Tópicos Especiais em Fundamentos da Computação
Shell Scripting

Entradas, Saídas e Redirecionamentos

Andrei Rimsa Álvares
andrei@cefetmg.br

Sumário

- Entrada, saída e saída de erro padrão
- Redirecionamentos
- *Pipes*
- Redirecionamentos avançados

ENTRADA, SAÍDA E SAÍDA DE ERRO PADRÃO

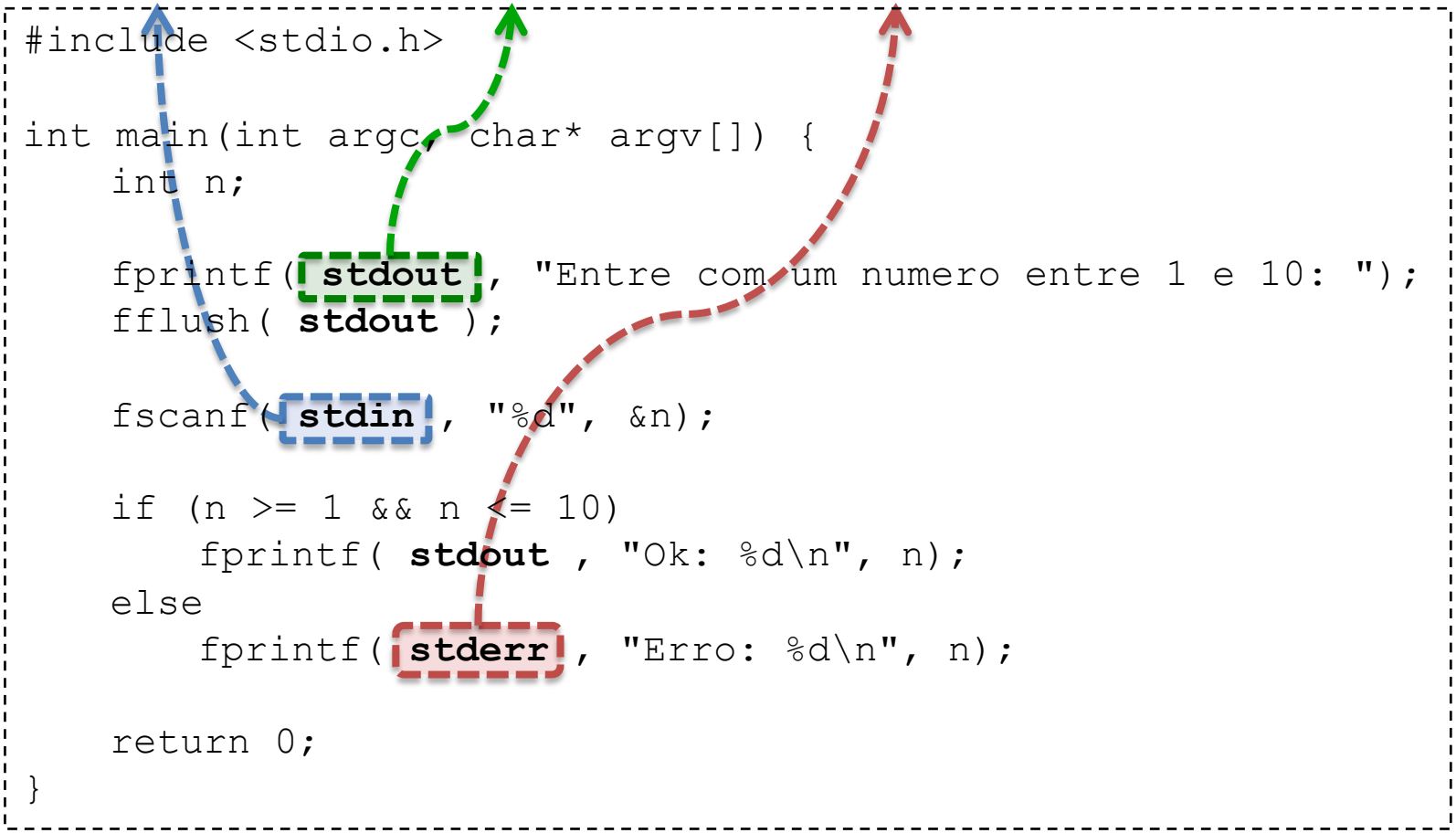
Shell Scripting

Entradas e Saídas

- Um programa quando executado espera que sejam configurados para ele a **entrada padrão**, a **saída padrão** e a **saída de erro padrão**
  - **Entrada padrão:** é o lugar de onde o programa recebe informações, por padrão do teclado
  - **Saída padrão:** é o lugar de onde o programa pode mandar informações, por padrão é enviado para a tela
  - **Saída de erro padrão:** é o lugar de onde o programa pode mandar mensagens de erro, por padrão é enviado também para a tela

