Capitulo 5. Criação, visualização e edição de arquivos de texto

Resumo

Meta	Criar, visualizar e editar arquivos de texto na saida do comando ou em um editor de texto
Objetivos	- Salvar a saida ou os erros em um arquivo com redirecionamento de shell e processar a saida do comando por meio de varios programas de linha de comando com pipes. - Criar e editar arquivos de texto a a partir da linha de comando com o editor vim - Definir variaveis do shell para executar comandos, editar scripts de inicialização Bash para definir variaveis shell de ambiente a fim de modificar o comportamento do shell e os programas executados a partir dele.
Seções	- Redirecionamento da saida para um arquivo ou programa (e teste) - Edição de arquivos de texto a partir do prompt do shell (e execicio orientado) - Alteração do ambiente do shell (e exercicio orientado)
Laboratorio	Criação, visualização e edição de arquivos de texto

Entrada padrão, saída padrão e erro padrão

Um programa ou processo em execução le a entrada e grava a saida. Quando um comando é executado no prompt ele le a entrada como o teclado e envia a saida para a tela do terminal.

Um processo usa varios canais chamados de DESCRITORES de ARQUIVOS para obter a entrada e envia a saida.

Todo processo começa com pelo menos 3 descritores de arquivos.

- A entrada padrão (canal 0) le a entrada do teclado
- A saida padrão (canal 1) envia a saida normal para o terminal

- O erro padrão (canal 2) envia mensagens de erro para o terminal

Se um programa abrir conexões separadas para outros arquivos, ele podera usar descritores de arquivo de numeração mais alta.

Figura de Canais de Entrada e Saída (descritores de arquivos) dos processos

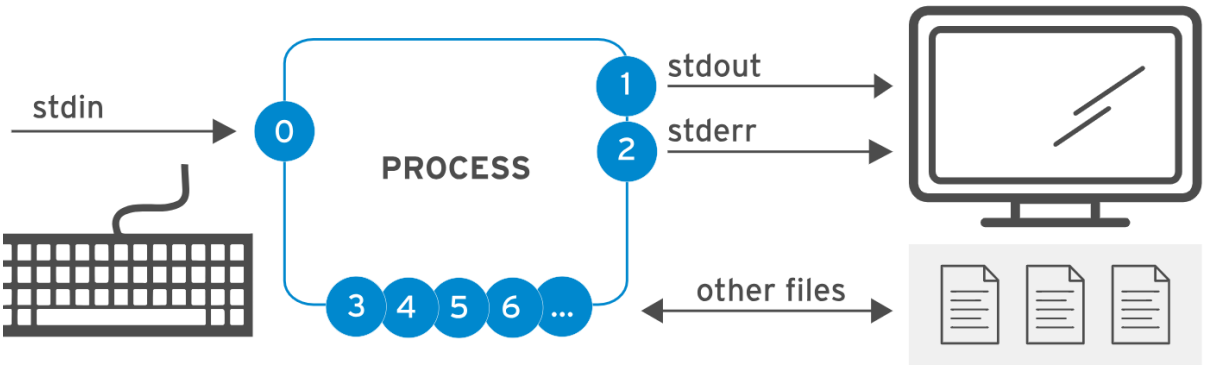


Tabela sobre Canais (descritores do arquivo)

Numero	Nome do canal	Descrição	Conexão padão	Uso
0	stdin	Entrada padão	Teclado	Somente leitura
1	stdout	Saída padão	Terminal	Somente gravação
2	stderr	Erro padão	Terminal	Somente gravação
Mais de 3	filename	Outros arquivos	Nenhum	Leitura, gravação ou ambos

Redirecionamento da saída para um arquivo

O redirecionamento Entrada/Saida muda como o processo obtem sua entrada ou saida.

Em vez de obter entrada do teclado ou enviar saida de erros para o terminal,

O processo pode ler ou gravar em arquivos.

Com o redirecionamento, pode salvar as mensagens em um arquivo ao inves de aparecer no terminal.

O redirecionamento pode ser usado para redirecionar a saida e os erros, para que nao sejam exibidos no terminal e nem salvos.

STDOUT

Pode redirecionar um processo stdout para impedir que a saida do processo apareça no terminal.

CASO 1 = Porem se redirecionar o `stdout` para um aruqivo e o arquivo não existir, o arquivo sera criado.

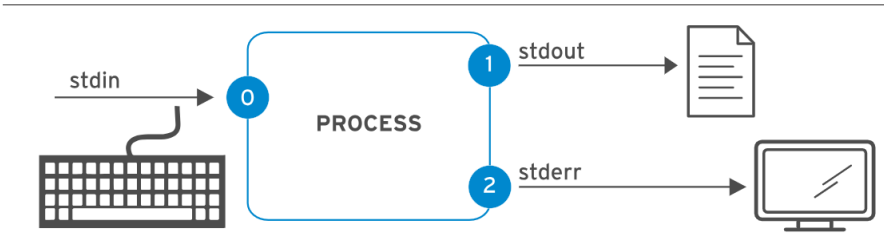
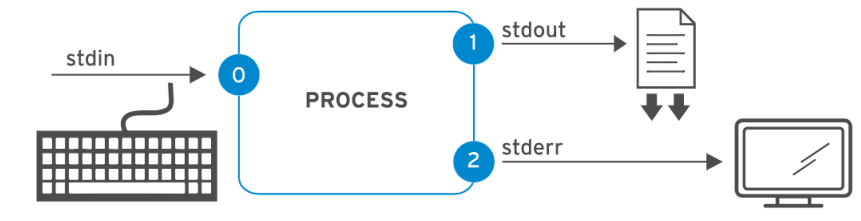
Caso 2 = Se o arquivo existir e o redirecionamento não for o anexado ao arquivo, o redirecionamento substituira o conteudo do arquivo

