# PIPES, REDIRECIONAMENTO E REGEX

## 8.1 Introdução

Um grande número de arquivos em um sistema de arquivos típico são *arquivos de texto*. Os arquivos de texto contêm simplesmente texto, sem recursos de formatação que você possa ver em um arquivo de processamento de texto.

Como há muitos desses arquivos em um sistema Linux típico, existe um grande número de comandos para ajudar os usuários a manipular arquivos de texto. Existem comandos para visualizar e modificar esses arquivos de várias maneiras.

Além disso, existem recursos disponíveis para o shell controlar a saída de comandos, portanto, em vez de ter a saída colocada na janela do terminal, a saída pode ser *redirecionada* para outro arquivo ou outro comando. Esses recursos de redirecionamento fornecem aos usuários um ambiente muito mais flexível e poderoso para trabalhar.

# 8.2 Pipes de linha de comando

Os capítulos anteriores discutiram como usar comandos individuais para executar ações no sistema operacional, incluindo como criar / mover / excluir arquivos e se mover pelo sistema. Normalmente, quando um comando gera ou gera um erro, a saída é exibida na tela; no entanto, isso não precisa ser o caso.

O caractere *pipe* | pode ser usado para enviar a saída de um comando para outro. Em vez de ser impressa na tela, a saída de um comando se torna entrada para o próximo comando. Essa pode ser uma ferramenta poderosa, especialmente ao procurar por dados específicos; o piping é usada frequentemente para refinar os resultados de um comando inicial.

Os comandos `head` e `tail` serão usados em muitos exemplos abaixo para ilustrar o uso de pipes. Esses comandos podem ser usados para exibir apenas as primeiras ou últimas linhas de um arquivo (ou, quando usado com um pipe, a saída de um comando anterior).

Por padrão, os comandos `head` e `tail` exibirão dez linhas. Por exemplo, o seguinte comando exibirá as dez primeiras linhas do arquivo `/etc/sysctl.conf`:

```
sysadmin@localhost:~$ head /etc/sysctl.conf
#
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables
# See sysctl.conf (5) for information.
#


#kernel.domainname = example.com


# Uncomment the following to stop low-level messages on console
#kernel.printk = 3 4 1 3
sysadmin@localhost:~$
```

No próximo exemplo, as últimas dez linhas do arquivo serão exibidas:

```
sysadmin@localhost:~$ tail /etc/sysctl.conf
# Do not send ICMP redirects (we are not a router)
#net.ipv4.conf.all.send_redirects = 0
#
# Do not accept IP source route packets (we are not a router)
#net.ipv4.conf.all.accept_source_route = 0
#net.ipv6.conf.all.accept_source_route = 0
#
# Log Martian Packets
#net.ipv4.conf.all.log_martians = 1
```

```
#


sysadmin@localhost:~$
```

O caractere pipe permite que os usuários utilizem esses comandos não apenas em arquivos, mas na saída de outros comandos. Isso pode ser útil ao listar um diretório grande, por exemplo, o diretório `/etc`:

```
ca-certificates         insserv              nanorc          services
ca-certificates.conf    insserv.conf         network         sgml
calendar                insserv.conf.d       networks        shadow
cron.d                  iproute2             nologin         shadow-
cron.daily              issue                nsswitch.conf   shells
cron.hourly             issue.net            opt             skel
cron.monthly            kernel               os-release      ssh
cron.weekly             ld.so.cache          pam.conf        ssl
crontab                 ld.so.conf           pam.d           sudoers
dbus-1                  ld.so.conf.d         passwd          sudoers.d
debconf.conf            ldap                 passwd-         sysctl.co
nf
debian_version          legal                perl            sysctl.d
default                 locale.alias         pinforc         systemd
deluser.conf            localtime            ppp             terminfo
depmod.d                logcheck             profile         timezone
dpkg                    login.defs           profile.d       ucf.conf
environment             logrotate.conf       protocols       udev
fstab                   logrotate.d          python2.7       ufw
fstab.d                 lsb-base             rc.local        update-mo
td.d
gai.conf                lsb-base-logging.sh  rc0.d           updatedb.
conf
groff                   lsb-release          rc1.d           vim
group                   magic                rc2.d           wgetrc
group-                  magic.mime           rc3.d           xml
sysadmin@localhost:~$
```

Se você observar a saída do comando anterior, você notará que o primeiro nome de arquivo é `ca-certificates`. Mas há outros arquivos listados "acima" que só podem ser visualizados se o usuário usar a barra de rolagem. E se você apenas quisesse listar os primeiros arquivos do diretório `/etc`?

