CURSO PRÁTICO EM 12 AULAS

Linux na Prática
scripts e automação

Escreva em Bash programas que trabalham por você, passo a passo, com demonstrações práticas, exercícios e atividades para praticar em casa.



Ștefan Nistor

Uma publicação do projeto Terminal Prático

Linux na Prática

scripts e automação

Escreva em Bash programas que trabalham por você, passo a passo, com demonstrações práticas, exercícios e atividades para praticar em casa.

Ștefan Nistor

Sobre esta publicação

Linux na Prática é uma publicação do Terminal Prático, projeto educacional de tecnologia idealizado por Ștefan Nistor, com foco no aprendizado por meio de exemplos e exercícios práticos. É a continuação de *Linux para Iniciantes*.

Título	Linux na Prática: scripts e automação
Autor	Ștefan Nistor
Idioma	Português do Brasil
Projeto	Terminal Prático
Website	https://terminalpratico.com.br/
Ano da publicação	2026

Todos os exemplos deste livro foram executados e conferidos no Debian 13, no laboratório do autor. As explicações, os exercícios e as atividades foram escritos especialmente para este curso.

Como usar este guia

Este guia tem 12 aulas. Cada uma começa com uma página que mostra o que você vai aprender e depois explica os comandos e as ideias novas um por um, com exemplos exatamente como aparecem no terminal. No final há exercícios, um desafio para você resolver por conta própria e uma atividade para fazer antes da aula seguinte.

A partir da aula 6 você escreve scripts: arquivos cheios de comandos que você executa como qualquer outro programa. A cada aula eles ficam um pouco mais espertos, e na aula 12 você junta tudo em um script de backup que roda sozinho, à noite.

Como ler os exemplos

```
lab@lpic1-deb:~$ whoami
lab
# as linhas cinza são explicações; não digite
```

- Digite o que vem depois de `lab@lpic1-deb:~$` e pressione Enter. O prompt você não digita.
- O prompt também mostra a pasta em que você está: `~` é o seu diretório pessoal, e depois de `cd laboratorio-scripts` ele vira `lab@lpic1-deb:~/laboratorio-scripts$`.
- As linhas brancas logo abaixo são a resposta do computador.
- Um comando longo demais para a largura da página continua na linha de baixo. Você o digita em uma linha só.
- Uma linha que começa com `>` continua o comando de cima: o bash a mostra quando um comando (um laço `for`, por exemplo) ainda não terminou quando você aperta Enter. Digite só o que vem depois de `>`.
- Os scripts aparecem em fundo azul-claro, com o nome do arquivo em cima. Você os digita no nano, linha por linha.
- Os exemplos foram executados em um computador chamado `lpic1-deb`, com o usuário `lab`. No seu computador os nomes, as datas e os horários serão outros. Isso é normal.
- As mensagens dos programas aparecem em inglês, como em qualquer Linux instalado em inglês. Os exemplos as mostram exatamente como saíram.

Os quadros do texto

Quadro azul
Resumos e desafios: coisas para lembrar ou para tentar por conta própria.

Quadro amarelo
Avisos e atividades: aqui é fácil errar, ou há algo para você fazer.

Como aprender

- Digite cada comando e cada script você mesmo, não copie e cole. Os erros de digitação e as mensagens que eles causam ensinam tanto quanto os exemplos que funcionam de primeira.
- Antes de executar um script, leia-o linha por linha, em voz alta se ajudar. Isso pega metade dos erros.
- Quando não souber o que um comando faz, pergunte ao `man` (ou ao `help`, para os comandos que fazem parte do shell, como você verá na aula 1).
- Faça a atividade entre as aulas: de 20 a 30 minutos bastam.

