Laboratório 1.2: Entrada e Saída Padrões, Redirecionamento, Pipes

Introdução

As mensagens normais dos programas em modo texto são enviadas para o que é conhecido como *saída padrão* (*stdout - standard output*). Em geral, essa saída padrão é o próprio terminal e é por isso que conseguimos ver tais mensagens.

Além da saída padrão, tem-se também a *saída de erro* (*stderr - standard error*), onde as mensagens de erro são enviadas. Normalmente, a saída de erro é o terminal também.

Por fim, os programas possuem também uma *entrada padrão*, em que eles esperam entrada de valores para serem manipulados. Em geral, essa entrada é o teclado. Nem todos os programas esperam alguma entrada, mas os que precisam, fazem-no através da entrada padrão.

A figura a seguir mostra uma visão geral dessas entradas e saídas:

O Linux permite redirecionar as saídas para arquivos (veremos na próxima seção) ou até mesmo para a entrada de outros programas (veremos mais adiante).

1. Redirecionando a Saída para Arquivos

Para redirecionar a saída padrão para um arquivo, usamos o caractere > (maior que):

```
$ ls -lh /usr/bin > ArquivosDoBin.txt
```

O comando anterior executa o `ls` e, ao invés de ser enviado para o terminal, a saída padrão foi redirecionada para o arquivo `ArquivosDoBin.txt`. Portanto, nada será impresso no terminal. Vamos ver o final do arquivo para ver se a saída foi realmente enviada para ele:

```
$ tail -5 ArquivosDoBin.txt
```

```
-rwxr-xr-x 1 root   root    7.5K Dec 13  2019 zgrep
-rwxr-xr-x 1 root   root     50K Oct 19  2020 zipdetails
-rwxr-xr-x 1 root   root    2.2K Dec 13  2019 zless
-rwxr-xr-x 1 root   root    1.8K Dec 13  2019 zmore
-rwxr-xr-x 1 root   root    4.5K Dec 13  2019 znew
```

Comando `echo`

Podemos usar o comando `echo` para escrever uma mensagem na saída padrão:

```
$ echo -e "(\(\\ \n(-.-)\no_(\")(\")"
```

```
(\(\
(-.-)
o_(")(")
```

A opção `-e` (*escapes*), instrui o `echo` a interpretar caracteres especiais como o `\n` (nova linha), dentre outros. Podemos usar o `echo` para enviar um texto rapidamente para um arquivo, redirecionando a saída padrão:

```
$ echo "Primeira linha do arquivo" > ArquivoTeste.txt
$ cat ArquivoTeste.txt
```

```
Primeira linha do arquivo
```

Se executarmos o redirecionamento novamente, ele sobrescreve o arquivo:

```
$ echo "Segunda linha do arquivo (só que não)" > ArquivoTeste.txt
$ cat ArquivoTeste.txt
```

```
Segunda linha do arquivo (só que não)
```

Concatenando

Caso você queira escrever no final do arquivo, sem sobrescrevê-lo, use `>>`:

```
$ echo "Segunda linha do arquivo (de verdade)" >> ArquivoTeste.txt
$ cat ArquivoTeste.txt
```

```
Segunda linha do arquivo (só que não)
Segunda linha do arquivo (de verdade)
```

Redirecionando a Saída de Erro

Conforme mencionado, além da saída padrão (*stdout*), tem-se também a saída de erro (*stderr*). Normalmente, a saída de erro é também o terminal, mas podemos redirecioná-la para um arquivo também.

Só para você perceber que são coisas diferentes, execute o comando abaixo, que pede para listar um arquivo inexistente e outro arquivo válido:

```
$ ls ArquivoInvalido.txt ArquivoTeste.txt > LsStdout.txt
```

```
ls: cannot access 'ArquivoInvalido.txt': No such file or directory
```

Note como estamos pedindo para redirecionar a saída do comando para um arquivo mas, mesmo assim, a mensagem de erro (arquivo não encontrado) é impressa na tela e o arquivo `LsStdout.txt` contém apenas o nome do arquivo existente, sem mensagens de erro. O motivo disso é que a mensagem de erro foi enviada para a saída de erro (terminal), e não para a saída padrão (redirecionada para o arquivo).