Entradas e Saídas

- O programa pode então fazer uso desses "arquivos" configurados: **entrada padrão**, **saída padrão** e **saída de erro padrão**

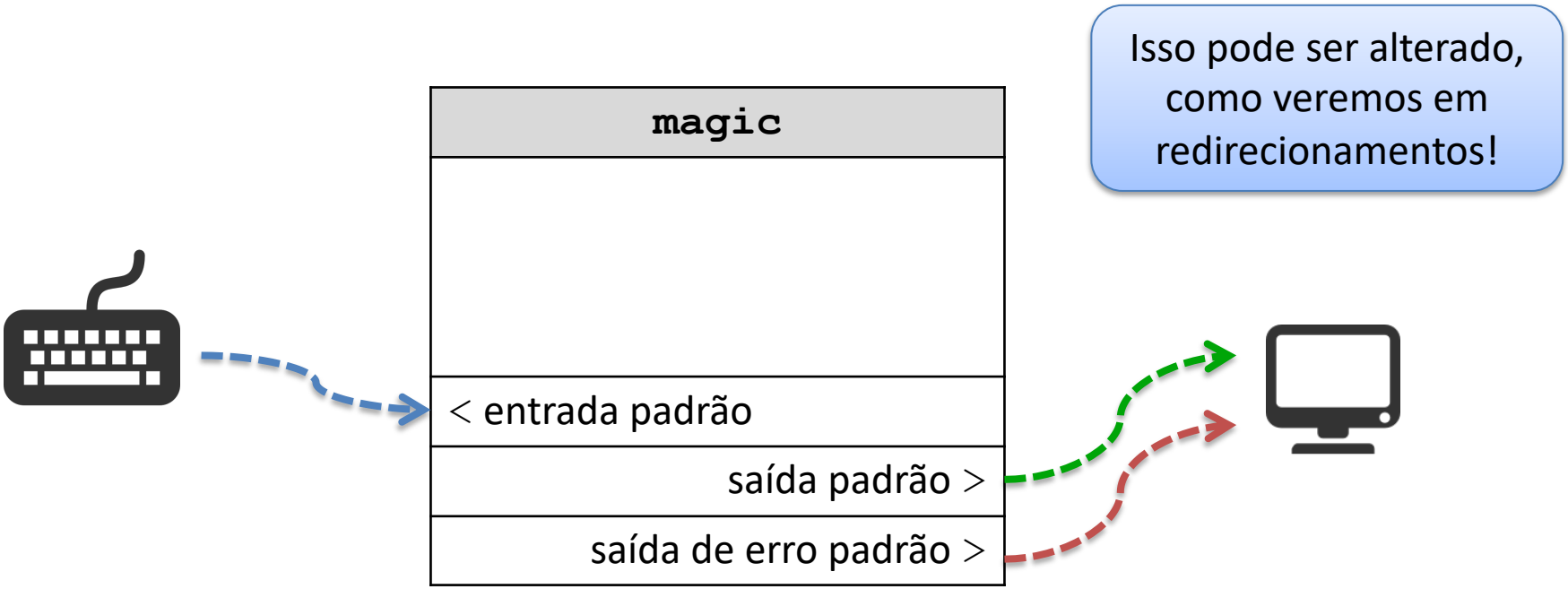


```c
#include <stdio.h>

int main(int argc, char* argv[]) {
    int n;

    fprintf( stdout, "Entre com um numero entre 1 e 10: ");
    fflush( stdout );

    fscanf( stdin, "%d", &n);

    if (n >= 1 && n <= 10)
        fprintf( stdout , "Ok: %d\n", n);
    else
        fprintf( stderr, "Erro: %d\n", n);

    return 0;
}
```

