

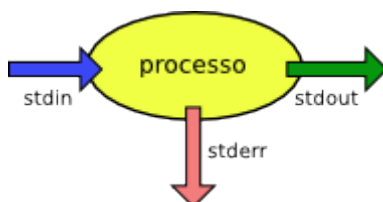
# UNIX: Uso avançado do Shell

Neste texto são apresentados os principais conceitos associados a entradas e saídas padrão, como redirecionamentos e pipes. Também são vistos uma série de programas simples (os filtros), que podem ser muito úteis quando associados através de pipes.

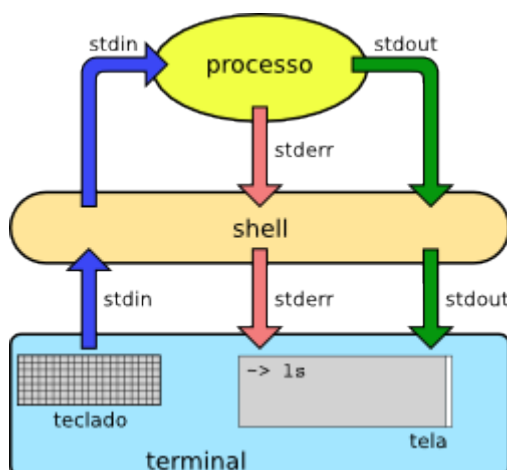
## Entradas e saídas padrão

A maioria dos comandos UNIX pode comunicar-se com o sistema através de descritores de arquivos especiais conhecidos como entradas e saídas padrão. Eles são:

- Entrada padrão (`stdin` - *standard input*): onde o comando vai ler seus dados de entrada. No Bash, esse arquivo é referenciado pelo descritor 0.
- Saída padrão (`stdout` - *standard output*): onde o comando vai escrever seus dados de saída. No Bash, esse arquivo é referenciado pelo descritor 1.
- Saída de erro (`stderr` - *standard error*): onde o comando vai enviar mensagens de erro. No Bash, esse arquivo é referenciado pelo descritor 2.



Quando um comando é lançado sem indicar seu arquivo de trabalho, ele busca seus dados da entrada padrão. Por default, o shell onde o comando foi lançado associa o processo ao seu terminal, ou seja: a entrada padrão do processo é associada ao teclado e as saídas padrão e de erros à tela da sessão corrente.



Um exemplo de uso da entrada e saída padrão é o comando `rev`, que escreve em sua saída padrão as linhas de texto lidas em sua entrada padrão, invertendo-as:

```
$ rev
vamos fazer um teste
etset mu rezaf somav          *
temos que achar um palindromo
omordnilap mu rahca euq somet  *
opoetaamaateopo
```

```
opoetaamaateopo          *
^D
$
```

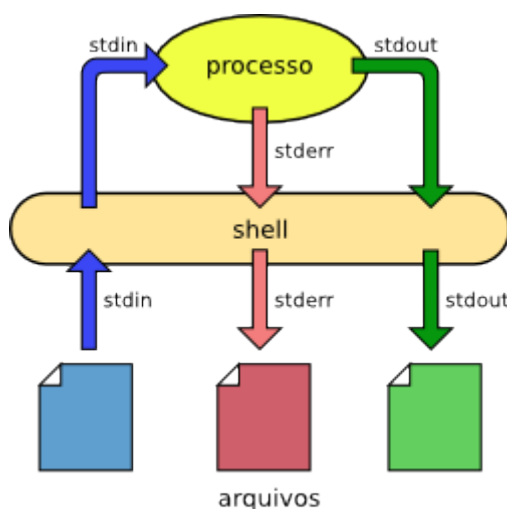
No exemplo, as linhas marcadas com \* indicam as saídas geradas pelo comando rev. O caractere ^D (*Control-D*) no final indica o final da entrada padrão (ou seja, o fim de arquivo). Ao receber esse caractere, o comando rev encerra sua execução, pois chegou ao final de seu arquivo de entrada (que neste caso é o teclado). Outro exemplo de uso da entrada e saída padrão é comando sort:

```
$ sort
joao
maria
antonio
carlos
manoel
^D
antonio carlos joao manoel maria
$
```

Normalmente o shell direciona a entrada padrão para o teclado e a saída padrão para a tela da sessão do usuário, mas ele pode ser instruído para redirecioná-las para arquivos ou mesmo para outros programas, como será visto na sequência.

## Redireção para arquivos

O shell pode redirecionar as entrada e saídas padrão de comandos para arquivos normais no disco, usando operadores de redireção, como mostra a figura abaixo:



A sintaxe de redireção é específica para cada shell, isto é, não é a mesma entre o C-Shell e o Bourne Shell; aqui será apresentada a sintaxe do shell Bash.