Em vez de exibir a saída completa do comando acima, fazendo o piping para o comando `head` exibirá apenas as dez primeiras linhas:

```
sysadmin@localhost:~$ ls /etc | head
adduser.conf
adjtime
alternatives
apparmor.d
apt
bash.bashrc
bash_completion.d
bind
bindresvport.blacklist
blkid.conf
sysadmin@localhost:~$
```

A saída completa do comando `ls` é passada para o comando `head` pelo shell, em vez de ser impressa na tela. O comando `head` toma essa saída (de `ls`) como "dados de entrada" e a saída da cabeça é então impressa na tela.

Vários pipes podem ser usados consecutivamente para vincular vários comandos juntos. Se três comandos forem conectados juntos, a saída do primeiro comando será passada para o segundo comando. A saída do segundo comando é então passada para o terceiro comando. A saída do terceiro comando seria então impressa na tela.

É importante escolher cuidadosamente a ordem na qual os comandos são canalizados, pois o terceiro comando só verá a entrada da saída do segundo. Os exemplos abaixo ilustram isso usando o comando `nl`. No primeiro exemplo, o comando `nl` é usado para numerar as linhas da saída de um comando anterior:

```
sysadmin@localhost:~$ ls -l /etc/ppp | nl
1  total 44
     2  -rw------- 1 root root   78 Aug 22  2010 chap-secrets
     3  -rwxr-xr-x 1 root root  386 Apr 27  2012 ip-down
     4  -rwxr-xr-x 1 root root 3262 Apr 27  2012 ip-down.ipv6to4
     5  -rwxr-xr-x 1 root root  430 Apr 27  2012 ip-up
     6  -rwxr-xr-x 1 root root 6517 Apr 27  2012 ip-up.ipv6to4
     7  -rwxr-xr-x 1 root root 1687 Apr 27  2012 ipv6-down
     8  -rwxr-xr-x 1 root root 3196 Apr 27  2012 ipv6-up
     9  -rw-r--r-- 1 root root    5 Aug 22  2010 options
    10  -rw------- 1 root root   77 Aug 22  2010 pap-secrets
    11  drwxr-xr-x 2 root root 4096 Jun 22  2012 peers
sysadmin@localhost:~$
```

No próximo exemplo, observe que o comando `ls` é executado primeiro e sua saída é enviada para o comando `nl`, numerando todas as linhas da saída do comando `ls`. Então o comando `tail` é executado, exibindo as últimas cinco linhas da saída do comando `nl`:

```
sysadmin@localhost:~$ ls -l /etc/ppp | nl | tail -5
     7  -rwxr-xr-x 1 root root 1687 Apr 27  2012 ipv6-down
     8  -rwxr-xr-x 1 root root 3196 Apr 27  2012 ipv6-up
     9  -rw-r--r-- 1 root root    5 Aug 22  2010 options
    10  -rw------- 1 root root   77 Aug 22  2010 pap-secrets
    11  drwxr-xr-x 2 root root 4096 Jun 22  2012 peers
sysadmin@localhost:~$
```

Compare a saída acima com o próximo exemplo:

```
sysadmin@localhost:~$ ls -l /etc/ppp | tail -5 | nl
    1  -rwxr-xr-x 1 root root 1687 Apr 27  2012 ipv6-down
    2  -rwxr-xr-x 1 root root 3196 Apr 27  2012 ipv6-up
    3  -rw-r--r-- 1 root root    5 Aug 22  2010 options
    4  -rw------- 1 root root   77 Aug 22  2010 pap-secrets
    5  drwxr-xr-x 2 root root 4096 Jun 22  2012 peers
sysadmin@localhost:~$
```

Observe como os números de linha são diferentes. Por que é isso?

No segundo exemplo, a saída do comando `ls` é enviada primeiro ao comando `tail` que "agarra" somente as últimas cinco linhas da saída. Em seguida, o comando `tail` envia essas cinco linhas para o comando `nl`, que os números `1-5`.

Os pipess podem ser poderosos, mas é importante considerar como os comandos são ordenados no piping para garantir que a saída desejada seja exibida.

# 8.3 I/O Redirecionamento

O redirecionamento de entrada/saída (E / S) permite que as informações da linha de comando sejam transmitidas para *fluxos*. Antes de discutir o redirecionamento, é importante entender os fluxos padrão.