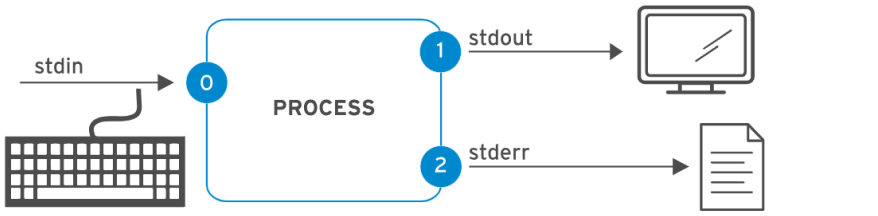
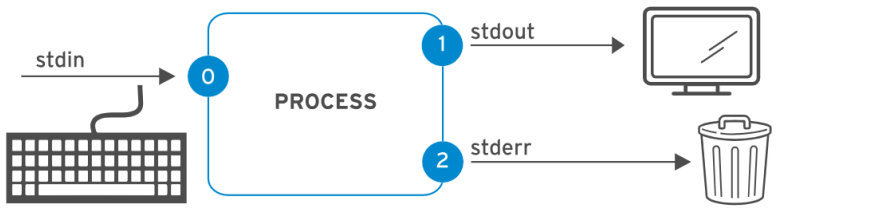
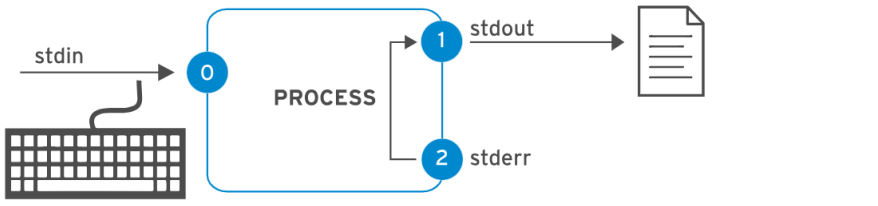
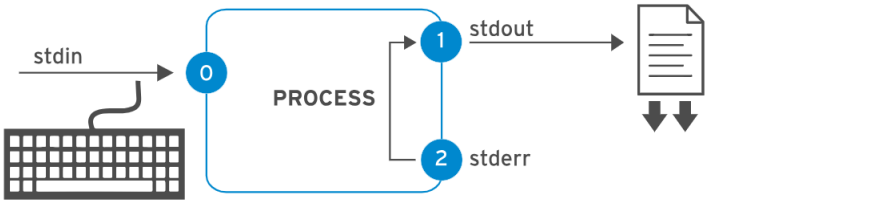
Como descarta a saida de um processo?

Pode redirecionar para o arquivo especial `/dev/null` vazio que descarta a saida do canal que é redirecionada para ele.

OBS: Conforme exibido na tabela a seguir, *apenas* `stdout` redirecionar não impede a exibição de mensagens de erro `stderr` no terminal.

Tabela Operadores de redirecionamento de saída

Uso	Explicação	Ajuda visual
>file	Redirecionar `stdout` para sobrescrever um arquivo	
>> file	Redirecionar `stdout` para anexar um arquivo.	

2> file	Redirecionar `stderr` para sobrescrever um arquivo	
2> /dev/null	Descartar mensagens de erro `stderr` redirecionando para `/dev/null`	
> file *2>&1* *&> file*	Redirecionar `stdout` e `stderr` para sobrescrever o mesmo arquivo	
>> file *2>&1* *&>> file*	Redirecionar `stdout` e `stderr` para anexar ao mesmo arquivo	

Importante

A ordem das operações de redirecionamento é importante.

A sequência a seguir redireciona a saída padrão para o arquivo `output.log` e, depois, redireciona as mensagens de erro padrão para o mesmo local da saída padrão (`output.log`).

```
> output.log 2>&1
```

A sequência a seguir faz o redirecionamento na ordem oposta. Essa sequência redireciona mensagens de erro padrão ao local padrão da saída padrão (a janela de terminal, assim, não há mudanças) e *depois* redireciona apenas a saída padrão para o arquivo `output.log`.

```
2>&1 > output.log
```

Devido a isso, alguns preferem usar os operadores de fusão de redirecionamento:

- &> output.log **em vez de** > output.log 2>&1
- &>> output.log em vez de >> output.log 2>&1

Exemplos de redirecionamento de saída

01 - Salve um carimbo de data e hora no arquivo `/tmp/saved-timestamp` para referência futura.

```
[user@host ~]$ date > /tmp/saved-timestamp
```

02 - Copie as últimas 100 linhas do arquivo `/var/log/secure` para o arquivo `/tmp/last-100-log-secure`.

```
tail -n 100 /var/log/secure > /tmp/last-100-log-secure
```

03 - Concatene todos os quatro arquivos `step` em um no diretório `tmp`.

```
cat step1 step2 step3 step4 > /tmp/all-four-steps-in-one
```

04 - Liste os nomes de arquivos regulares e ocultos do diretório pessoal e salve a saída no arquivo `my-file-names`.

```
ls -a > my-file-names
```

05 - Anexe uma linha ao arquivo `/tmp/many-lines-of-information` existente.

```
echo “new line information” >> /tmp/many-lines-of-information
```

Os próximos comandos geram mensagens de erro, porque alguns diretórios do sistema estão inacessíveis para usuários normais. Observe o redirecionamento da mensagem de erro.

06 - Redirecione os erros do comando `find` para o arquivo `/tmp/errors` quando visualizar a saída normal do comando no terminal.

```
find /etc -name passwd 2> /tmp/erros
```

07 - Salve a saída do processo no arquivo `/tmp/output` e as mensagens de erro no arquivo `/tmp/errors`.

```
find /etc -name passwd > /tmp/output 2> /tmp/errors
```

08 - Salve a saída do processo no arquivo `/tmp/output` e descarte as mensagens de erro.

```
find /etc -name passwd > /tmp/output 2> /dev/null
```

09 - Armazene em conjunto a saída e os erros gerados no arquivo `/tmp/all-message-output`.

```
find /etc -name passwd &> /tmp/all-message-output
```

10 - Anexe a saída e os erros gerados ao arquivo `/tmp/all-message-output`.

```
find /etc -name passwd >> /tmp/all-message-output 2>&1
```

Construção de pipelines

Um *pipeline* é uma sequência de um ou mais comandos separados pelo caractere de barra vertical (`|`). Um pipeline conecta a saída padrão do primeiro comando à entrada padrão do comando seguinte.

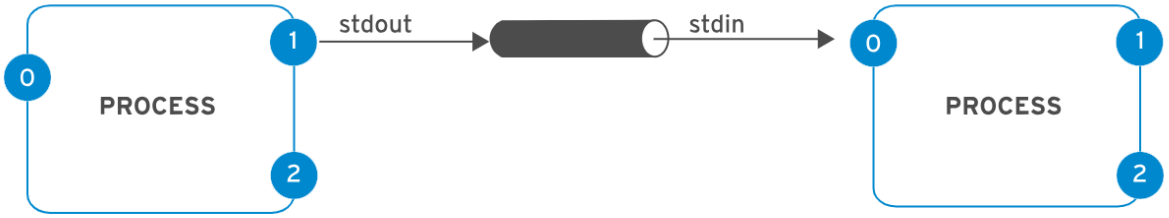


Figura 5.8: Piping de E/S de processos

Use pipelines para manipular e formatar a saída de um processo por outros processos antes de sua saída para o terminal. Imagine que os dados "fluem" pelo pipeline de um processo a outro, sendo levemente alterados a cada comando no pipeline em que passam.

- Pipelines e redirecionamento de E/S manipulam a saída padrão e a entrada padrão.
- Os pipelines enviam a saída padrão de um processo para a entrada padrão de outro processo.
- O redirecionamento envia a saída padrão para arquivos ou recebe deles a entrada padrão.