Os principais operadores de redireção para arquivos são:

- Saída em arquivo: a saída padrão (*stdout*) do comando é desviada para um arquivo usando o operador >:

```
$ ls > listagem.txt
```

- Entrada de arquivo: a entrada padrão (*stdin*) pode ser obtida a partir de um arquivo usando o operador <:

```
$ rev < listagem.txt
```

- Uso combinado: os dois operadores podem ser usados simultaneamente.

```
$ rev < listagem.txt > listrev.txt
```

- Concatenação: a saída padrão pode ser concatenada a um arquivo existente usando-se o operador >>, como mostra o exemplo:

```
$ ls /etc >> listagem.txt
```

- Saída de erros: a saída de erros (*stderr*) também pode ser redirecionada, através do operador 2> (que faz referência ao descritor 2):

```
$ ls /xpto > teste.txt
ls: /xpto: No such file or directory

$ ll /xpto 2> erro.txt
$ cat error.txt
ls: /xpto: No such file or directory
```

- As saídas padrão e de erro podem ser redirecionadas de forma independente:

```
$ ll /xpto /etc/passwd > acerto.txt 2> erro.txt

$ cat error.txt
ls: /xpto: No such file or directory

$ cat acerto.txt
-rw-r--r-- 1 root root 2136 Mai 14 17:02 /etc/passwd
```

- Além disso, a saída de erro pode ser sobreposta à saída padrão:

```
$ ll /xpto /etc/passwd > acerto.txt 2>&1

$ cat acerto.txt
-rw-r--r-- 1 root root 2136 Mai 14 17:02 /etc/passwd ls: /xpto: No such file or
directory
```

- Forçar um desvio: Caso a saída seja redirecionada para um arquivo já existente, o shell recusa a operação indicando o erro (somente se a variável `noclobber` estiver setada através do comando `set -C`). Essa operação pode ser forçada através do operador !:

```
$ ls > teste.txt
teste.txt: File exists.

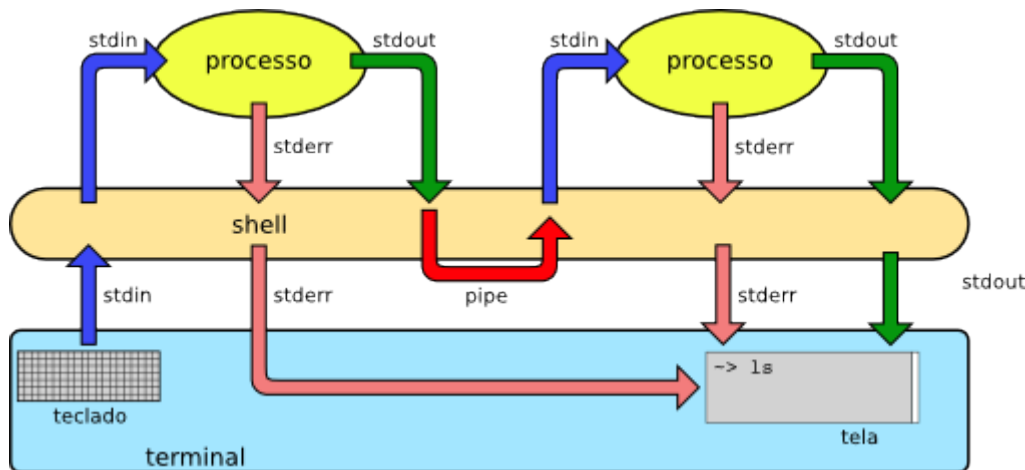
$ ls >! teste.txt

$ ls >> novo.txt
novo.txt: No such file or directory

$ ls >>! novo.txt
```

## Redireção usando pipes

O shell permite a construção de comandos complexos através da combinação de vários comandos simples. O operador `|`, conhecido como *pipe*, ou tubo, permite conectar a saída padrão de um comando à entrada padrão de outro. Com isso, um mesmo fluxo de dados pode ser tratado por diversos comandos consecutivamente, como mostra a figura:



É importante ressaltar que os comandos conectados são lançados simultaneamente pelo shell e executam ao mesmo tempo. O shell controla a execução de cada um para que não haja acúmulo de dados entre os comandos (a cada pipe é associado um buffer de tamanho limitado).

A sintaxe usada para redireção é simples. Eis alguns exemplos:

```
$ ls -l /etc | more
$ ls -l /tmp | sort | more
$ ls -l /usr/bin | cut -c31-40 | sort | more
```

O mecanismo de redireção de entrada/saída é genérico, ou seja, funciona para qualquer programa que use as entradas e saídas padrão, em qualquer linguagem de programação.

## Filtros

Um filtro é basicamente um programa que lê dados da entrada padrão, realiza algum processamento e escreve os dados resultantes na saída padrão. Um exemplo simples de filtro seria:

[filtro.c](#)

```
#include <stdio.h>
#include <stdlib.h>

// lê caracteres em stdin e escreve em stdout, convertendo
// vogais minúsculas em '*' e vogais maiúsculas em '#'.

int main ()
{
    char c ;

    c = getchar () ;
```

```
while (c != EOF)
{
    switch (c)
    {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
            c = '*' ;
            break ;
        case 'A':
        case 'E':
        case 'I':
        case 'O':
        case 'U':
            c = '#' ;
            break ;
    }
    putchar (c) ;
    c = getchar () ;
}
return (0);
}
```

Para compilar esse filtro basta digitar: `gcc -o filtro filtro.c`. Uma vez compilado, o arquivo executável `filtro` pode ser usado na linha de comando UNIX, como qualquer outro filtro.