Sumário

Como usar este guia 3

Antes de começar 9

Aula 1. Como o shell encontra os comandos 11

1.1 Quem responde a um comando 12

1.2 PATH: onde o shell procura os programas 12

1.3 Variáveis 13

1.4 Variáveis para outros programas: export 14

1.5 A pasta dos seus scripts: ~/bin 14

1.6 Configurações que ficam: ~/.bashrc e alias 16

1.7 Prática 17

Resumo 17

Aula 2. Expressões regulares com grep 18

2.1 Um log de verdade 19

2.2 Várias opções de uma vez: grep -E e | 19

2.3 Caracteres especiais: o ponto, ^ e $ 20

2.4 Classes e repetições 21

2.5 Só a parte que combina: grep -o 21

2.6 O que não combina: grep -v 22

2.7 Prática 23

Resumo 24

Aula 3. Ferramentas de texto: cortar, ordenar, contar 25

3.1 Arquivos com colunas 26

3.2 Cortar colunas: cut 26

3.3 Ordenar e contar: sort e uniq -c 27

3.4 Trocar caracteres: tr 28

3.5 Tabelas e listas: column e paste 29

3.6 Salvar no meio do caminho: tee 29

3.7 Prática 29

Resumo 30

Aula 4. sed: buscar e substituir 31

4.1 A primeira substituição: s/velho/novo/ 32

4.2 Só algumas linhas: -n, p e endereços 33

4.3 Apagar linhas: d 33

4.4 Alterar o arquivo, com cópia: -i.bak 33

4.5 Anonimizar dados: padrões e grupos 34

4.6 Caminhos com /: outro separador 34

4.7 Prática 35

Resumo 35

Aula 5. awk: colunas, condições, totais 36

5.1 As colunas de uma linha 37

5.2 Outro separador: -F 38

5.3 Só as linhas que interessam: condições 38

5.4 Somas e totais: END e printf 39

5.5 Totais por grupo: arrays 39

5.6 Quem é o robô? 40

5.7 Prática 41

Resumo 41

Aula 6. O script completo 42

6.1 Anatomia de um script 43

6.2 Argumentos e aspas 44

6.3 Deu certo? O status de saída 44

6.4 Encadear comandos: && e || 44

6.5 A saída de um comando dentro do texto: $( ) 45

6.6 Aritmética: $(( )) 45

6.7 O primeiro script útil: conta-erros.sh 45

6.8 Prática 46

Resumo 46

Aula 7. Decisões: if, test, case 47

7.1 Perguntas para o shell: test e [ ] 48

7.2 Texto ou número? A armadilha da comparação 49

7.3 if, elif, else 49

7.4 Um script que se confere: verifica.sh 49

7.5 Mensagens de erro em >&2 . . . 50
7.6 Muitas opções: case . . . 50
7.7 Prática . . . 52
Resumo . . . 52

Aula 8. Laços e renomeação em massa . . . 53

8.1 Repetir para cada item: for . . . 54
8.2 Pedaços de um nome . . . 55
8.3 Renomear em massa, com echo antes . . . 55
8.4 Linha por linha: while read . . . 56
8.5 Parar ou pular: break e continue . . . 56
8.6 Espaços nos nomes, e as aspas . . . 58
8.7 Prática . . . 59
Resumo . . . 59

Aula 9. Funções e menus . . . 60

9.1 Os seus próprios comandos: funções . . . 61
9.2 Variáveis só da função: local . . . 61
9.3 O que uma função devolve . . . 62
9.4 Perguntar ao usuário: read -p . . . 62
9.5 Projeto: loja.sh . . . 62
9.6 Prática . . . 64
Resumo . . . 65

Aula 10. Scripts seguros e depuração . . . 66

10.1 Três scripts com erros . . . 67
10.2 Erro 1: percorrer a saída do ls . . . 67
10.3 Erro 2: variáveis sem aspas . . . 68
10.4 Erro 3: uma variável vazia em um rm . . . 68
10.5 Redes de proteção: set -euo pipefail . . . 68
10.6 Veja o que o script faz: bash -x . . . 69
10.7 Limpeza garantida: mktemp e trap . . . 69
10.8 Um segundo leitor: ShellCheck . . . 70
10.9 Prática . . . 71
Resumo . . . 72

Aula 11. Automação: arquivos, limpeza, cron . . . 73

11.1 A data nos nomes de arquivo . . . 74
11.2 Arquivos compactados: tar -czf e -C . . . 74
11.3 Arquivos antigos e novos: find -mtime . . . 75
11.4 Manter só os N últimos: rotação . . . 76
11.5 Cópias sincronizadas: rsync . . . 76
11.6 Tarefas que rodam sozinhas: cron . . . 77
11.7 Prática . . . 79
Resumo . . . 79