Exemplos de pipeline

01 - Redirecione a saída do comando `ls` para o comando `less` para exibição no terminal, uma tela por vez.

```
ls -l /usr/bin | less
```

02 - Redirecione a saída do comando `ls` para o comando `wc -l`, que conta o número de linhas recebidas de `ls` e o imprime o valor no terminal.

```
ls | wc -l
```

03 - Redirecione a saída do comando `ls -t` para o comando `head` para exibir as primeiras 10 linhas, com o resultado final redirecionado para o arquivo `/tmp/first-ten-changed-files`.

```
ls -t | head -n 10 > /tmp/firts-ten-changed-files
```

Pipelines, redirecionamento e anexação a um arquivo

Quando configura um redirecionamento a um pipeline:

- o shell configura todo o pipeline primeiro
- e depois redireciona a entrada/saida

Se você usar o redirecionamento de saída no *meio* de um pipeline, a saída irá para o arquivo e não para o próximo comando no pipeline.

No próximo exemplo, a saída do comando `ls` irá para o arquivo `/tmp/saved-output` e o comando `less` não exibirá nada no terminal:

```
ls > /tmp/saved-output | less
```

Comando TEE

Em um pipeline, tee copia sua entrada padrão para a saida padrao e redireciona a saída padrão para os arquivos fornecidos como argumentos do comando.

Se você imaginar os dados como água fluindo através de uma tubulação, o `tee` pode ser visualizado como uma junção "T" no tubo, que direciona a saída em duas direções.

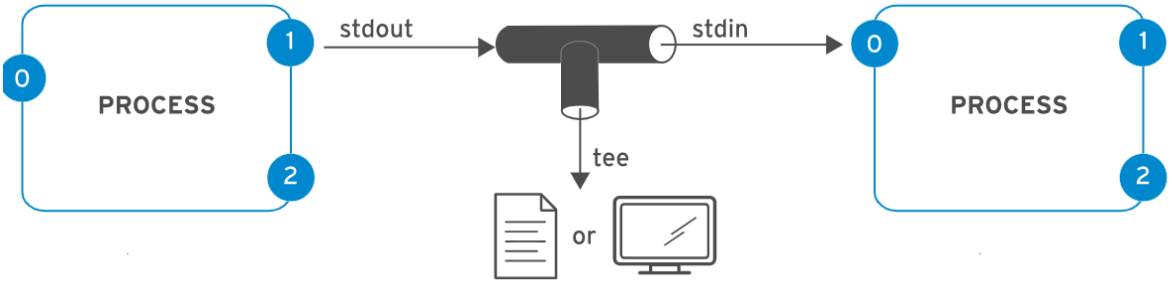


Figura 5.9: Piping de E/S do processo com tee

Exemplos de pipeline com o comando tee

1- O próximo exemplo redireciona a saída do comando `ls` para o arquivo `/tmp/saved-output` e passa-o ao comando `less` para exibição no terminal, uma tela por vez.

```
ls -l | tee /tmp/saved-output | less
```

2- Se você usar o comando `tee` no final de um pipeline, o terminal mostrará a saída dos comandos no pipeline e a salvará em um arquivo ao mesmo tempo.

```
ls -t | head -n 10 | tee /tmp/ten-last-changed-files
```

3-Use a opção
`-a` do comando `tee` para anexar o conteúdo a um arquivo em vez de substituí-lo.

```
ls -l | tee -a /tmp/append-files
```

Você pode redirecionar o erro padrão através de um pipeline, mas não pode usar os operadores de fusão de redirecionamento (
`&>` e `&>>`).

Este exemplo é a maneira correta de redirecionar tanto a saída padrão quanto o erro padrão através de um pipeline:

```
find / -name passwd 2>&1 | less
```

Perguntas:

1- Qual operador de redirecionamento de saída exibe a saída para um terminal e descarta todas as mensagens de erro?

2> /dev/null

2- Qual operador de redirecionamento de saída envia a saída para um arquivo e os erros para um arquivo diferente?

>file 2> file2

3-Qual operador de redirecionamento de saída envia a saída e os erros para um arquivo, e cria ou substitui seu conteúdo?

&> file

4- Qual operador de redirecionamento de saída envia saída e erros para o mesmo arquivo e preserva o conteúdo do arquivo se ele existir?

>>file 2>&1

5- Qual operador de redirecionamento de saída descarta todas as mensagens normalmente enviadas para o terminal?

&> /dev/null

6- Qual operador de redirecionamento de saída envia a saída do comando para a tela e para um arquivo ao mesmo tempo?

| tee file

7- Qual operador de redirecionamento de saída salva a saída em um arquivo e descarta todas as mensagens de erro?

>file2 > /dev/null

Edição de arquivos com o Vim

Introdução ao Vim

Você pode instalar o editor Vim no Red Hat Enterprise Linux usando um de dois pacotes. Esses dois pacotes fornecem diferentes recursos e comandos do Vim para edição de arquivos baseados em texto.

Com o pacote `vim-minimal`, você pode instalar o editor `vi` com recursos principais. Essa é uma instalação muito leve que inclui apenas o conjunto de recursos principais e o comando `vi`. Você pode abrir um arquivo para edição usando o comando `vi`:

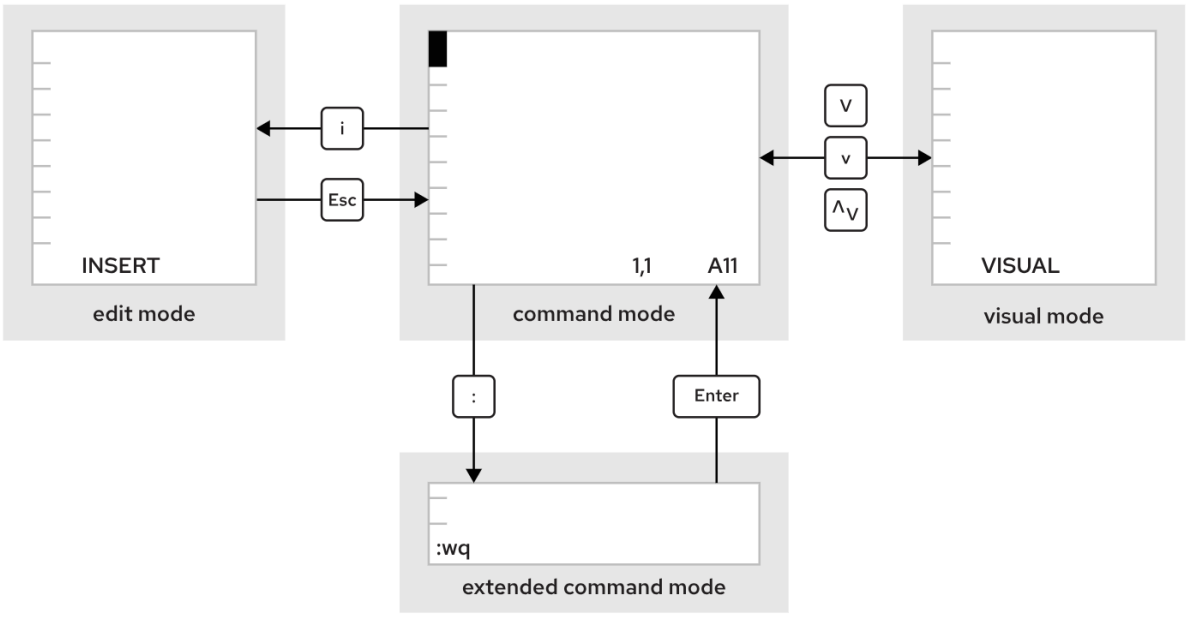
Como alternativa, você pode usar o pacote `vim-enhanced` para instalar o editor Vim. Ele fornece um conjunto mais abrangente de recursos, um sistema on-line de ajuda e um programa de tutorial. Para iniciar o Vim nesse modo avançado, use o comando `vim`:

Modos de operação do Vim

O editor Vim oferece vários modos de operação:

- *modo de comando*

- *modo de comando estendido*
- *modo de edição* e *modo visual*



Quando você abre o Vim pela primeira vez, ele é iniciado em *Modo de comando*

Que é usado para navegação, operações de cortar e colar e modificação de texto. Pressionar a tecla necessária acessa funções de edição específicas:

- O atalho de teclado **i** entra no *modo de edição*, em que todo o texto digitado se torna conteúdo do arquivo. Pressionar **Esc** retorna para o modo de comando.
- O atalho de teclado **v** entra no *modo visual*, em que vários caracteres podem ser selecionados para manipulação de texto. Use **Shift**+**V** para seleção de várias linhas e **Ctrl**+**V** para seleção em bloco. Para sair do modo visual, use os atalhos de teclado **v**, **Shift**+**V** ou **Ctrl**+**V.**
- O atalho de teclado **:** inicia o *modo de comando estendido* para tarefas como gravar o arquivo (para salvá-lo) ou sair do editor do Vim.

O fluxo de trabalho mínimo e básico do Vim

O Vim tem atalhos de teclado eficientes e coordenados para tarefas de edição avançadas. Embora se tornem muito benéficos com a prática, os recursos do Vim podem sobrecarregar novos usuários.

- A tecla **u** desfaz a edição mais recente
- A tecla **x** exclui um único caractere.
- O comando `:w` grava (salva) o arquivo e permanece no modo de comando para continuar editando
- O comando `:wq` grava (salva) o arquivo e sai do Vim.
- O comando `:q!` sai do Vim e descarta todas as alterações no arquivo desde a última gravação.

Reorganização do texto existente

No Vim, você pode *retirar* e *colocar* (copiar e colar) usando os caracteres de comando **y** e **p**. Posicione o cursor no primeiro caractere a ser selecionado e entre no modo visual. Use as teclas de seta para expandir a seleção visual. Quando estiver pronto, pressione **y** para *copiar* (yank) a seleção para a memória. Posicione o cursor no local novo e pressione **p** para *colocar* a seleção no cursor.

Modo visual no Vim

O modo visual é útil para destacar e manipular texto em diferentes linhas e colunas. Você pode inserir vários modos visuais no Vim usando as combinações de teclas a seguir:

- Modo de caractere: **v**
- Modo de linha: **Shift+v**
- Modo de bloqueio: **Ctrl+v**

O modo de caractere destaca frases em um bloco de texto. A palavra `VISUAL` é exibida na parte inferior da tela. Pressione **v** para entrar no modo de caracteres visuais. **Shift**+**v** entra no modo de linha. `VISUAL LINE` é exibido na parte inferior da tela.

O modo de bloqueio visual é perfeito para manipular arquivos de dados. Pressione a tecla **Ctrl**+**v** para entrar no bloqueio visual do cursor. `VISUAL BLOCK` é exibido na parte inferior da tela. Use as teclas de seta para realçar a seção a ser alterada.

Arquivos de configuração do Vim

Os arquivos de configuração `/etc/vimrc` e `~/.vimrc` alteram o comportamento do editor `vim` para todo o sistema ou para um usuário específico, respectivamente.

Nesses arquivos de configuração, você pode especificar o comportamento, como o espaçamento de tabulações padrão, realce de sintaxe, esquemas de cores e muito mais.

Considere o arquivo `~/.vimrc` a seguir, que define a parada de tabulação padrão (denotada pelos caracteres `ts`) para dois espaços ao editar arquivos YAML. O arquivo também inclui o parâmetro `set number` para exibir números de linha durante a edição de todos os arquivos.

```
[user@host ~]$cat ~/.vimrc
autocmd FileType yaml setlocal ts=2
set number
```

Alteração do ambiente do shell

Uso da variável do shell

Com o shell Bash, você pode definir *variáveis de shell* para ajudar a executar comandos ou modificar o comportamento do shell. Você também pode exportar variáveis de shell como variáveis de ambiente, que são automaticamente copiadas para programas executados a partir desse shell.

Você pode usar variáveis para facilitar a execução de um comando com um argumento longo ou para aplicar uma configuração comum aos comandos executados a partir desse shell.

As variáveis do shell são exclusivas de uma sessão de shell específica. Se você tiver duas janelas de terminal abertas ou duas sessões de login independentes para o mesmo servidor remoto, você está executando dois shells. Cada shell tem seu próprio conjunto de valores para suas variáveis do shell.

Atribuição de valores a variáveis

Atribua um valor a uma variável de shell com a seguinte sintaxe:

`VARIABLENAME=value`

Os nomes das variáveis podem conter letras maiúsculas ou minúsculas, dígitos e o caractere de sublinhado (`_`). Por exemplo, os seguintes comandos definem as variáveis de shell:

```
[user@host ~]$ COUNT=40
[user@host ~]$ first_name=John
[user@host ~]$ file1=/tmp/abc
[user@host ~]$ _ID=RH123
```

Você pode usar o comando `set` para listar todas as variáveis de shell que estão atualmente configuradas. (Ele também lista todas as funções de shell, que você pode ignorar.).

Para facilitar a leitura, você pode enviar a saída para o comando `less` e visualizar uma página por vez.

```
[user@host ~]$ set | less
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:expand_alias
:force_fignore:globasciiranges:histappend:interactive_comment
omp:promptvars:sourcepath
BASHRCSOURCED=Y
...output omitted...
```

Recuperação de valores com expansão de variável

Você pode usar a *expansão de variável* para se referir ao valor de uma variável que você definiu.

Para usar a expansão de variável, use um símbolo de dólar (`$`) antes do nome da variável. Nos exemplos a seguir, a expansão de variável ocorre primeiro e, depois, o comando `echo` imprime o restante da linha de comando inserida.

Por exemplo, o seguinte comando define a variáveis `COUNT` como `40`:

```
[user@host ~]$ COUNT=40
```

Se você inserir o comando `echo COUNT`, a string `COUNT` será impressa.

```
[user@host ~]$ echo COUNT
COUNT
```

Você também pode ser usar uma variável para se referir a um nome de arquivo longo para vários comandos.

```
[user@host ~]$ file1=/tmp/tmp.z9pXW0HqcC
[user@host ~]$ ls -l $file1
-rw-------. 1 student student 1452 Jan 22 14:39 /tmp/tmp.z9pX
[user@host ~]$ rm $file1
[user@host ~]$ ls -l $file1
total 0
```

Importante

Você sempre pode usar chaves na expansão de variáveis, embora muitas vezes elas sejam desnecessárias.