magic.c

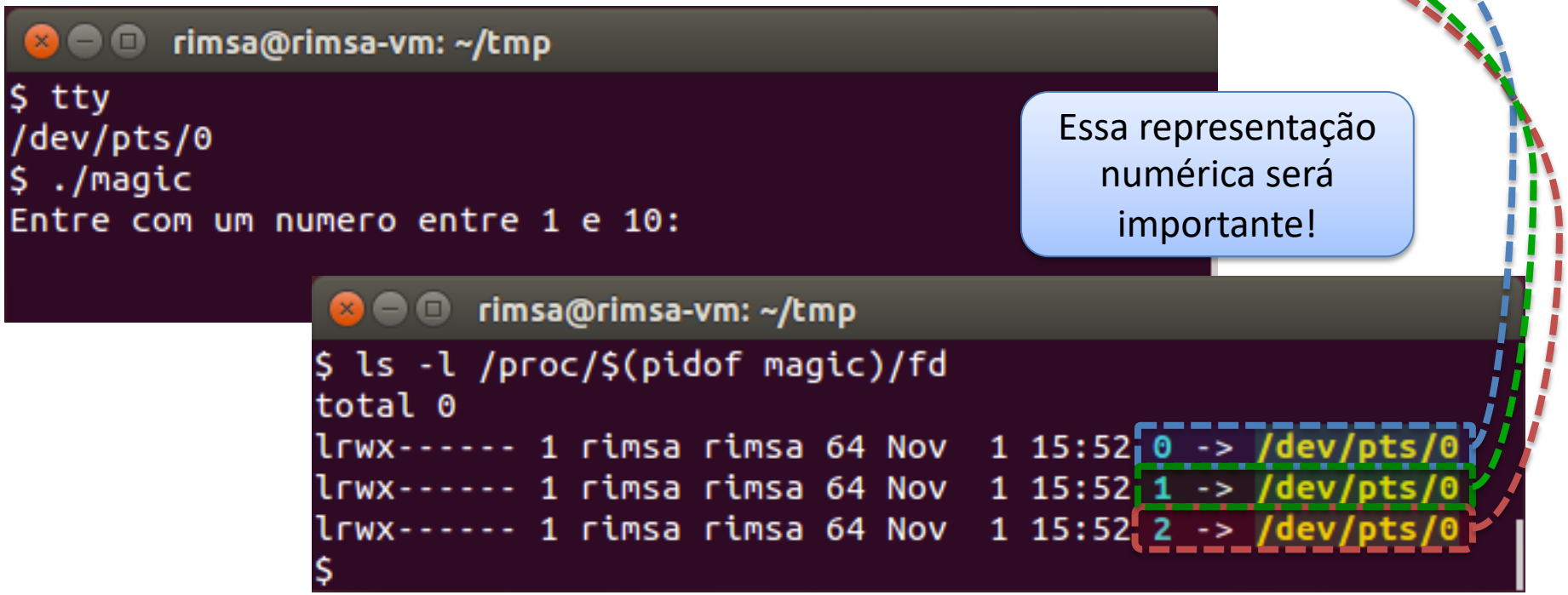
Entradas e Saídas

- As entradas e saídas são mapeadas no dispositivo de terminal (*tty*), em que as entradas são obtidas por padrão do teclado e a saída por padrão exibida na tela