Aula 12. Projeto final: backup.sh . . . 80
12.1 O que o script precisa fazer . . . 81
12.2 O começo: segurança e valores padrão . . . 82
12.3 Opções como as dos comandos de verdade: getopts . . . 82
12.4 Verificações antes de tudo . . . 83
12.5 O arquivo compactado e o log . . . 83
12.6 Rotação . . . 83
12.7 Testes . . . 84
12.8 Executado pelo cron . . . 85
12.9 Extensões, se você quiser mais . . . 86
Resumo . . . 86

Anexo A. Respostas dos desafios . . . 87

Anexo B. Índice de comandos . . . 92

Anexo C. Glossário . . . 94

Anexo D. O script do laboratório . . . 97

Agora os scripts trabalham por você . . . 101

Antes de começar

O que você já precisa saber

Este livro é a continuação de *Linux para Iniciantes*. Não é obrigatório ter lido o primeiro, mas você precisa estar à vontade com os comandos abaixo. Se algum for novidade, releia a aula correspondente do primeiro livro ou comece pelo `man`.

O que você faz	Os comandos
navegar entre pastas	`pwd`, `ls -la`, `cd`, caminhos relativos e absolutos
trabalhar com arquivos	`mkdir -p`, `touch`, `cp -r`, `mv`, `rm -r`
ler arquivos	`cat`, `less`, `head`, `tail`, `wc`
ligar comandos	`|`, `>`, `>>`, `2>`, `sort`, `uniq`
buscar	`grep` (simples), `find`
usar permissões	`sudo`, `chmod +x`, `ls -l` (ler as permissões)
editar um arquivo	`nano` (Ctrl+O salva, Ctrl+X sai)

O terminal

Você usa o mesmo terminal do primeiro livro:

Seu sistema	O que fazer
Windows	Ubuntu no WSL. Se ainda não tem: PowerShell **como administrador**, `wsl --install`, reinicie e escolha um nome de usuário e uma senha (anote os dois). Os comandos são digitados na janela do Ubuntu, não no PowerShell.
Mac	O aplicativo **Terminal**. O shell padrão é o zsh, mas os scripts deste livro começam com `#!/bin/bash` e rodam no bash. As diferenças que importam são indicadas no texto.
Linux	O aplicativo **Terminal**. Os exemplos foram executados no Debian 13.

> **Você precisa de sudo nas aulas 10 e 11**
> Nelas você instala dois programas pequenos: `shellcheck` (em todos os sistemas) e `rsync` (no Debian; o Ubuntu já tem). No Debian e no Ubuntu, com `sudo apt install …`; no Mac, com o Homebrew (`brew install shellcheck`). Se você não tem direitos de administrador, peça antes.

O laboratório de prática

As aulas trabalham com os arquivos de uma padaria online imaginária, a “Padaria Lab”: os logs do servidor, as vendas, os contatos dos clientes, um arquivo de configuração, fotos, um pequeno site. Eles são criados pelo script `preparar-laboratorio.sh`, na pasta `~/laboratorio-scripts`.

Este e-book acompanha um kit digital com esse script e com os scripts escritos nas aulas. Depois de baixar e extrair o kit, siga as instruções do arquivo LEIA-ME: o script fica na pasta `02-Laboratorio` e também está impresso no Anexo D, para consulta. Coloque-o no seu diretório pessoal do Linux e

execute:

```
lab@lpic1-deb:~$ bash preparar-laboratorio.sh
Laboratorio pronto: /home/lab/laboratorio-scripts
Arquivos: 51, pastas: 8
```

Pasta	O que tem dentro
`logs/`	`acesso.log`: 2.000 visitas ao site; `aplicacao.log`: 600 mensagens da aplicação (INFO, WARN, ERROR)
`dados/`	`vendas.csv`: 300 vendas de outubro; `contatos.txt`; `contas.txt`
`config/`	`loja.conf`: as configurações da loja
`fotos/`	27 fotos para renomear (três têm espaços no nome)
`documentos/`	relatórios antigos e documentos novos, para a limpeza automática
`projeto/`	um pequeno site: a origem dos backups das aulas 11 e 12
`scripts/`	três scripts com erros, para corrigir na aula 10

O laboratório sai idêntico a cada execução, então os números do livro (quantos erros, quantas vendas, que total) devem sair iguais para você. Se estragar algo, execute o script de novo e o laboratório volta ao estado inicial.