# 8.3.1 STDIN

Entrada padrão, ou STDIN, é uma informação inserida normalmente pelo usuário através do teclado. Quando um comando solicita dados ao shell, o shell fornece ao usuário a capacidade de digitar comandos que, por sua vez, são enviados para o comando como STDIN.

# 8.3.2 STDOUT

Saída padrão, ou STDOUT, é a saída normal dos comandos. Quando um comando funciona corretamente (sem erros), a saída que produz é chamada STDOUT. Por padrão, STDOUT é exibido na janela do terminal (tela) onde o comando está sendo executado.

# 8.3.3 STDERR

Erro padrão, ou STDERR, são mensagens de erro geradas por comandos. Por padrão, STDERR é exibido na janela do terminal (tela) onde o comando está sendo executado.

O redirecionamento de E/S permite que o usuário redirecione STDIN para que os dados provenham de um arquivo e STDOUT/STDERR para que a saída vá para um arquivo. O redirecionamento é obtido usando os caracteres < e >.

# 8.3.4 Redirecionando STDOUT

STDOUT pode ser direcionado para arquivos. Para começar, observe a saída do seguinte comando que será exibido na tela:

```
sysadmin@localhost:~$ echo "Line 1"
Line 1
sysadmin@localhost:~$
```

Usando o caractere > a saída pode ser redirecionada para um arquivo:

```
sysadmin@localhost:~$ echo "Line 1" > example.txt
sysadmin@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  example.txt  test
Documents  Music      Public    Videos     sample.txt
sysadmin@localhost:~$ cat example.txt
Line 1
sysadmin@localhost:~$
```

Este comando não exibe saída, porque o STDOUT foi enviado para o arquivo `example.txt` em vez da tela. Você pode ver o novo arquivo com a saída do comando `ls`. O arquivo recém-criado contém a saída do comando `echo` quando o arquivo é exibido com o comando `cat`.

É importante perceber que um único caractere > irá sobrescrever qualquer conteúdo de um arquivo existente:

```
sysadmin@localhost:~$ cat example.txt
Line 1
sysadmin@localhost:~$ echo "New line 1" > example.txt
sysadmin@localhost:~$ cat example.txt
New line 1
sysadmin@localhost:~$
```

O conteúdo original do arquivo desapareceu, substituído pela saída do novo comando `echo`.

Também é possível preservar o conteúdo de um arquivo existente anexando a ele. Use dois caracteres >> para anexar a um arquivo em vez de substituí-lo:

```
sysadmin@localhost:~$ cat example.txt
New line 1
sysadmin@localhost:~$ echo "Another line" >> example.txt
sysadmin@localhost:~$ cat example.txt
New line 1
Another line
sysadmin@localhost:~$
```

Em vez de ser sobrescrito, a saída do comando `echo` mais recente é adicionada à parte inferior do arquivo.

# 8.3.5 Redirecionando STDERR

O STDERR pode ser redirecionado de maneira semelhante ao STDOUT. STDOUT também é conhecido como fluxo (ou canal) #1. STDERR recebe o fluxo #2.

Ao usar os caracteteres de redirecionamento, o fluxo #1 é assumido, a menos que outro fluxo seja especificado. Portanto, o fluxo #2 deve ser especificado ao redirecionar o STDERR.

Para demonstrar o redirecionamento de STDERR, primeiro observe o seguinte comando que produzirá um erro porque o diretório especificado não existe:

```
sysadmin@localhost:~$ ls /fake
ls: cannot access /fake: No such file or directory
sysadmin@localhost:~$
```

Note que não há nada no exemplo acima que implique que a saída seja STDERR. A saída é claramente uma mensagem de erro, mas como você pode dizer que está sendo enviado para o STDERR? Uma maneira fácil de determinar isso é redirecionar o STDOUT:

```
sysadmin@localhost:~$ ls /fake > output.txt
ls: cannot access /fake: No such file or directory
sysadmin@localhost:~$
```

No exemplo acima, STDOUT foi redirecionado para o arquivo `output.txt`. Portanto, a saída exibida não pode ser STDOUT, porque ela teria sido colocada no arquivo `output.txt`. Como toda a saída do comando vai para STDOUT ou STDERR, a saída exibida acima deve ser STDERR.

A saída STDERR de um comando pode ser enviada para um arquivo:

```
sysadmin@localhost:~$ ls /fake 2> error.txt
sysadmin@localhost:~$ cat error.txt
ls: cannot access /fake: No such file or directory
sysadmin@localhost:~$
```

No comando acima, o `2>` indica que todas as mensagens de erro devem ser enviadas para o arquivo error.txt.