Existe um grande número de comandos UNIX bastante simples, cujo uso direto é pouco útil, mas que podem ser de grande valia quando associados entre si através de pipes. Esses comandos são chamados filtros, porque funcionam como filtros para o fluxo de dados. Eis alguns filtros de uso corrente:

- `cat` : concatena diversos arquivos na saída padrão
- `tac` : idem, mas inverte a ordem das linhas
- `more` : permite a paginação do fluxo de dados
- `tr` : troca de caracteres entre dois conjuntos
- `head` : seleciona as *n* linhas iniciais do fluxo de dados
- `tail` : seleciona as *n* linhas finais do fluxo de dados
- `wc` : conta o número de linhas, palavras e bytes do fluxo
- `sort` : ordena as linhas segundo critérios ajustáveis
- `uniq` : remove linhas repetidas, deixando uma só linha
- `sed` : para operações complexas de strings (trocas, etc)
- `grep` : seleciona linhas contendo uma determinada expressão
- `cut` : seleciona colunas do fluxo de entrada
- `rev` : reverte a ordem dos caracteres de cada linha do fluxo de entrada
- `tee` : duplica o fluxo de entrada (para um arquivo e para a saída standard)
- ... : qualquer programa que leia dados de `stdin` e escreva sua saída em `stdout` pode ser usado como filtro

Para conhecer melhor cada um dos comandos acima, basta consultar suas respectivas páginas de manual.

## Exercícios

1. Usando comandos e *pipes*, determine o número de linhas da página de manual do shell Bash.
2. Determine quanto arquivos normais (não diretórios nem links) existem em `/usr`.

3. Monte uma linha de comandos usando *pipes* para identificar todos os usuários proprietários de arquivos ou diretórios a partir de `/tmp`, colocando o resultado no arquivo `users-tmp.txt`. Siga os seguintes passos:
  - Use o comando `find` para listar os proprietários de todos os arquivos dentro de `/tmp` (dica: use a opção `-printf` do comando `find`).
  - Ordene a listagem obtida, usando o comando `sort`
  - Remova as linhas repetidas, usando o comando `uniq`
  - Direcione a saída para o arquivo indicado `users-tmp.txt`.
4. Use o comando `cut` na saída de um comando `ls -l` para mostrar apenas as permissões dos arquivos no diretório `/etc`. Depois use `sort` e `uniq` para mostrar quantas permissões diferentes existem naquele diretório.
5. Quantos arquivos invisíveis (iniciados com `.`) há na sua área HOME?
6. Quantos diretórios há na sua área HOME?
7. Liste todos os atributos de todos os arquivos de um diretório e utilize o `cut` para mostrar apenas suas permissões e seu nome.
8. Liste todos os arquivos e seus atributos (somente os arquivos, diretórios não devem aparecer) do diretório `/etc`, ordenando a saída por data do arquivo, e guarde a saída no arquivo `teste.txt` na sua área.
9. Mostre apenas o vigésimo arquivo do diretório `/etc`
10. Mostre apenas os arquivos e diretórios para os quais você tem permissão de execução na sua área HOME.
11. Acesse o servidor `ssh.inf.ufpr.br`. Utilize o comando `finger` para mostrar o Login de todos usuários cujo primeiro nome seja Daniel.
12. Execute os comandos a seguir como usuário normal. Determine o que é `stdin`, `stdout` e `stderr` para cada comando (o conteúdo de cada fluxo para cada comando):
  1. `cat nonexistentfile`
  2. `file /sbin/ifconfig`
  3. `grep root /etc/passwd /etc/nofiles > grepresults`
  4. `/etc/init.d/sshd start > /var/tmp/output`
  5. `/etc/init.d/crond start > /var/tmp/output 2>&1`
  6. Confira seu resultado repetindo os comandos e atribuindo `stdout` para `$HOME/saida.txt` e `stderr` para `$HOME/erro.txt`.
13. Observe as seguintes sequências de comandos e responda às perguntas:

```
$ mkdir vazio
$ cd vazio
$ cp a b
cp: cannot stat 'a': No such file or directory
$ cp a b >a
```

1. Por que não há mensagem de erro após o segundo comando `cp`? Qual o conteúdo do arquivo `a`?

```
$ date >a
$ cat a
Wed Feb  8 03:01:21 EST 2012

$ cp a b
$ cat b
Wed Feb  8 03:01:21 EST 2012

$ cp a b >a
$ cat b
```

2. Por que o arquivo `b` está vazio? O que há no arquivo `a`?

From:

<https://wiki.inf.ufpr.br/maziero/> - **Prof. Carlos Maziero**

Permanent link:

[https://wiki.inf.ufpr.br/maziero/doku.php?id=unix:shell\\_avancado](https://wiki.inf.ufpr.br/maziero/doku.php?id=unix:shell_avancado)

Last update: **2020/08/18 18:59**