Os seus scripts ficam em ~/bin, não no laboratório
Quando você refaz o laboratório, tudo o que está em `~/laboratorio-scripts` é apagado, inclusive o que você escreveu ali. Por isso, na aula 1, você cria uma pasta `~/bin` para os seus scripts. Eles trabalham com os arquivos do laboratório, mas ficam a salvo fora dele.

O script foi feito para não apagar mais nada
• Ele só trabalha em `~/laboratorio-scripts`. Você pode escolher outro nome (`bash preparar-laboratorio.sh lab2` cria `~/lab2`), mas apenas um nome simples, direto no seu diretório pessoal: caminhos com `/` são recusados.
• Ele reconhece o próprio laboratório por um arquivo oculto, `.laboratorio-linux-na-pratica`. Só refaz uma pasta que tenha esse arquivo, ou uma pasta vazia. O laboratório do primeiro livro fica intocado.
• Se já existe uma pasta com o mesmo nome criada por você, o script para com uma mensagem de erro e não apaga nada.

LINUX NA PRÁTICA

Aula 1

Como o shell encontra os comandos

aula 1 de 12



Nesta aula

- Quem responde a um comando: type e which
- Onde o shell procura os programas: PATH
- Variáveis e aspas
- Variáveis para outros programas: export
- A pasta dos seus scripts: ~/bin
- Configurações que ficam: ~/.bashrc e alias

O que você vai saber no fim desta aula
• de onde vem um comando: parte do shell, um alias ou um programa no disco (`type`, `which`);
• o que é o `PATH` e por que aparece “command not found”;
• como criar e usar variáveis, e quando as aspas fazem diferença;
• como preparar a sua pasta `~/bin`, para que os seus scripts rodem de qualquer lugar;
• como manter uma configuração depois de fechar o terminal (`~/.bashrc`, `alias`).

Prepare o laboratório, se ainda não fez isso (veja “Antes de começar”):

```
lab@lpic1-deb:~$ bash preparar-laboratorio.sh
Laboratorio pronto: /home/lab/laboratorio-scripts
Arquivos: 51, pastas: 8
lab@lpic1-deb:~$ cd laboratorio-scripts
```

1.1 Quem responde a um comando

`type`: que tipo de comando é

No primeiro livro você digitou dezenas de comandos. Eles não são todos iguais. Alguns fazem parte do shell (o bash), outros são programas separados, guardados no disco. O `type` diz qual é qual:

```
lab@lpic1-deb:~/laboratorio-scripts$ type cd
cd is a shell builtin
lab@lpic1-deb:~/laboratorio-scripts$ type ls
ls is aliased to `ls --color=auto'
lab@lpic1-deb:~/laboratorio-scripts$ type grep
grep is /usr/bin/grep
```

A resposta	O que significa	Onde buscar ajuda
shell builtin	um comando embutido no bash; não existe como arquivo separado	`help cd`
aliased to	um alias, um atalho (você vai criar um na seção 1.6)	o comando por trás do alias
`/usr/bin/grep`	um programa, ou seja, um arquivo no disco	`man grep`

`which`: onde está o programa

O `which` só procura programas no disco e mostra o caminho deles:

```
lab@lpic1-deb:~/laboratorio-scripts$ which ls grep
/usr/bin/ls
/usr/bin/grep
```

Para o `cd`, o `which` não mostra nada: não existe um arquivo `cd`. Por isso o `type` é mais útil quando você quer saber o que está executando de verdade.