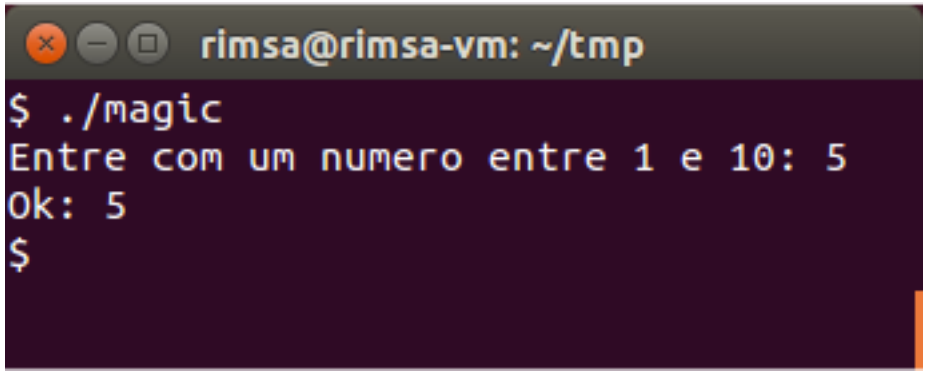
Entradas e Saídas

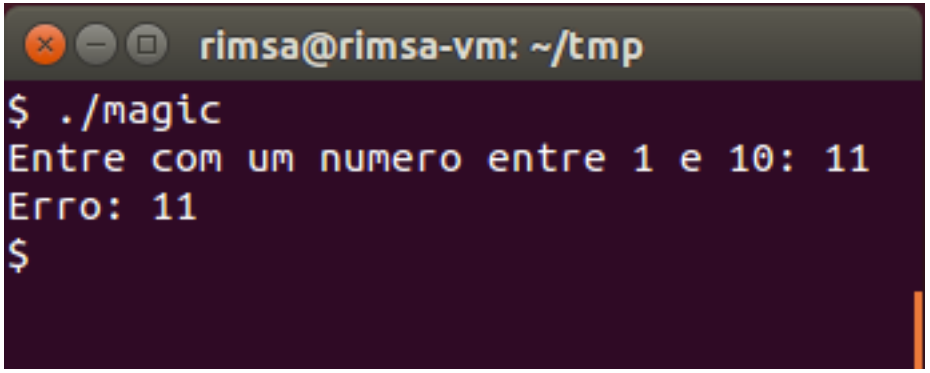
- Esses arquivos são mapeados nos seguintes descritores de arquivos
  - **Entrada padrão:** descritor de arquivo `0` **(zero)**
  - **Saída padrão:** descritor de arquivo `1` **(um)**
  - **Saída de erro padrão:** descritor de arquivo `2` **(dois)**



Entradas e Saídas

- Executando o programa com saída padrão e com saída de erro





Qual imprimiu na **saída padrão** e qual imprimiu na **saída de erro padrão**? Dá para saber?

REDIRECIONAMENTOS

Redirecionamentos

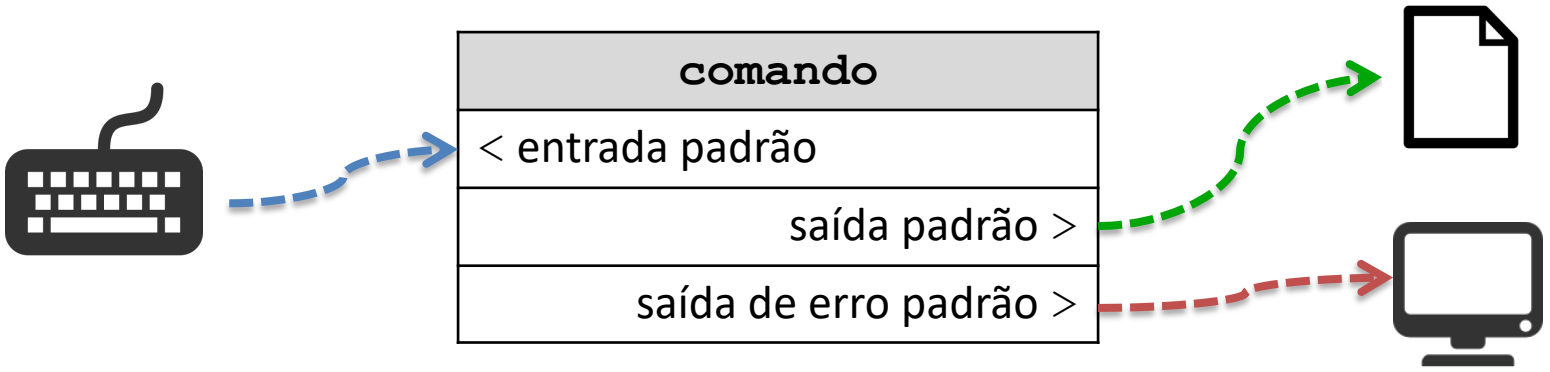
- Muitas vezes você quer salvar a saída de um programa em um arquivo, ao invés de mostrar na tela; ou você quer usar um arquivo de entrada ao invés de digitar toda vez

- Redirecionamentos de entrada e saída permitem **modificar** de onde vem a entrada e para onde vai a saída
  - O programa não "sabe" exatamente de onde vem os dados e nem para onde vão!♣

♣: Um processo pode consultar o kernel para saber qual o tipo de dispositivo

Redirecionando a Saída Padrão

- O símbolo de redirecionamento de saída padrão (>) instruí a *shell* a **redirecionar a saída do comando para o arquivo especificado** ao invés da tela



- O formato para redirecionamento na linha de comando é dado por

```
comando [argumentos] > arquivo
```

Redirecionando a Saída Padrão

- Exemplo usando o comando `cat`

```
$ cat > texto.txt
Esse texto está sendo escrito pelo teclado e o comando
cat está copiando para o arquivo. Pressione CONTROL-D
para sinalizar o fim do arquivo.
^D
```

Útil para escrever em um arquivo sem a ajuda de um editor de textos!