Para redirecionar a saída de erro, usamos o `2>` :

```
$ ls ArquivoInvalido.txt ArquivoTeste.txt 2> LsStderr.txt
```

```
ArquivoTeste.txt
```

Note agora que nenhuma mensagem de erro apareceu no terminal, enquanto que o arquivo existente foi listado. O motivo é que estamos direcionando a saída de erro para o arquivo `LsStderr.txt`. Veja o seu conteúdo:

```
$ cat LsStderr.txt
```

```
ls: cannot access 'ArquivoInvalido.txt': No such file or directory
```

Você pode redirecionar a saída padrão para um arquivo e a saída de erro para outro em um único comando:

```
$ ls ArquivoInvalido.txt ArquivoTeste.txt > LsStdout.txt 2> LsStderr.txt
```

Ou, se você quiser redirecionar tanto a saída padrão quanto a de erro para o mesmo arquivo, você pode usar o `&>` :

```
$ ls ArquivoInvalido.txt ArquivoTeste.txt &> LsStdoutStdErr.txt
```

Redirecionando para o `/dev/null`

O arquivo `/dev/null` é um arquivo especial do Linux/Unix em que tudo que é enviado para ele é descartado. Imagine esse arquivo como uma lixeira ou mesmo um buraco negro de dados. Uma prática comum quando queremos executar um comando que sabemos que pode gerar alguns erros que queremos ignorar é mandar a saída de erro para o `/dev/null`:

```
$ ls ArquivoInvalido.txt ArquivoTeste.txt 2> /dev/null
```

O `/dev/null` é um de vários outros arquivos "especiais" no Linux. Estes arquivos se parecem com um arquivo normal, mas eles são, na verdade, uma porta de comunicação com alguma parte ou módulo do kernel. O motivo disso é que o Linux segue uma filosofia muito usada no mundo Unix: *everything is a file*, ou seja, tudo é arquivo. Nesta filosofia, o kernel e seus módulos criam arquivos nos diretórios `/dev`, `/proc` ou `/sys` para disponibilizar e receber informações dos programas normais. Isso permite separar os dois mundos usando toda a segurança dos

arquivos e permite também usar as ferramentas já disponíveis para manipular tais arquivos (e.g., `cat`, `echo`, editores de texto). Nestes casos, e no caso do `/dev/null`, o disco rígido não é realmente acessado, apesar deles serem arquivos.

2. Redirecionando a Saída para a Entrada de Outro Programa

Conforme mencionado, os programas possuem também uma entrada padrão (*stdin*) que, normalmente, é o teclado. Um exemplo de programa que usa a entrada padrão é o `grep`.

Comando `grep`

O `grep` (*g/re/p - globally search for a regular expression and print matching lines*) é um programa usado para filtrar a entrada padrão e enviar para a saída padrão apenas as linhas correspondentes à expressão regular passada como parâmetro para o comando.

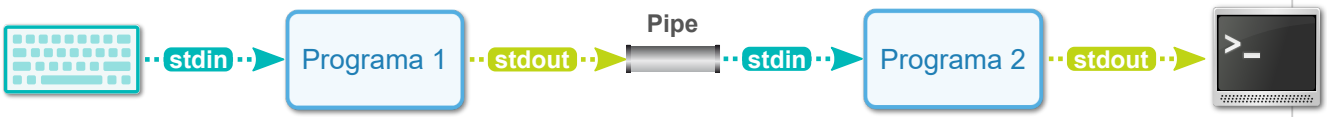
Apesar de não ser tão comum, você pode executar esse comando usando o teclado como entrada padrão:

```
$ grep a
```

O comando anterior ficará esperando por linhas na entrada padrão (teclado) e irá imprimir na saída padrão (terminal) apenas as linhas que possuem o caractere "a". Por exemplo, digite a frase "`Goofy Goober Rock`" e pressione *enter* (nova linha). Note que nada acontece. Agora, digite o nome "`Dovahkiin`" e pressione *enter*. Note que a linha é repetida no terminal (saída padrão) com o caractere "a" marcado. Pressione `Ctrl+C` para finalizar o programa.