1.2 PATH: onde o shell procura os programas

Quando você digita `grep`, o bash não vasculha o disco inteiro. Ele procura apenas nas pastas listadas na variável `PATH`, em ordem, e executa o primeiro `grep` que encontrar:

```
lab@lpic1-deb:~/laboratorio-scripts$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

As pastas são separadas por `:`. Se um programa não está em nenhuma delas, aparece “command not found”, mesmo que o arquivo exista. Você vai ver isso na seção 1.5, com o seu primeiro script.

A sua lista pode ser mais longa (no WSL ela inclui também pastas do Windows, parecidas com `/mnt/c/...`). O que importa é a ideia, não a lista exata.

1.3 Variáveis

Uma variável é um nome para um valor. Você a cria com `NOME=valor` e a usa com um `$` na frente:

```
lab@lpic1-deb:~/laboratorio-scripts$ PESSOA=Ana
lab@lpic1-deb:~/laboratorio-scripts$ echo $PESSOA
Ana
lab@lpic1-deb:~/laboratorio-scripts$ echo "Ola, $PESSOA!"
Ola, Ana!
lab@lpic1-deb:~/laboratorio-scripts$ echo 'Ola, $PESSOA!'
Ola, $PESSOA!
```

Aspas	O que acontece com $PESSOA	Quando usar
duplas "..."	é substituída pelo valor	quase sempre, em volta de variáveis
simples '...'	fica exatamente como você digitou	quando você quer o texto como está (um alias, por exemplo)

Nos exemplos, as mensagens saem sem acento (`Ola`, não “Olá”). Assim elas aparecem certas em qualquer terminal. Os scripts do livro e do kit seguem a mesma regra.

> **Sem espaços em volta do =**
> Com espaços, o bash acha que PESSOA é um comando e que = e Ana são os argumentos dele:

```
lab@lpic1-deb:~/laboratorio-scripts$ PESSOA = Ana
-bash: PESSOA: command not found
```

Algumas variáveis já existem, definidas pelo sistema quando você entrou:

```
lab@lpic1-deb:~/laboratorio-scripts$ echo $HOME
/home/lab
lab@lpic1-deb:~/laboratorio-scripts$ echo $USER
lab
lab@lpic1-deb:~/laboratorio-scripts$ echo $SHELL
/bin/bash
```

`HOME` é o seu diretório pessoal (o mesmo que você escreve como `~`, para abreviar), `USER` é o seu nome de usuário e `SHELL` é o shell com que você entra. No Mac, `SHELL` é `/bin/zsh`.

Escolha nomes que o sistema ainda não usa. Em inglês é fácil tropeçar: no WSL, por exemplo, `NAME` já existe e guarda o nome do computador. Nomes em português, como `PESSOA` ou `CIDADE`, ficam livres.

1.4 Variáveis para outros programas: export

`export`

Uma variável criada como na seção 1.3 vive só no seu shell. Os programas que você inicia a partir dele não a enxergam. Para testar, o `bash -c` inicia um shell novo que executa um único comando:

```
lab@lpic1-deb:~/laboratorio-scripts$ bash -c 'echo "Novo shell: [$PESSOA]"'
Novo shell: []
```

Os colchetes estão vazios: o shell novo não conhece `PESSOA`. O `export` transforma a variável em **variável de ambiente**, que todo programa iniciado a partir de agora recebe:

```
lab@lpic1-deb:~/laboratorio-scripts$ export PESSOA
lab@lpic1-deb:~/laboratorio-scripts$ bash -c 'echo "Novo shell: [$PESSOA]"'
Novo shell: [Ana]
```

Isso importa porque os seus scripts, a partir da aula 6, também são programas iniciados pelo shell: eles só enxergam as variáveis exportadas.

`printenv` e `env`

O `printenv` mostra as variáveis de ambiente que você pede, e o `env` mostra todas (muitas, por isso filtramos com o `grep`):