Redirecionando a Saída Padrão

- Outro exemplo usando `cat` para concatenar arquivos

```
$ echo "1 caixa de leite" > leite.txt
$ echo "6 pães doce e 3 de sal" > paes.txt
$ cat leite.txt paes.txt > pedido.txt
$ cat pedido.txt
1 caixa de leite
6 pães doce e 3 de sal
```

Cuidado: se o arquivo já existir, a *shell* irá sobreescrevê-lo e destruir seu conteúdo

Evitando Sobreescrever Arquivos

- A *shell* provê a funcionalidade `noclobber` que previne sobreescrever um arquivo usando redirecionamentos
  - Se tentar sobreeescrever o arquivo, será exibida uma mensagem e o comando não será executado

- Exemplo tentando escrever em um arquivo com o `noclobber` habilitado e posteriormente desabilitado

```
$ touch tmp
$ set -o noclobber
$ echo "hi there" > tmp
bash: tmp: cannot overwrite existing file
$ set +o noclobber
$ echo "hi there" > tmp
```

É possível sobreescrever usando >|

Cuidado

- Dependendo de qual *shell* está usando e como o ambiente está configurado, o seguinte programa pode causar resultados inesperados

```
$ echo "Isso é uma laranja" > laranja.txt
$ echo "Isso é um limão" > limao.txt
$ cat laranja.txt limao.txt > laranja.txt
$ cat laranja.txt
Isso é um limão
```

Como resolver esse problema?

Anexando na Saída

- O símbolo de anexar na saída (>>) faz com que a *shell* adicione novas informações ao final do arquivo especificado, deixando as informações pré-existentes intactas

- Exemplo

```
$ echo "Isso é uma laranja" > laranja.txt
$ echo "Isso é um limão" > limao.txt
$ cat limao.txt >> laranja.txt
$ cat laranja.txt
Isso é uma laranja
Isso é um limão
```

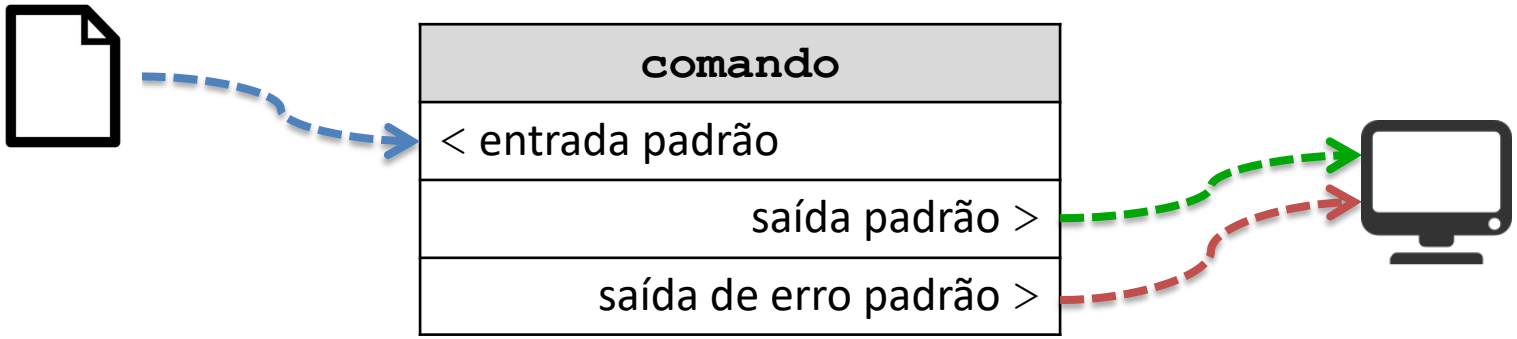
Anexando na Saída

- O exemplo a seguir mostra a concatenação usando um comando de cada vez

```
$ date > online.txt
$ cat online.txt
Sáb Nov  1 19:39:30 BRST 2014
$ who >> online.txt
$ cat online.txt
Sáb Nov  1 19:39:30 BRST 2014
rimsa    :0           2014-11-01 15:10 (:0)
rimsa    pts/0        2014-11-01 15:47 (:0)
rimsa    pts/1        2014-11-01 15:50 (:0)
```

Redirecionando a Entrada Padrão

- O símbolo de redirecionamento de entrada padrão (<) instruí a *shell* a **redirecionar a entrada padrão para um comando vindo do arquivo especificado** ao invés do teclado



- O formato para redirecionamento na linha de comando é dado por

```
comando [argumentos] < arquivo
```