Usando o *pipe* para Redirecionar a Saída

Assim como nós redirecionamos a saída dos programas para um *arquivo* na seção anterior, nós podemos redirecionar a saída de um programa para ser a entrada para outro *programa*. Isso é feito através do caractere `|`, conhecido no mundo Unix como *pipe* (do inglês, cano, tubo). A figura a seguir ilustra o uso do pipe, ignorando a saída de erro para simplificar:



Note como a saída de um programa (*stdout*) vira a entrada para outro programa (*stdin*).

No comando a seguir, usaremos o `cat` para enviar o conteúdo do arquivo `/proc/cpuinfo` para a saída padrão. Em seguida, estamos usando o *pipe* para redirecionar a saída do `cat` para a entrada do comando `grep`, usado para filtrar apenas as linhas que possuem o conteúdo "`model name`":

```
$ cat /proc/cpuinfo | grep "model name"
```

```
model name      : Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
model name      : Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
model name      : Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
```

Se você tiver um processador com vários núcleos, você deve ter percebido que o comando anterior gerou várias linhas iguais (uma para cada núcleo do processador). Podemos resolver esse problema, pedindo para mostrar apenas a primeira linha do resultado do `grep`. Para isso, usaremos o comando `head` (explicado no laboratório anterior):

```
$ cat /proc/cpuinfo | grep "model name" | head -1
```

```
model name      : Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
```

Note como o comando está começando a ficar mais complexo. Estamos executando três comandos diferentes, conectados via *pipes*. Primeiro, o `cat` envia o conteúdo do arquivo para a entrada do `grep` que filtra o conteúdo enviando apenas as linhas que possuem "`model name`" para o comando `head`, que irá mostrar apenas a primeira das várias linhas.

Dependendo do que queremos, esse comando pode ficar ainda mais complexo. No exemplo a seguir, continuaremos a aumentar o comando anterior, desta vez para remover tudo que está antes do "`:`". Para isso, usamos o comando chamado `cut`:

```
$ cat /proc/cpuinfo | grep "model name" | head -1 | cut -d: -f2
```

```
cat /proc/cpuinfo | grep "model name" | head -1 | cut -d: -f2
```

A opção `-d` do `cut` seta o "delimitador" para ser o caractere "`:`". Já a opção `-f` (*field*, campo) diz que queremos pegar apenas a segunda parte, ignorando a primeira (o que está antes do "`:`"). Entretanto, note que ficou um espaço no início da saída. Vamos usar o `cut` novamente para remover esse espaço:

```
$ cat /proc/cpuinfo | grep "model name" | head -1 | cut -d: -f2 | cut -c2-
```

```
Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
```

Na última parte do comando, estamos usando novamente o `cut` para pegar os caracteres (`-c`) da entrada a partir do segundo caractere até o final. Veja a sintaxe da opção `-c` no `man` do comando para ver sua descrição detalhada.

Desafio: *Você conseguiria, usando as informações já disponíveis, melhorar o comando anterior para ter a mesma saída, mas usando o comando `cut` apenas uma única vez? Teste no terminal e escreva o comando abaixo.*

```
cat /proc/cpuinfo | grep "model name" | head -1 | cut -d: -f2
```

3. Desafios (Opcionais)

A seguir, serão pedidas algumas tarefas. Todas elas podem ser feitas com apenas uma única linha de comando, usando redirecionamentos e *pipes*. Você não precisa submeter o resultado (saída) do comando, mas sim o comando executado.

Comando `wc`

Um comando não mostrado ainda é o `wc`. Ele conta a quantidade de linhas passadas na entrada padrão. Por exemplo, `cat /etc/passwd | wc -l` retornará a quantidade total de usuários do sistema. Use o `man` para saber mais detalhes. Com base nisso, escreva abaixo um comando para contar a quantidade de usuários que possuem a letra "h" em alguma parte de sua definição.

```
cat /etc/passwd | grep h| wc -l
```

O comando anterior contou os usuários que possuem a letra "h" não apenas no login (o primeiro campo do arquivo `passwd`), mas também nos outros campos (diretório home, shell, etc). Você consegue fazer um comando para contar apenas os usuários que possuem "h" no login (primeiro campo). Use o comando `cut` para mostrar só o primeiro campo. Dê um `cat` no arquivo `passwd` para você ver o seu formato. Os campos são separados por ":" (dois pontos).

```
cat /etc/passwd| cut -d: -f1| grep h | wc -l
```

Nana korobi, ya oki. — Japanese Proverb

π