No exemplo a seguir, o comando `echo` tenta expandir a variável inexistente `COUNTx`, mas não retorna nada. O comando também não relata erros.

```
[user@host ~]$ echo Repeat $COUNTx
Repeat
```

Se algum caractere estiver ao lado de um nome de variável, delimite o nome da variável com chaves. No exemplo a seguir, o comando `echo` agora expande a variável `COUNT`.

```
[user@host ~]$ echo Repeat ${COUNT}x
Repeat 40x
```

Configuração do Bash com variáveis de Shell

Algumas variáveis de shell são definidas quando o Bash é iniciado. Você pode modificá-las para ajustar o comportamento do shell.

Por exemplo, as variáveis de shell `HISTFILE`, `HISTFILESIZE` e `HISTTIMEFORMAT` afetam o histórico do shell e o comando `history`. A variável `HISTFILE` especifica em qual arquivo salvar o histórico do shell, e o padrão é o arquivo `~/.bash_history`. A variável `HISTFILESIZE` especifica quantos comandos devem ser salvos naquele arquivo do histórico. A variável `HISTTIMEFORMAT` define o formato de carimbo de data e hora para cada comando no histórico. Essa variável não existe por padrão.

```
[user@host ~]$ history
...output omitted...
    6  ls /etc
    7  uptime
    8  ls -l
    9  date
   10  history
[user@host ~]$ HISTTIMEFORMAT="%F %T "
[user@host ~]$ history
...output omitted...
```

```
     6  2022-05-03 04:58:11 ls /etc
     7  2022-05-03 04:58:13 uptime
     8  2022-05-03 04:58:15 ls -l
     9  2022-05-03 04:58:16 date
    10  2022-05-03 04:58:18 history
    11  2022-05-03 04:59:10 HISTTIMEFORMAT="%F %T "
    12  2022-05-03 04:59:12 history
```

Outro exemplo é a variável `PS1`, que controla a aparência do prompt de shell. Se você alterar esse valor, ele alterará a aparência do prompt de shell. Várias expansões de caracteres especiais compatíveis com o prompt são listadas na seção "SOLICITAÇÃO" da página do man `bash`(1).

```
[user@host ~]$ PS1="bash\$ "
bash$ PS1="[\u@\h \W]\$ "
[user@host ~]$
```

Como o valor que a variável `PS1` define é um prompt, a Red Hat recomenda terminar o prompt com um espaço à direita. Além disso, sempre que o valor de uma variável contiver alguma forma de espaço, incluindo um espaço, um recuo ou uma quebra de linha, o valor deve estar entre aspas, simples ou duplas. Resultados inesperados podem ocorrer se as aspas forem omitidas. A variável anterior `PS1` está em conformidade com a recomendação de espaço à direita e a regra de aspas.

Configuração de programas com variáveis de ambiente

O shell fornece um *ambiente* para os programas que você executa a partir desse shell. Entre outros itens, esse ambiente inclui informações sobre o diretório de trabalho atual no sistema de arquivos, opções de linha de comando transmitidas para o programa e valores de *variáveis de ambiente*. Os programas podem usar essas variáveis de ambiente para alterar seu comportamento ou suas configurações padrão.

Se uma variável de shell não for uma variável de ambiente, somente o shell poderá usá-la. No entanto, se uma variável de shell for uma variável de

ambiente, o shell e todos os programas executados a partir desse shell poderão usar a variável.

Você pode atribuir qualquer variável definida no shell em uma variável de ambiente marcando-a para exportação com o comando `export`.

```
[user@host ~]$EDITOR=vim
[user@host ~]$export EDITOR
```

Você pode definir e exportar uma variável em uma etapa:

`[user@host ~]$ export EDITOR=vim`

Aplicativos e sessões usam essas variáveis para determinar seu comportamento. Por exemplo, o shell define automaticamente a variável `HOME` para o nome do arquivo do diretório pessoal do usuário quando ele é iniciado. Você pode usar essa variável para ajudar os programas a definir onde salvar os arquivos.

Outro exemplo é a variável `LANG`, que define a codificação do local. Essa variável ajusta o idioma preferido para a saída do programa, o conjunto de caracteres, a formatação de datas, números e moeda e a ordem de classificação dos programas. Se estiver configurado para `en_US.UTF-8`, o idioma usará inglês dos EUA com codificação de caracteres UTF-8 Unicode. Se estiver configurado para, por exemplo, `fr_FR.UTF-8`, o idioma usará francês com codificação UTF-8 Unicode.

```
[user@host ~]$date
Tue Jan 22 16:37:45 CST 2019
[user@host ~]$export LANG=fr_FR.UTF-8
[user@host ~]$date
mar. janv. 22 16:38:14 CST 2019
```

Outra variável de ambiente importante é `PATH`. A variável `PATH` contém uma lista de diretórios separados por dois-pontos que contêm programas:

```
[user@host ~]$echo $PATH
/home/user/.local/bin:/home/user/bin:/usr/share/Modules/bi
n:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
```

Quando você executa um comando como o comando `ls`, o shell procura pelo arquivo executável `ls` em cada um desses diretórios em ordem e executa o primeiro arquivo correspondente encontrado. (Em um sistema típico, esse arquivo é `/usr/bin/ls`.)

Você pode anexar diretórios à sua variável `PATH`. Por exemplo, talvez queira usar alguns programas executáveis ou scripts como comandos regulares no diretório `/home/user/sbin` . Você pode anexar o diretório `/home/user/sbin` ao seu `PATH` para a sessão atual da seguinte maneira:

`[user@host ~]$ export PATH=${PATH}:/home/user/sbin`

Para listar todas as variáveis de ambiente para um shell, execute o comando `env`:

```
[user@host ~]$env
...output omitted...
LANG=en_US.UTF-8
HISTCONTROL=ignoredups
HOSTNAME=host.example.com
XDG_SESSION_ID=4
...output omitted...
```

Definição do editor de texto padrão

A variável de ambiente `EDITOR` especifica o editor de texto padrão para programas de linha de comando. Muitos programas usam o editor `vi` ou `vim` se nada for especificado, mas você pode substituir essa preferência:

```
[user@host ~]$ export EDITOR=nano
```

#Por convenção, as variáveis de ambiente e de shell que são definidas automaticamente pelo shell têm nomes com todos os caracteres maiúsculos. Se você estiver definindo suas próprias variáveis, convém usar nomes com caracteres minúsculos para evitar conflitos de nomenclatura.

Definição automática de variáveis

Quando o Bash é iniciado, vários arquivos de texto são executados com comandos shell que inicializam o ambiente do shell. Para definir variáveis shell ou de ambiente automaticamente ao iniciar o shell, é possível editar esses scripts de inicialização do Bash.

Os scripts exatos que são executados dependem do fato de o shell ser interativo ou não interativo e um shell de login ou sem login.

Um usuário insere comandos diretamente em um shell interativo.

Enquanto um shell não interativo, como um script, é executado em segundo plano sem intervenção do usuário.