Redirecionando a Entrada Padrão

- Usando o comando `cat` com redirecionamento da entrada vindo de um arquivo tem o mesmo efeito de usar o nome do arquivo como parâmetro

```
$ cat < pedido.txt
1 caixa de leite
6 pães doce e 3 de sal
```

Outros programas funcionam de forma similar: `lpr`, `grep`, `sort`

Combinando Entradas e Saídas

- É possível usar mais de um redirecionamento em um mesmo comando, conforme mostrado a seguir

```
$ cat nomes.txt
Maria
Paulo
João
$ sort < nomes.txt > ordenados.txt
$ cat ordenados.txt
João
Maria
Paulo
```

/dev/null

Dica: video sobre TTY e *bit bucket*: https://youtu.be/S81GyMKH7zw

- Existe um dispositivo especial (`/dev/null`) que é uma espécie de sumidouro de dados (*data sink*), comumente conhecido com a lixeira de bits (*bit bucket*)
- Pode-se usar esse dispositivo para redirecionar saídas que não deseja manter ou ver, fazendo com que os dados desapareçam

```
$ echo 'olá mundo' > /dev/null
```

- Se ler do dispositivo, recebe uma *string* vazia; pode ser usado para truncar um arquivo, mantendo suas permissões

```
$ ls -l mensagens.txt
-rw-rw-r-- 1 rimsa rimsa 25315 Nov  1 19:54 mensagens.txt
$ cat /dev/null > mensagens.txt
$ ls -l mensagens.txt
-rw-rw-r-- 1 rimsa rimsa 0 Nov  1 19:54 mensagens.txt
```

PIPES

Pipes

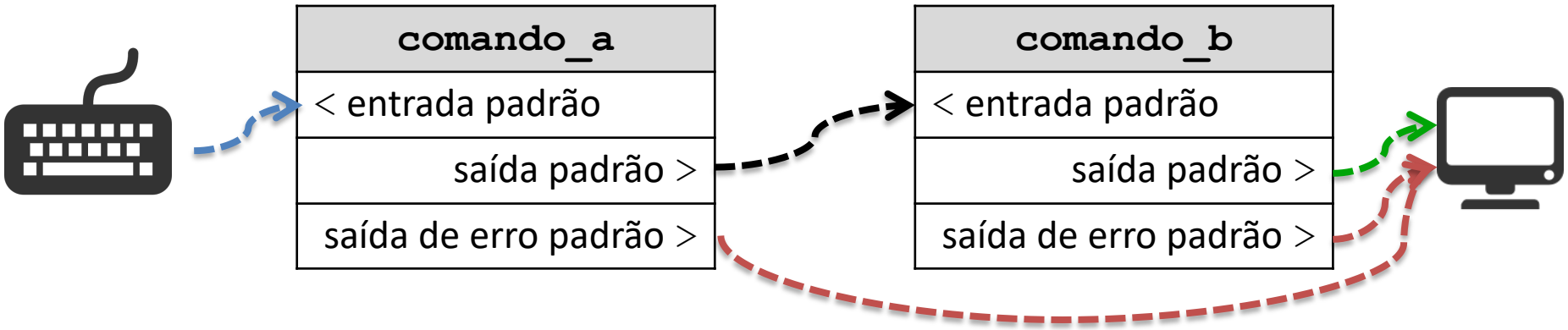
- Muitas vezes é desejável usar a saída de um comando como entrada para um outro comando

```
$ comando_a [argumentos] > temp
$ comando_b [argumentos] < temp
$ rm temp
```

Tem um jeito melhor de fazer isto?

Pipes

- A shell utiliza o símbolo especial | (*pipe*) para conectar a saída padrão de um comando na entrada padrão de um outro comando



- O formato para ligar os comandos é dado por

```
comando_a [argumentos] | comando_b [argumentos]
```

Pode-se conectar vários comandos com *pipes*

Exemplo

- Exibir o conteúdo de um grande diretório por páginas

```
$ ls -l /etc | more
total 688
-rw-r--r--  1 root  wheel     515 Sep  9 17:47 afpovertcp.cfg
-rw-r-----  1 root  wheel   16384 Sep  9 19:37 aliases.db
drwxr-xr-x  6 root  wheel     306 Sep  9 19:11 apache2
drwxr-xr-x  2 root  wheel    1292 Oct 24 17:02 asl
-rw-r--r--  1 root  wheel     975 Sep  9 17:47 asl.conf
-rw-r--r--  1 root  wheel     149 Sep 19 04:15 auto_home
-rw-r--r--  1 root  wheel     194 Sep 19 04:15 auto_master
-rw-r--r--  1 root  wheel    1935 Sep 19 04:15 autofs.conf
-r--r--r--  1 root  wheel     745 Sep 26 23:03 bashrc
-rw-r--r--  1 root  wheel     189 Sep  9 19:40 csh.cshrc
-rw-r--r--  1 root  wheel     121 Sep  9 19:40 csh.login
-rw-r--r--  1 root  wheel      39 Sep  9 19:40 csh.logout
drwxr-xr-x  5 root  lp        306 Sep  9 19:16 cups
drwxr-xr-x  2 root  wheel     102 Sep  9 20:24 defaults
```