```
lab@lpic1-deb:~/laboratorio-scripts$ printenv PESSOA
Ana
lab@lpic1-deb:~/laboratorio-scripts$ printenv HOME USER
/home/lab
lab
lab@lpic1-deb:~/laboratorio-scripts$ env | grep -E '^(PESSOA|HOME|SHELL)='
SHELL=/bin/bash
PESSOA=Ana
HOME=/home/lab
```

O padrão `^(PESSOA|HOME|SHELL)=` quer dizer “linhas que começam com PESSOA=, HOME= ou SHELL=”. Padrões assim se chamam expressões regulares; você vai aprendê-los na aula 2.

As variáveis que você cria somem quando você fecha o terminal. Nas seções 1.5 e 1.6 você vai ver como fazer uma configuração ficar.

1.5 A pasta dos seus scripts: ~/bin

Um script é um arquivo de texto com comandos dentro. Você vai criar o primeiro agora, bem curto, para ver como o shell o encontra. O que cada linha significa, você aprende na aula 6. Primeiro confira se o nome está livre e depois crie a pasta `~/bin`:

```
lab@lpic1-deb:~$ type ola
-bash: type: ola: not found
lab@lpic1-deb:~$ mkdir ~/bin
lab@lpic1-deb:~$ nano ~/bin/ola
# digite as duas linhas abaixo, depois Ctrl+O, Enter, Ctrl+X
lab@lpic1-deb:~$ cat ~/bin/ola
#!/bin/bash
echo "Ola! Rodando a partir de ~/bin."
lab@lpic1-deb:~$ chmod +x ~/bin/ola
```

O `chmod +x` (do primeiro livro) o torna executável. Agora execute:

```
lab@lpic1-deb:~$ ola
-bash: ola: command not found
lab@lpic1-deb:~$ ~/bin/ola
Ola! Rodando a partir de ~/bin.
```

Com o caminho completo funciona. Só com o nome, não: `~/bin` não está no `PATH`. Acrescente-a no começo:

```
lab@lpic1-deb:~$ export PATH="$HOME/bin:$PATH"
lab@lpic1-deb:~$ echo $PATH
/home/lab/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
lab@lpic1-deb:~$ ola
Ola! Rodando a partir de ~/bin.
lab@lpic1-deb:~$ type ola
ola is hashed (/home/lab/bin/ola)
lab@lpic1-deb:~$ cd /tmp
lab@lpic1-deb:/tmp$ ola
Ola! Rodando a partir de ~/bin.
```

O `PATH` novo é o `$PATH` antigo com `/home/lab/bin:` colado na frente. Agora o `ola` funciona de qualquer pasta, como qualquer outro comando. “Hashed” quer dizer que o bash guardou onde o encontrou, para não precisar procurar de novo na próxima vez.

Para a configuração ficar

O `export` só vale até você fechar o terminal. No Debian e no Ubuntu você não precisa fazer mais nada: o arquivo `~/.profile`, lido toda vez que você entra, acrescenta `~/bin` ao `PATH` sozinho, se a pasta existir:

```
lab@lpic1-deb:~$ grep -A3 'private bin' ~/.profile
# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
# (seguido de um bloco parecido para ~/.local/bin)
```

Feche o terminal, abra de novo e confira:

```
lab@lpic1-deb:~$ echo $PATH
/home/lab/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
lab@lpic1-deb:~$ ola
Ola! Rodando a partir de ~/bin.
lab@lpic1-deb:~$ type ola
ola is hashed (/home/lab/bin/ola)
```

No Mac e no WSL
Mac: o zsh não lê o `~/.profile`. Acrescente você mesmo a linha `export PATH="$HOME/bin:$PATH"` no fim do `~/.zshrc` (com `nano ~/.zshrc`) e abra um terminal novo.
WSL: igual ao Debian. Se o `ola` ainda não funcionar depois de reabrir o terminal, desligue o Ubuntu por completo: no PowerShell, `wsl --shutdown`, e depois inicie de novo.

Não dê aos seus scripts nomes de comandos que já existem (`test`, `ls`, `cp`). Como `~/bin` vem primeiro no `PATH`, o seu script rodaria no lugar do comando verdadeiro. Confira antes com `type nome`, como você fez acima: se ele disser "not found", o nome está livre.

1.6 Configurações que ficam: ~/.bashrc e alias

`alias` e `source`

Um alias é um atalho para um comando (você os conheceu no primeiro livro). Para que ele fique, você o escreve no `~/.bashrc`, o arquivo que o bash lê toda vez que um terminal novo abre. Vamos criar um que leva você direto ao laboratório, `lab`. Confira antes se o nome está livre:

