimas linhas com tail.
3. Conte registros com wc -l.

Desafio

Crie um cabeçalho CSV somente quando o arquivo ainda não existir; valide os caracteres do destino.

Unidade 7 — Usuários e grupos sem sudo

Objetivo profissional: Consultar identidades e permissões sem modificar contas.

Conceito e situação real

id e getent consultam contas. A existência de uma conta não significa que ela esteja conectada ou desbloqueada.

Preparação e execução

No terminal: nano nome_do_script.sh (cole o código, salve com Ctrl+O e saia com Ctrl+X). Execute com bash

nome_do_script.sh. Os nomes dos arquivos de cada unidade podem ser adaptados.

Script comentado / exemplo funcional

```
#!/bin/bash
# consulta_usuario.sh

read -r -p "Usuario Linux: " usuario

if [[ ! "$usuario" =~ ^[a-z_][a-z0-9_-]*[!@#$%^&*]?$ ]]; then
    echo "Nome de usuario invalido"; exit 1
fi

if getent passwd "$usuario" >/dev/null; then
    echo "Conta encontrada"
    id "$usuario"
else
    echo "Conta nao encontrada"
fi
```

Entenda o resultado

getent consulta a base de usuários configurada, que pode incluir diretórios externos. id apresenta UID, GID e grupos.

Experimente modificar

Consulte sua própria conta e uma conta inexistente.

Exercícios

1. Explique o que é UID.
2. Identifique seu grupo principal.
3. Por que o script não precisa de sudo?

Desafio

Construa um relatório de usuários informados pelo professor, sem revelar senhas ou dados sensíveis.

Unidade 8 — Monitoramento de recursos

Objetivo profissional: Criar um resumo de CPU, memória, disco e carga.

Conceito e situação real

free, uptime, df e ps fornecem dados úteis. Carga média não é igual ao percentual instantâneo de CPU.

Preparação e execução

No terminal: nano nome_do_script.sh (cole o código, salve com Ctrl+O e saia com Ctrl+X). Execute com bash

nome_do_script.sh. Os nomes dos arquivos de cada unidade podem ser adaptados.

Script comentado / exemplo funcional

```
#!/bin/bash
# monitor.sh

echo "=== MONITORAMENTO $(date '+%F %T') ==="

echo "--- Memoria ---"

free -h

echo "--- Disco raiz ---"
```

```
df -h /  
echo "--- Carga do sistema ---"  
uptime  
echo "--- Processos por CPU ---"  
ps -eo pid,comm,%cpu --sort=-%cpu | head -n 6
```

Entenda o resultado

O pipeline | passa a saída de um comando para outro; head limita a lista.

Experimente modificar

Redirecione a saída para um relatório: `bash monitor.sh > relatorio.txt`.

Exercícios

1. Explique o significado de memória disponível.
2. Compare `df -h` e `free -h`.
3. Identifique os cinco primeiros processos exibidos.

Desafio

Crie um relatório datado e inclua hostname e nome do operador.

Unidade 9 — Diagnóstico de rede

Objetivo profissional: Consultar interfaces, rotas e portas em escuta.

Conceito e situação real

`ip` mostra endereços e rotas; `ss` lista sockets. Portas em escuta não garantem acesso externo.

Preparação e execução

No terminal: `nano nome_do_script.sh` (cole o código, salve com `Ctrl+O` e saia com `Ctrl+X`). Execute com `bash`

`nome_do_script.sh`. Os nomes dos arquivos de cada unidade podem ser adaptados.

Script comentado / exemplo funcional

```
#!/bin/bash
# diagnostico_rede.sh
echo "=== INTERFACES ==="
ip -brief address
echo "=== ROTAS ==="
ip route
echo "=== PORTAS TCP EM ESCUTA ==="
ss -ltn
echo "=== RESOLUCAO DE NOME (se disponivel) ==="
if command -v getent >/dev/null 2>&1; then
  getent ahostsv4 debian.org | head -n 3
fi
```

Entenda o resultado

`-ltn` lista sockets TCP em escuta com portas numéricas. O teste de nome depende de DNS e conectividade.

Experimente modificar

Identifique uma interface e sua rota padrão, quando houver.

Exercícios

1. Qual comando revela o gateway padrão?
2. Explique a diferença entre endereço IP e porta.
3. O que significa LISTEN?

Desafio

Monte um relatório com interface, rota e portas em escuta sem modificar o firewall.

Unidade 10 — Backup seguro de laboratório

Objetivo profissional: Automatizar uma cópia datada de arquivos próprios.

Conceito e situação real

tar compacta dados e permite restaurá-los. O exemplo trabalha exclusivamente na pasta do aluno.

Preparação e execução

No terminal: nano nome_do_script.sh (cole o código, salve com Ctrl+O e saia com Ctrl+X). Execute com bash

nome_do_script.sh. Os nomes dos arquivos de cada unidade podem ser adaptados.

Script comentado / exemplo funcional

```
#!/bin/bash
# backup_lab.sh

base="$HOME/lab_bash"
origem="$base/dados"
destino="$base/backups"

mkdir -p -- "$origem" "$destino"

data=$(date '+%Y%m%d_%H%M%S')
arquivo="$destino/backup_${data}.tar.gz"

if tar -czf "$arquivo" -C "$base" dados; then
    echo "Backup criado: $arquivo"
    tar -tzf "$arquivo" | head
else
    echo "Falha no backup"; exit 1
fi
```

Entenda o resultado

-C define a pasta base; tar -tzf lista o conteúdo sem extrair. O nome com segundos reduz colisões, mas execuções simultâneas ainda podem coincidir.

Experimente modificar

Crie arquivos de teste dentro de ~/lab_bash/dados e repita o backup.