Filtros

- Um **filtro** é um comando que processa um fluxo de dados de entrada e produz um fluxo de dados de saída

- Uma linha de comando que inclui um filtro utiliza um *pipe* para conectar a saída padrão de um comando para a entrada padrão do filtro; um outro *pipe* pode conectar a saída padrão do filtro como entrada para outro programa

```
$ ls -l /etc | grep host
-rw-r--r--  1 root root       92 Fev 19  2014 host.conf
-rw-r--r--  1 root root        9 Ago 17 16:43 hostname
-rw-r--r--  1 root root      223 Ago 17 16:43 hosts
-rw-r--r--  1 root root      411 Jul 22 19:17 hosts.allow
-rw-r--r--  1 root root      711 Jul 22 19:17 hosts.deny
```

Saída em Duas Direções

- O utilitário `tee` copia os dados de entrada para um arquivo e para sua saída padrão

O comando `tee` (`T`) foi batizado apropriadamente: um único fluxo de entrada é enviado para a saída em duas direções

- Exemplo que salva as informações do disco para o arquivo `fs.txt` e manda para a saída para ser filtrada pelo comando `grep`

```
$ df -h | tee fs.txt | grep tmp
tmpfs           302M  884K  301M   1% /run
$ cat fs.txt
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        48G  4,1G   42G   9% /
none            4,0K     0  4,0K   0% /sys/fs/cgroup
udev            1,5G  4,0K  1,5G   1% /dev
tmpfs           302M  884K  301M   1% /run
none            5,0M     0  5,0M   0% /run/lock
none            1,5G  152K  1,5G   1% /run/shm
none            100M   36K  100M   1% /run/user
```

Dica: use `-a` com o `tee` para anexar ao arquivo, ao invés de sobreeescrever

Exemplo Real

- Um exemplo complexo: **descobrir as palavras mais frequentes em uma lista de músicas**

```
$ find ./Music -type f -exec basename {} \; | # Obter todos os arquivos
> sed "s/\..*//g" | # Remover a extensão do nome do arquivo
> tr -cs A-Za-z\' '\n' | # Substituir todas as não letras por nova linha
> tr A-Z a-z | # Substituir caracteres maiúsculos por minúsculos
> sort | # Ordenar todas as palavras
> uniq -c | # Contar palavras repetidas
> sort -k1,1nr -k2,2 | # Ordenar pelas palavras mais repetidas
> head -n 25 | # Obter as 25 palavras mais frequentes
> pr -c4 -t -w80 # Imprimir em 4 colunas
  547 the         194 album      157 de          106 in
  219 a           191 unknown    123 you         105 to
  214 o           190 of         116 e           103 i
```

REDIRECIONAMENTOS AVANÇADOS

Shell Scripting

Saída de Erro Padrão

- Além de enviar mensagens para a saída padrão, programas podem enviar mensagens para a saída de erro padrão para evitar misturar as informações de uma saída com a outra

- Assim como a saída padrão, por padrão a *shell* direciona a saída de erro padrão para a tela

- A não ser que direcione uma ou a outra, não é possível distinguir qual das duas saídas foi utilizada

Outros Redirecionamentos

- A *shell* permite selecionar qual descritor de arquivos se deseja redirecionar o fluxo: basta colocar o número do descritor antes do símbolo de redirecionamento (`n<` ou `n>`)
  - **Entrada padrão:** `0<` (abreviado: `<`)
  - **Saída padrão:** `1>` (abreviado: `>`)
  - **Saída de erro:** `2>`

fd	Tipo
0	Entrada padrão
1	Saída padrão
2	Saída de erro padrão

Redirecionando

- Considere a existência de um arquivo `y` com algum conteúdo e a não existência de um arquivo `x`

```
$ cat x
cat: x: No such file or directory
$ cat y
Isso é y.
$ cat x y
cat: x: No such file or directory
Isso é y.
```

