Capítulo 5: Redirecionamento e Pipes; Filtrando e Buscando Informações

Type	Leitura
Materials	Capítulo 5.pdf
Reviewed	☐

REDIRECIONAMENTO E PIPES

- Todo shell em um sistema operacional precisa comunicar-se com o usuário por meio de dispositivos de entrada e saída.
- Um shell Linux, como bash, recebe entrada e envia saída como sequências ou fluxos de caracteres.
  - Cada caractere é independente do que vem antes e do que vem depois dele.
  - Os caracteres não são organizados em registros estruturados ou blocos de tamanho fixo.
- Fluxos são acessados utilizando técnicas de E/S (**E**ntrada/**S**aída) de arquivo,
  - Não importa se o fluxo de caracteres real vem de ou vai para um arquivo, um teclado, uma janela em um monitor ou outro dispositivo de E/S.

- Shells Linux usam três fluxos de E/S padrão, cada um dos quais é associado com um descritor de arquivo:
  - **Entrada padrão (stdin) –** Entrada de um fluxo de dados, podendo ser destacado o teclado.
    - Fluxos de entrada fornecem entrada para programas, normalmente de digitações em um terminal.
    - O descritor é representado pelo **número 0**.
  - **Saída padrão (stdout) –** Saída de um fluxo de dados em condições normais. Exemplos: monitor, impressora, arquivos, etc.
    - Fluxos de saída imprimem caracteres de texto.
    - O descritor é representado pelo **número 1**.
  - **Saída de erro (stderr) –** Saída de um fluxo de dados em condições de erro ou insucesso em um determinado processamento, que poderá ser direcionada para o monitor ou arquivo de LOG.
    - O descritor é representado pelo **número 2**.

`>`

- Redireciona a saída padrão de um programa/comando/script para algum arquivo ao invés do dispositivo de saída padrão (tela).Ele sobrescreve

```
Exemplo:
– ls > teste.txt : Envia a saída do comando ls para o arquivo teste.txt.
– ls > /dev/tty2 : Envia a saída do comando ls para o segundo console.
```

`>>`

- Adiciona as linhas ao final do arquivo ao invés do dispositivo de saída padrão (tela).Basicamente ele dá um 'append' e ele não sobrescreve
- `Ex: ls /usr/fake > deucerto.txt2> deuerrado.txt`:
- se o diretório `/usr/fake` existir, a lista de arquivos será salva em `deucerto.txt`, e se não existir, a mensagem de erro será salva em `deuerrado.txt`.

```
Exemplo:
– ls >> teste.txt : Adiciona a saída do comando ls ao final d
arquivo teste.txt, se ele existir.
– ls >> /dev/tty2 : Envia a saída do comando ls para o
segundo console

echo "testando 1234..." >> arquivo1.txt
```

<

- Faz com que o comando leia os dados do arquivo especificado em vez de esperar por entrada do teclado.

```
cat < teste.txt

Isso faz com que o conteúdo do arquivo teste.txt seja enviado
```

<<

- insira várias linhas de entrada interativamente, terminando a entrada quando uma determinada palavra-chave é digitada.

```
#!/bin/bash

# Este script solicita ao usuário para inserir seu nome
# e, em seguida, imprime uma saudação personalizada.

echo "Por favor, insira seu nome:"
# O operador '<<' redireciona a entrada para o comando 'read'
# a partir de uma sequência de caracteres terminada pela pala
read -r nome << EOF
EOF

echo "Olá, $nome! Bem-vindo!"
```

`tr`

- Troca/substituição de caracteres
- Não aceita arquivo como argumento

```
tr 'a-z' 'A-Z'

echo "Hello World" | tr '[:lower:]' '[:upper:]'
```

CONECTORES

`|` (Pipe)

- **Descrição**: Conecta a saída de um comando como entrada para outro comando.
- **Exemplo**:

```
cat a.txt b.txt | grep nome
```

`;` (Ponto e Vírgula)

- **Descrição**: Usado para executar vários comandos em sequência, independentemente do sucesso ou falha de cada um.
- **Exemplo**:

```
echo "Arquivo 1"; cat $1; echo "Arquivo 2"; cat $2
```

`&&` (E Lógico)

- **Descrição**: O segundo comando só é executado se o primeiro for bem-sucedido (retornar código 0).
- **Exemplo**:

```
mkdir teste && echo "Diretório criado com sucesso"
```

`||` (Ou Lógico)

- **Descrição**: O segundo comando só é executado se o primeiro falhar (retornar código diferente de 0).
- **Exemplo**:

```
mkdir teste || echo 'Não foi possível criar o diretório "teste"'
```

tee

- ler a entrada padrão e escrever tanto na saída padrão quanto em um ou mais arquivos. - recebe os dados da saída padrão e os grava no arquivo "copia_arquivo.txt". Além disso, ele também exibe esses dados na saída padrão.
    - Por exemplo, se você deseja visualizar o conteúdo de um arquivo e, ao mesmo tempo, salvá-lo em outro arquivo, você pode usar o `tee`

```
cat arquivo.txt | tee novo_arquivo.txt
```

FILTRANDO E BUSCANDO INFORMAÇÕES

sort

- classificar as linhas de texto em ordem alfabética ou numérica

- `t`: Define o delimitador para separar os campos de cada linha.
- `k3`: Especifica o campo a ser usado como chave para ordenação.