Um shell de login é invocado quando um usuário faz login localmente pelo terminal ou remotamente pelo protocolo SSH. Um shell sem login é invocado a partir de uma sessão existente, como para abrir um terminal na GUI do GNOME.

Para shells de login interativos,

os arquivos `/etc/profile` e `~/.bash_profile` configuram o ambiente Bash.

Os arquivos `/etc/profile` e `~/.bash_profile` também fornecem os arquivos `/etc/bashrc` e `~/.bashrc`, respectivamente.

Para shells interativos que não são de login,

somente os arquivos `/etc/bashrc` e `~/.bashrc` configuram o ambiente Bash.

Enquanto os arquivos `/etc/profile` e `/etc/bashrc` se aplicam a todo o sistema, os arquivos `~/.bash_profile` e `~/.bashrc` são específicos do usuário.

Os shells não interativos invocam todos os arquivos que a variável `BASH_ENV` define. Essa variável não é definida por padrão.

Para criar uma variável que esteja disponível para todos os shells interativos

Edite o arquivo `~/.bashrc`.

Para aplicar uma variável apenas uma vez depois que o usuário fizer login

Edita o arquivo `~/.bash_profile`.

Por exemplo, para alterar o editor padrão ao fazer login por SSH, você pode modificar a variável `EDITOR` em seu arquivo `~/.bash_profile`:

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs
export EDITOR=nano
```

A melhor maneira de ajustar as configurações que afetam todas as contas de usuários

é adicionando um arquivo com uma extensão `.sh` que contenha as alterações no diretório `/etc/profile.d`.

Para criar os arquivos no diretório `/etc/profile.d`, faça login como o usuário `root`.

Aliases de Bash

Os aliases de Bash são atalhos para outros comandos do Bash.

Por exemplo, se você precisar digitar com frequência um comando longo, poderá criar um alias mais curto para chamá-lo.

Você usa o comando `alias` para criar aliases. Considere o exemplo a seguir que cria um alias `hello` para um comando `echo`.

```
[user@host ~]$ alias hello='echo "Hello, this is a long strin
```

Você pode, então, executar o comando `hello`, e ele invocará o comando `echo`.

```
[user@host ~]$hello
Hello, this is a long string.
```

Adicione aliases ao arquivo `~/.bashrc` de um usuário para que fiquem disponíveis em qualquer shell interativo.

Desfazer definição e exportação de variáveis e aliases

Para desfazer a configuração e a exportação de uma variável, use o comando `unset`:

```
[user@host ~]$echo $file1
/tmp/tmp.z9pXW0HqcC
[user@host ~]$unset file1
[user@host ~]$echo $file1

[user@host ~]$
```

Para remover a exportação de uma variável sem remover a configuração, use o comando `export -n`:

```
[user@host ~]$ export -n PS1
```

Para desfazer a definição de um alias, use o comando `unalias`:

```
[user@host ~]$ unalias hello
```

Exercicio:

```
Exercício orientado: Alteração do ambiente do shell
Neste exercício, você usa variáveis do shell e expansão de va

Resultados

Editar um perfil de usuário.

Criar uma variável de shell.

Criar uma variável de ambiente

Com o usuário student na máquina workstation, use o comando l

Esse comando garante que todos os recursos necessários estejam

[student@workstation ~]$ lab start edit-bashconfig
Instruções

Altere a variável do shell PS1 do usuário student para [\u@\h

Use o comando ssh para fazer login no servera como o usuário

[student@workstation ~]$ ssh student@servera
...output omitted...
[student@servera ~]$
Use o Vim para editar o arquivo de configuração ~/.bashrc.

[student@servera ~]$ vim ~/.bashrc
Adicione a variável de shell PS1 e seu valor ao arquivo ~/.ba
```

```
...output omitted...
export PATH
PS1='[\u@\h \t \w]$ '
Saia do servera e faça login novamente usando o comando ssh p

[student@servera ~]$ exit
logout
Connection to servera closed.
[student@workstation ~]$ ssh student@servera
...output omitted...
[student@servera 14:45:05 ~]$
Atribua um valor a uma variável de shell local. Os nomes das

Crie uma variável chamada file com o valor tmp.zdkei083. O ar

[student@servera 14:47:05 ~]$ file=tmp.zdkei083
Recupere o valor da variável file.

[student@servera 14:48:35 ~]$ echo $file
tmp.zdkei083
Use o nome da variável file e o comando ls -l para listar o a

[student@servera 14:59:07 ~]$ ls -l $file
-rw-r--r--. 1 student student 0 Jan 23 14:59 tmp.zdkei083
[student@servera 14:59:10 ~]$ rm $file
[student@servera 14:59:15 ~]$ ls -l $file
ls: cannot access 'tmp.zdkei083': No such file or directory
Atribua um valor à variável EDITOR. Use um comando para atrib

[student@servera 14:46:40 ~]$ export EDITOR=vim
[student@servera 14:46:55 ~]$ echo $EDITOR
vim
Retorne ao sistema workstation como o usuário student.

[student@servera 14:47:11 ~]$ exit
logout
Connection to servera closed.
[student@workstation ~]$
```

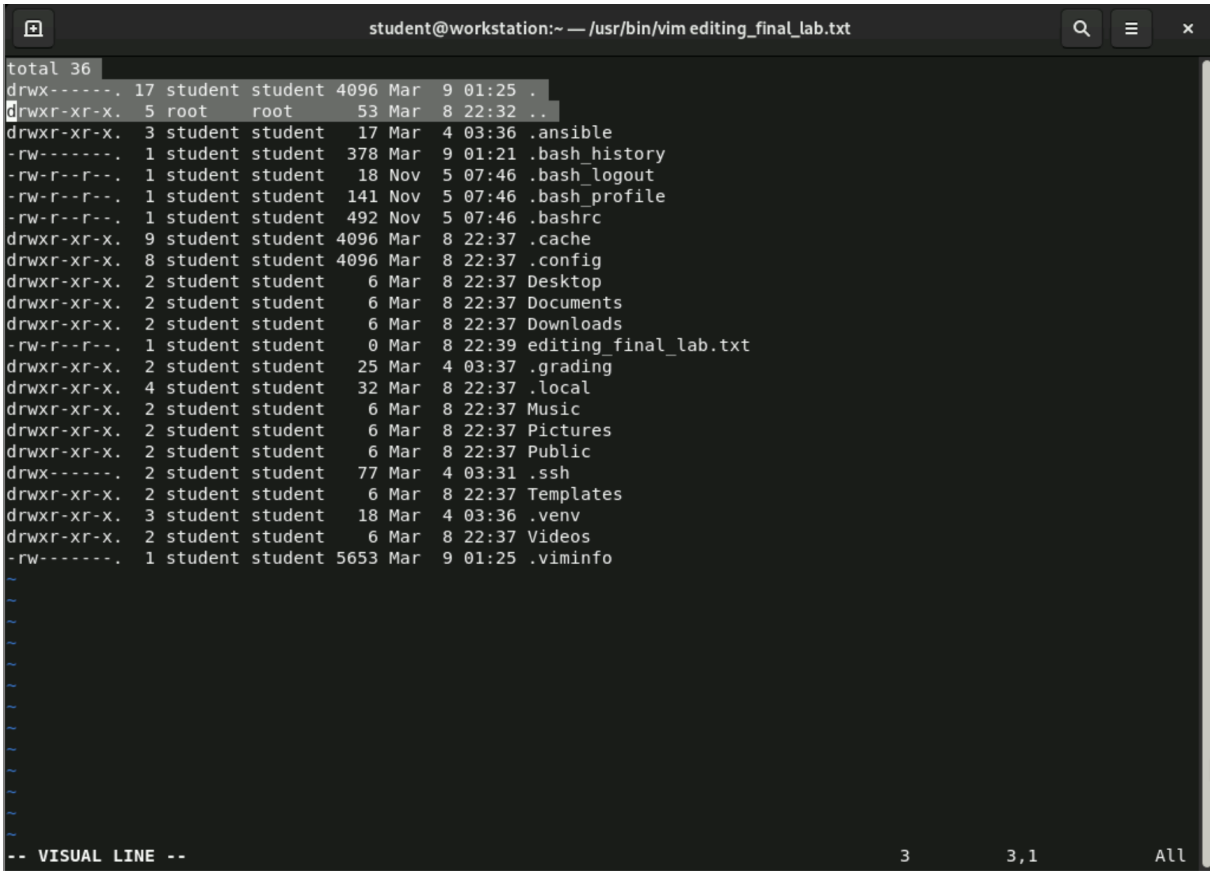
```
Encerramento