- Exemplo redirecionando somente a saída padrão e outro mostrando que o *pipe* conecta somente a saída padrão

```
$ cat x y >output
cat: x: No such file or directory
$ cat output
Isso é y.
```

```
$ cat x y | tr '[a-z]' '[A-Z]'
cat: x: No such file or directory
ISSO é Y.
```

Redirecionando a Saída de Erro Padrão

- Para redirecionar a saída de erro padrão, utilize `2>`

```
$ cat x y 2>errput
Isso é y.
$ cat errput
cat: x: No such file or directory
```

- Pode-se redirecionar cada descritor para um arquivo diferente ao mesmo tempo

```
$ cat x y 1>output 2>errput
$ cat output
Isso é y.
$ cat errput
cat: x: No such file or directory
```

Combinando Saída Padrão com Saída de Erro Padrão

- É possível redirecionar tanto a saída padrão quanto a saída de erro para um mesmo arquivo – usando `&>`

```
$ cat x y &>all
$ cat all
cat: x: No such file or directory
Isso é y.
```

Duplicando Descritores de Arquivos

- É possível duplicar um descritor de arquivos, redirecionando o fluxo de um descritor para outro – usando `n>&m`, tal que `n` é o descritor de onde vem o fluxo e `m` para onde vai

```
$ cat x y 1>output 2>&1
$ cat output
cat: x: No such file or directory
Isso é y.
```

O que aconteceria se a ordem dos redirecionamentos fosse invertida?

- Repare agora como toda a saída é afetada pelo comando `tr`

```
$ cat x y 2>&1 | tr '[a-z]' '[A-Z]'
CAT: X: NO SUCH FILE OR DIRECTORY
ISSO é Y.
```

Enviando Mensagens para a Saída de Erro Padrão

- Um script pode também enviar mensagens para a saída de erro padrão – basta redirecionar `1>&2`

```
#!/bin/sh

echo "Isso é uma mensagem de erro" 1>&2;
echo "Isso não é uma mensagem de erro";
```

msg.sh

Documentos HERE

- Uma outra forma de prover dados de entrada é através de documentos HERE
  - Ao invés de ler de um arquivo (operador <), pode-se ler diretamente da entrada até encontrar um delimitador especificado pelo usuário (<<DELIMITADOR)

Muito útil para usar dentro de *scripts*, já que não precisar usar o CTRL+D

```
$ sort
Maça
Pera
Banana
^D
Banana
Maça
Pera
$ sort << DONE
> Maça
> Pera
> Banana
> DONE
Banana
Maça
Pera
```

Documentos HERE

- O documento HERE processa símbolos especiais utilizados pelo *shell* para por exemplo expandir variáveis com seu conteúdo

```
$ cat << EOF
> Diretório atual: $PWD
> EOF
Diretório atual: /Users/rimsa
```

- Para usar o texto de forma crua, basta usar aspas simples em qualquer parte do delimitador (ou nele todo)

```
$ cat << 'E'OF
> Diretório atual: $PWD
> EOF
Diretório atual: $PWD
```

Documentos HERE

- Pode-se usar o operador <<-DELIMITADOR para remover tabulações no começo das linhas

```
$ cat <<-FIM
>   Texto indentado
> FIM
Texto indentado
```

Resumo

Operador	Significado
`< filename`	Redireciona arquivo para a entrada padrão
`> filename`	Redireciona saída padrão para arquivo sobreescrevendo o conteúdo do arquivo (se *noclobber* tiver ativado e o arquivo já existir, não é sobreescrito)
`>| filename`	Redireciona saída padrão para arquivo sobreescrevendo conteúdo independente da configuração de `noclobber`
`>> filename`	Similar ao operador `> filename`, mas concatena ao arquivo ao invés de sobreescrever
`<<PATTERN`	Documentos `HERE`
`&> filename`	Redireciona saída e saída de erro padrão para arquivo
`<&m`	Duplica entrada padrão de descritor de arquivo `m`
`[n]>&m`	Duplica saída padrão ou descritor de arquivos `n` (se especificado) do descritor de arquivo `m`
`[n]>>&m`	Similar ao operador `[n]>&m`, mas concatena ao invés de sobreescrever
`[n]<&-`	Fecha a entrada padrão ou o descritor de arquivo `n` (se especificado)
`[n]>&-`	Fecha a saída padrão ou o descritor de arquivo `n` (se especificado)
`|`	Envia dados da saída padrão do comando precedente para a entrada padrão do comando subsequente

ISSO É TUDO, PESSOAL!