- `n`: Indica que a ordenação deve ser numérica.
- `r`: Reverte a ordem de classificação, tornando-a decrescente.

```
cat arquivo.txt | tee novo_arquivo.txt
```

```
Ordenar um arquivo de texto em ordem alfabética:

sort arquivo.txt


Ordenar um arquivo de texto em ordem numérica:

sort -n numeros.txt


Ordenar um arquivo de texto em ordem inversa:

sort -r arquivo.txt


Ordenar um arquivo de texto ignorando maiúsculas e minúsculas

sort -f arquivo.txt


Ordenar um arquivo de texto e remover linhas duplicadas:
```

```
sort -u arquivo.txt
```

Ordenar um arquivo de texto e salvar a saída em um novo arqui

```
sort entrada.txt -o saida.txt
```

Ordenar um arquivo de texto considerando apenas os primeiros

```
sort -k1.1,1.3 arquivo.txt
sort -t, -k1n -k3 sort.txt
```

Suponha que temos o seguinte arquivo chamado "dados.txt":

```
1,Ana,25
2,João,30
3,Maria,22
```

Podemos ordenar este arquivo com base na terceira coluna (ida

bash

```
sort -t ',' -k3n dados.txt
```

Isso produzirá a seguinte saída:

```
3,Maria,22
1,Ana,25
2,João,30

Explicação do comando:

    -t ',': Define a vírgula como delimitador de campo.
    -k3n: Especifica que queremos classificar com base no ter
    dados.txt: O nome do arquivo de entrada.
```

cut (cortando campo)

`cut -d ":" -f 1 /etc/passwd` - pega o primeiro campo

`cut -d ":" -f 1, 2 /etc/passwd` - pega o primeiro e segundo campo

`cut -b 1-4 /etc/passwd` - pega os bytes de 1 ao 4, pega todos os espaços

`cut -c 1-4 /etc/passwd` - pega os bytes de 1 ao 4, porém sem contar os espaços

```
cut OPÇÕES ARQUIVO

Aqui estão algumas opções comuns do comando cut:

    -c LISTA: Especifica quais caracteres devem ser incluídos
    -f CAMPOS: Especifica quais campos devem ser incluídos na
    -d DELIMITADOR: Especifica o caractere delimitador de cam

Exemplo de uso:

Suponha que temos um arquivo chamado "nomes.txt" com o seguin

João, Silva, 25
Maria, Santos, 30
```

```
Para extrair apenas o primeiro campo (nome) deste arquivo, po

bash

cut -d ',' -f1 nomes.txt

Isso produzirá a seguinte saída:

João
Maria
```

grep

Comandos Básicos

- `grep 'root' /etc/passwd`

  Encontra todas as linhas no arquivo `/etc/passwd` que contêm "root".

- `grep -v 'www-data' /etc/passwd`

  Inverte a busca, exibindo linhas exceto as que contêm "www-data".

- `grep -f /tmp/patternfile /etc/passwd`

  Usa um arquivo (`/tmp/patternfile`) para definir o padrão de busca.

Opções de Contexto

- `A [número]`

  Mostra o número de linhas após a linha encontrada.

- `B [número]`

  Mostra o número de linhas antes da linha encontrada.

- `C [número]`

  Mostra o número de linhas ao redor da linha encontrada (antes e depois).

Opções de Contagem e Numeração

- `c`

Conta quantas linhas correspondem ao padrão.

- `n`

  Exibe os números das linhas correspondentes.

Opções de Correspondência e Sensibilidade ao Caso

- `w`

  Retorna apenas linhas que contêm correspondências como palavras inteiras.

- `i`

  Realiza uma busca case-insensitive (ignora maiúsculas e minúsculas).

- `grep -i 'WWW-DaTa' /etc/passwd`

  Busca case-insensitive por "WWW-DaTa" no arquivo `/etc/passwd`.

Opções de Expressões Regulares e Literais

- `E`

  Usa expressões regulares estendidas.

- `grep -iE '^www-data.*nologin$' /etc/passwd`

  Encontra linhas que começam com "www-data" e terminam com "nologin" (case-insensitive).

- `F`

  Interpreta o padrão como uma string literal (não regex).

- `grep -iF '.*' /etc/passwd`

  Interpreta ".*" como caracteres literais, útil para buscas de caracteres especiais.

Opções de Busca Recursiva

- `r`

  Realiza busca recursiva nos diretórios.

- `ri 'Foca02' .`

  Busca recursivamente, ignorando o case, por "Foca02" no diretório atual.

- `rh`

Oculta os nomes dos arquivos nos resultados da busca recursiva.

- `grep -ril 'Foca02' .`

  Lista os nomes dos arquivos que contêm "Foca02", sem mostrar o conteúdo.

Expressões regulares