Na máquina workstation, altere para o diretório pessoal do us

[student@workstation ~]$ lab finish edit-bashconfig
Isso conclui a seção.
```

Laboratório Aberto: Criação, visualização e edição de arquivos de texto

Em `workstation`, crie a variável de shell `lab_file` e atribua `editing_final_lab.txt` como o valor. Liste o diretório pessoal `student`, incluindo diretórios e arquivos ocultos, e redirecione a saída para o arquivo `editing_final_lab.txt` usando a variável de shell.

1. Use o Vim para editar o arquivo `editing_final_lab.txt`. Use a variável de shell `lab_file`.

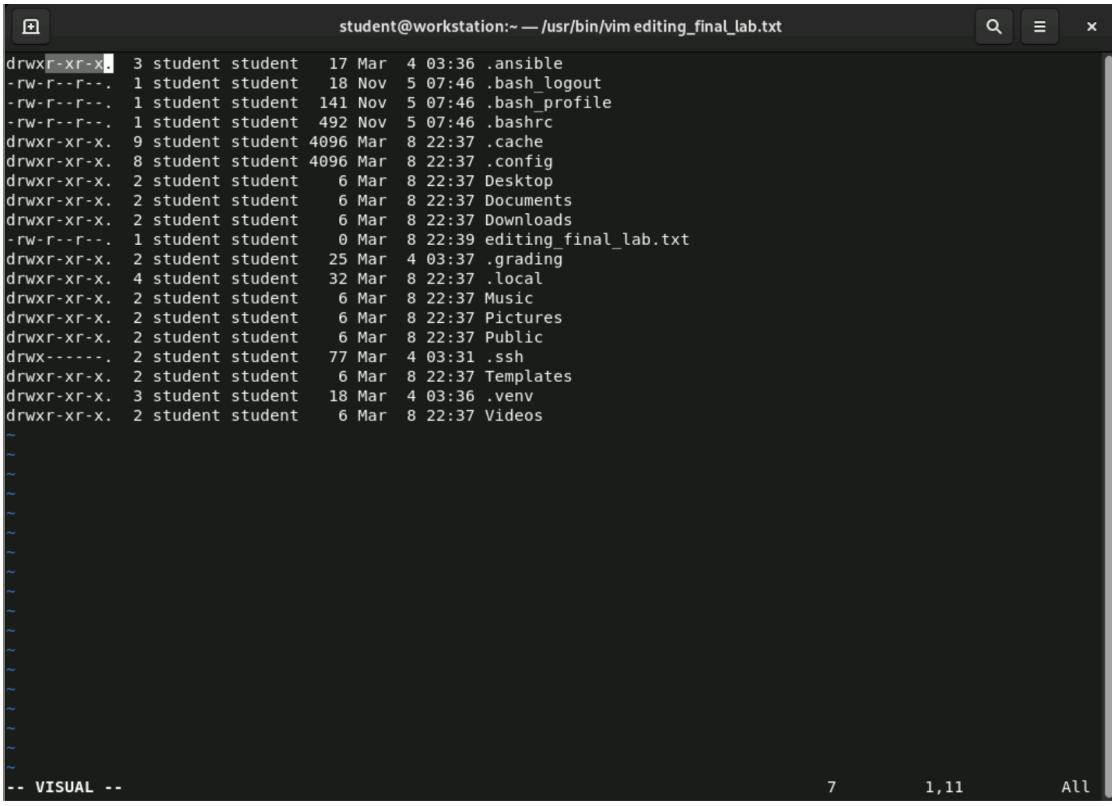
2. Entre no modo visual baseado em linhas do Vim. Sua saída de tela pode ser diferente desses exemplos. Remova as três primeiras linhas do arquivo `editing_final_lab.txt`.



- Aperte SHIFT + V
- Depois com as setas move pra cima e aperte x

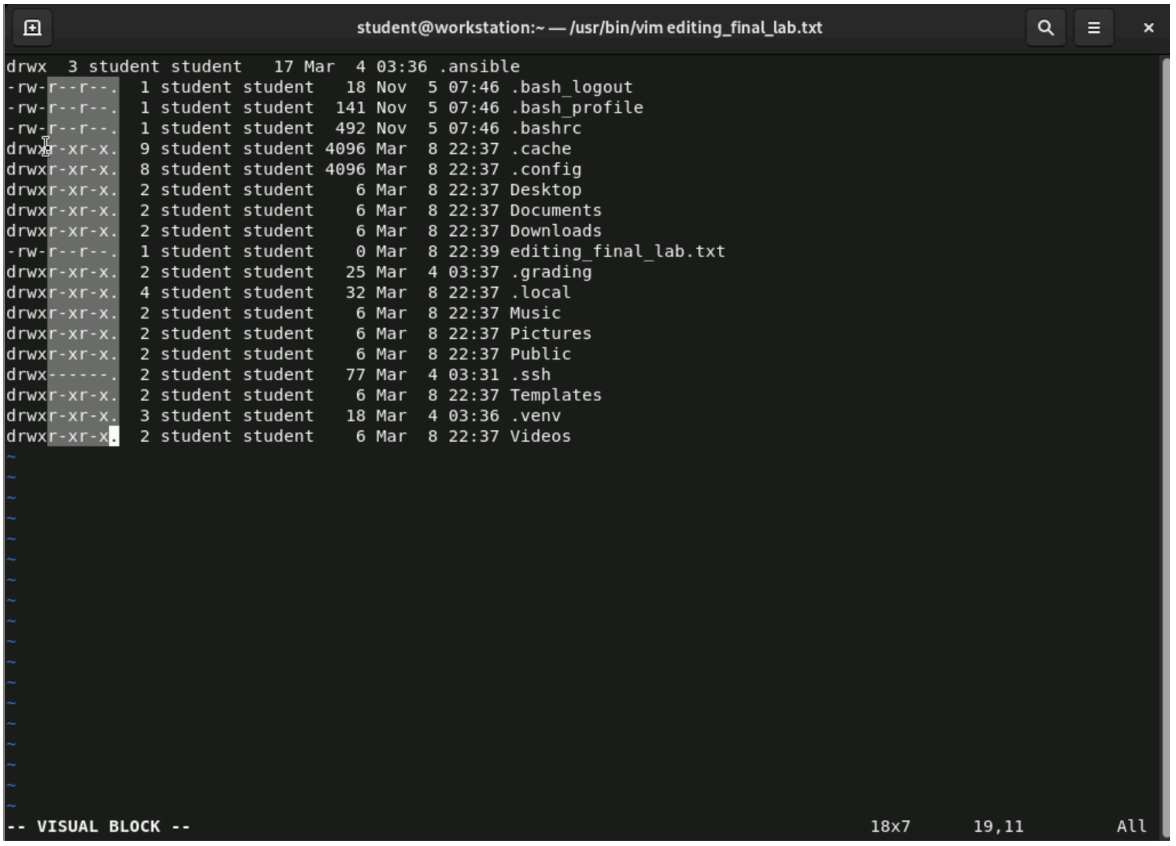
3. Entre no modo visual do Vim. Remova os últimos sete caracteres da primeira coluna na primeira linha. Preserve apenas os primeiros quatro caracteres da primeira coluna.

   Use as teclas de seta para posicionar o cursor no último caractere da primeira coluna na primeira linha. Exclua a seleção digitando **x**.



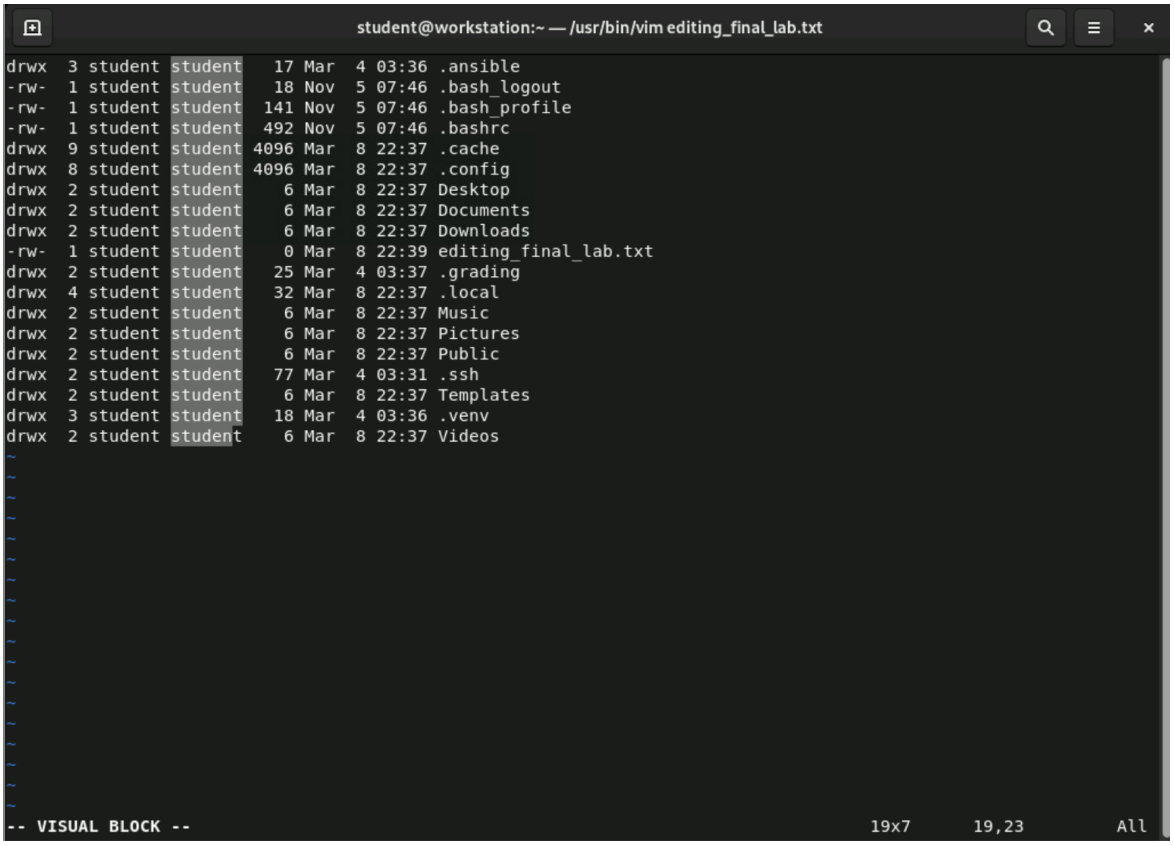
- Primeiro coloque o cursor encima da linha, depois aperta CONTROL + V e x para apagar