Exercícios

1. Liste os arquivos do backup.
2. Explique por que verificar a restauração é importante.
3. Qual a diferença entre origem e destino?

Desafio

Restaure em ~/lab_bash/restauracao com mkdir -p e tar -xzf, usando somente o arquivo criado por você.

Unidade 11 — Menus, funções e cases

Objetivo profissional: Reunir rotinas de suporte em um único programa.

Conceito e situação real

case seleciona uma ação; funções evitam repetição e organizam o código.

Preparação e execução

No terminal: nano nome_do_script.sh (cole o código, salve com Ctrl+O e saia com Ctrl+X). Execute com bash

nome_do_script.sh. Os nomes dos arquivos de cada unidade podem ser adaptados.

Script comentado / exemplo funcional

```
#!/bin/bash
# menu_suporte.sh
inventario() { hostname; uname -r; }
disco() { df -h /; }
rede() { ip -brief address; }
while true; do
echo "1) Inventario 2) Disco 3) Rede 0) Sair"
read -r -p "Opcao: " opcao
case "$opcao" in
1) inventario ;;
2) disco ;;
3) rede ;;
0) echo "Encerrando"; break ;;
*) echo "Opcao invalida" ;;
esac
done
```

Entenda o resultado

while mantém o menu ativo; case executa o bloco escolhido; break encerra o laço.

Experimente modificar

Inclua uma opção para consultar uptime.

Exercícios

1. Por que usamos funções?
2. O que acontece ao digitar 9?
3. Qual a diferença entre break e exit?

Desafio

Integre o script de diagnóstico e o monitoramento criados nas unidades anteriores.

Unidade 12 — Agendamento e operação responsável

Objetivo profissional: Preparar a execução periódica de relatórios sem intervenção manual.

Conceito e situação real

cron agenda comandos; o uso depende da política do servidor e da disponibilidade do serviço. Prefira simular

o agendamento antes de instalá-lo.

Preparação e execução

No terminal: nano nome_do_script.sh (cole o código, salve com Ctrl+O e saia com Ctrl+X). Execute com bash

nome_do_script.sh. Os nomes dos arquivos de cada unidade podem ser adaptados.

Script comentado / exemplo funcional

```
#!/bin/bash
# relatorio_agendavel.sh
base="$HOME/lab_bash"
mkdir -p "$base/relatorios"
arquivo="$base/relatorios/status_$(date '+%Y%m%d_%H%M%S').txt"
{
date
hostname
uptime
df -h /
} > "$arquivo"
echo "Relatorio: $arquivo"
```

Entenda o resultado

Execute com `bash ~/lab_bash/relatorio_agendavel.sh`. Para cron, uma entrada ilustrativa seria: `0 8 * * * /bin/bash /home/USUARIO/lab_bash/relatorio_agendavel.sh`. Substitua USUARIO pelo login real e só configure após autorização.

Experimente modificar

Simule três execuções manuais e confira os arquivos.

Exercícios

1. Explique o que significa `0 8 * * *`.
2. Por que caminhos absolutos são importantes no cron?
3. O que pode acontecer se relatórios crescerem sem limite?

Desafio

Proponha uma política de retenção de arquivos, sem implementar exclusões automáticas no servidor compartilhado.

Projeto integrador — Central de suporte Debian

Desenvolva um programa `central_suporte.sh` com menu que permita: (1) inventário do servidor; (2) monitoramento de disco/memória; (3) diagnóstico de interfaces e rotas; (4) ping de destino informado; (5) consulta de usuário; (6) criação de relatório; (0) sair. Use funções, condicionais, laços e arquivos. Todos os resultados gravados devem ficar em `~/lab_bash`.

Etapas de desenvolvimento

1. Desenhe o fluxograma do menu e identifique entradas, processamento e saídas.
2. Implemente duas opções e teste os casos de sucesso e erro.
3. Acrescente as demais opções reaproveitando funções dos exercícios.
4. Registre um relatório datado sem sobrescrever os anteriores.
5. Demonstre o programa com um roteiro de testes e explique as decisões de segurança.

Critérios de avaliação (10 pontos)

Funcionamento das opções (3); aplicação de variáveis, decisões, repetições e funções (2); tratamento de erros e segurança (2); organização e comentários (1); demonstração e interpretação dos resultados (2).

Checklist de testes do estudante

1. O script funciona com `bash nome.sh`?
2. Os caminhos apontam apenas para a pasta do laboratório?
3. Entradas vazias e opções inválidas são tratadas?
4. O resultado é claro mesmo quando o comando falha?
5. Os arquivos gerados podem ser localizados e consultados?
6. O script não solicita `sudo` nem altera configurações do servidor?

Nota sobre programação orientada a objetos

A ementa menciona programação orientada. Bash trabalha principalmente com procedimentos, funções, condições e laços; não possui classes e objetos nativos como Python ou Java. Nesta disciplina, o estudante

deve reconhecer a diferença conceitual entre programação procedural e orientada a objetos, sem desviar o

foco da automação operacional.

Glossário essencial

Shell: interpretador de comandos. Bash: shell e linguagem de scripts. Script: arquivo com instruções.

Variável: nome associado a valor. Condição: decisão baseada em teste. Laço: repetição. Função: bloco reutilizável. Código de saída: status numérico de um comando. Log: registro de eventos. Cron: agendador de

tarefas. SSH: acesso remoto seguro.

Referências e consulta

GNU Bash Reference Manual — <https://www.gnu.org/software/bash/manual/>

Debian Administrator's Handbook — <https://debian-handbook.info/>

Debian Manpages — <https://manpages.debian.org/>

No terminal: `man bash`; `man ping`; `man ip`; `man ss`; `man tar`; `man crontab`. A disponibilidade dos comandos

depende da instalação e das permissões locais.