```
lab@lpic1-deb:~$ type lab
-bash: type: lab: not found
lab@lpic1-deb:~$ echo "alias lab='cd ~/laboratorio-scripts'" >> ~/.bashrc
lab@lpic1-deb:~$ tail -1 ~/.bashrc
alias lab='cd ~/laboratorio-scripts'
lab@lpic1-deb:~$ source ~/.bashrc
lab@lpic1-deb:~$ type lab
lab is aliased to `cd ~/laboratorio-scripts'
lab@lpic1-deb:~$ lab
lab@lpic1-deb:~/laboratorio-scripts$ pwd
/home/lab/laboratorio-scripts
```

Dois sinais >, não um
O `>>` acrescenta no fim do arquivo. Com um `>` só, o `~/.bashrc` seria substituído por essa única linha e você perderia todas as configurações dele. Se quiser ir pelo seguro, abra o arquivo com `nano ~/.bashrc` e digite a linha no fim.

O `source ~/.bashrc` relê o arquivo no terminal atual, para você não precisar abrir outro. O `alias` sem nada depois lista todos os seus aliases:

```
lab@lpic1-deb:~/laboratorio-scripts$ alias
alias lab='cd ~/laboratorio-scripts'
alias ls='ls --color=auto'
```

Por que não `ll`, como em muitos tutoriais? No Ubuntu, `ll` já existe (`ls -alF`), e no Debian ele está no `~/.bashrc`, mas desativado com `#`. Um nome novo, conferido com `type`, não conflita com nada. No Mac, os aliases vão no `~/.zshrc`.

O arquivo	Quando é lido	O que vai nele
`~/.profile`	quando você entra no sistema	o `PATH` e outras variáveis de ambiente
`~/.bashrc`	a cada terminal novo com bash	aliases, configurações do shell
`~/.zshrc`	a cada terminal novo com zsh (Mac)	no Mac, o PATH e os aliases

1.7 Prática

1. Use o `type` para descobrir que tipo de comando são `echo`, `pwd`, `nano` e `history`. Quais têm `man` e também `help`?
2. Crie uma variável `CIDADE` com a sua cidade e mostre "Eu moro em …", uma vez com aspas duplas e outra com aspas simples. Explique a diferença.
3. Confira se um shell novo enxerga `CIDADE`, antes e depois de `export CIDADE`.
4. Escreva um script chamado `hoje` em `~/bin`, com as linhas `#!/bin/bash` e `date`. Torne-o executável e execute-o a partir de `~/laboratorio-scripts`.

Desafio
Escreva um script chamado `ondeestou` em `~/bin` que mostre a pasta em que você está e quantos itens ela tem. Execute-o em três pastas diferentes, só pelo nome. Dica: na aula 6 você vai conhecer o `$(comando)`, que coloca a saída de um comando no meio de um texto; experimente `echo "Estou em $(pwd)"`.

Atividade
1. Acrescente o alias `lab` ao `~/.bashrc` (se não fez isso junto com a aula), abra um terminal novo e confira com `type lab`.
2. Crie outro alias, `erros`, que mostre as linhas com ERROR de `~/laboratorio-scripts/logs/aplicacao.log`. Quantas são? (Conte com `erros | wc -l`.)

Resumo

Comando	O que faz
`type nome`	diz se é embutido, alias ou programa (e onde)
`which nome`	o caminho do programa no disco
`echo $PATH`	as pastas onde os programas são procurados, separadas por :
`NOME=valor`	cria uma variável (sem espaços em volta do =)
`"$NOME"` / `'$NOME'`	com o valor / o texto como está
`export NOME`	passa a variável para os programas iniciados pelo shell
`printenv NOME`, `env`	mostram variáveis de ambiente
`export PATH="$HOME/bin:$PATH"`	põe ~/bin no começo, até você fechar o terminal
`~/.profile`, `~/.bashrc`	configurações lidas na entrada / a cada terminal novo
`alias nome='cmd'`, `source f`	um atalho / relê o arquivo de configuração