4. Entre no modo de bloqueio visual do Vim. Repita a operação da etapa anterior, mas dessa vez selecione da segunda à última linha. Preserve apenas os primeiros quatro caracteres da primeira coluna.



- Na quarta linha, CONTROL + V

5. Entre no modo de bloqueio visual do Vim e remova a quarta coluna do arquivo.


student@workstation:~ — /usr/bin/vim editing_final_lab.txt
drwx 3 student student   17 Mar 4 03:36 .ansible
-rw- 1 student student   18 Nov 5 07:46 .bash_logout
-rw- 1 student student 141 Nov 5 07:46 .bash_profile
-rw- 1 student student 492 Nov 5 07:46 .bashrc
drwx 9 student student 4096 Mar 8 22:37 .cache
drwx 8 student student 4096 Mar 8 22:37 .config
drwx 2 student student   6 Mar 8 22:37 Desktop
drwx 2 student student   6 Mar 8 22:37 Documents
drwx 2 student student   6 Mar 8 22:37 Downloads
-rw- 1 student student   0 Mar 8 22:39 editing_final_lab.txt
drwx 2 student student   25 Mar 4 03:37 .grading
drwx 4 student student   32 Mar 8 22:37 .local
drwx 2 student student   6 Mar 8 22:37 Music
drwx 2 student student   6 Mar 8 22:37 Pictures
drwx 2 student student   6 Mar 8 22:37 Public
drwx 2 student student   77 Mar 4 03:31 .ssh
drwx 2 student student   6 Mar 8 22:37 Templates
drwx 3 student student   18 Mar 4 03:36 .venv
drwx 2 student student   6 Mar 8 22:37 Videos
-- VISUAL BLOCK --
19x7
19,23
All

- CONTROL + V e x