# 8.3.6 Redirecionando múltiplos fluxos

É possível direcionar o STDOUT e o STDERR de um comando ao mesmo tempo. O comando a seguir produzirá STDOUT e STDERR, porque um dos diretórios especificados existe e o outro não:

```
sysadmin@localhost:~$ ls /fake /etc/ppp
ls: cannot access /fake: No such file or directory
/etc/ppp:
chap-secrets    ip-down    ip-down.ipv6to4    ip-up         ip-up.ipv6to4
ipv6-down       ipv6-up    options            pap-secrets   peers
```

Se apenas o STDOUT for enviado para um arquivo, o STDERR ainda será impresso na tela:

```
sysadmin@localhost:~$ ls /fake /etc/ppp > example.txt
ls: cannot access /fake: No such file or directory
sysadmin@localhost:~$ cat example.txt
/etc/ppp:
chap-secrets
ip-down
ip-down.ipv6to4
ip-up
ip-up.ipv6to4
ipv6-down
ipv6-up
options
pap-secrets
peers
sysadmin@localhost:~$
```

Se apenas o STDERR for enviado para um arquivo, STDOUT ainda será impresso na tela:

```
sysadmin@localhost:~$ ls /fake /etc/ppp 2> error.txt
/etc/ppp:
hap-secrets    ip-down    ip-down.ipv6to4    ip-up         ip-up.ipv6to4
ipv6-down      ipv6-up    options            pap-secrets   peers
sysadmin@localhost:~$ cat error.txt
ls: cannot access /fake: No such file or directory
sysadmin@localhost:~$
```

Tanto STDOUT quanto STDERR podem ser enviados para um arquivo usando `&>`, um conjunto de caracteres que significa "ambos `1>` e `2>`":

```
sysadmin@localhost:~$ ls /fake /etc/ppp &> all.txt
sysadmin@localhost:~$ cat all.txt
ls: cannot access /fake: No such file or directory
/etc/ppp:
chap-secrets
ip-down
ip-down.ipv6to4
ip-up
ip-up.ipv6to4
ipv6-down
ipv6-up
options
pap-secrets
peers
sysadmin@localhost:~$
```

Note que quando você usa `&>`, a saída aparece no arquivo com todas as mensagens STDERR no topo e todas as mensagens STDOUT abaixo de todas as mensagens STDERR:

```
sysadmin@localhost:~$ ls /fake /etc/ppp /junk /etc/sound &> all.txt
sysadmin@localhost:~$ cat all.txt
ls: cannot access /fake: No such file or directory
ls: cannot access /junk: No such file or directory
/etc/ppp:
chap-secrets
ip-down
ip-down.ipv6to4
ip-up
ip-up.ipv6to4
ipv6-down
ipv6-up
options
pap-secrets
peers

/etc/sound:
events
sysadmin@localhost:~$
```

Se você não quiser que STDERR e STDOUT acessem o mesmo arquivo, eles podem ser redirecionados para arquivos diferentes usando tanto `>` quanto `2>`. Por exemplo:

```
sysadmin@localhost:~$ rm error.txt example.txt
sysadmin@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  all.txt
Documents  Music      Public    Videos
sysadmin@localhost:~$ ls /fake /etc/ppp > example.txt 2> error.txt
sysadmin@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  all.txt    example.txt
Documents  Music      Public    Videos     error.txt
sysadmin@localhost:~$ cat error.txt
ls: cannot access /fake: No such file or directory
sysadmin@localhost:~$ cat example.txt
/etc/ppp:
chap-secrets
ip-down
ip-down.ipv6to4
ip-up
ip-up.ipv6to4
ipv6-down
ipv6-up
options
pap-secrets
peers
sysadmin@localhost:~$
```

A ordem em que os fluxos são especificados não importa.

# 8.3.7 Redirecionando STDIN

O conceito de redirecionar STDIN é difícil porque é mais difícil entender por que você deseja redirecionar STDIN. Com STDOUT e STDERR, a resposta para isso é bastante fácil: porque às vezes você deseja armazenar a saída em um arquivo para uso futuro.

A maioria dos usuários de Linux acaba redirecionando STDOUT rotineiramente, STDERR ocasionalmente e STDIN... bem, muito raramente. Existem muito poucos comandos que requerem que você redirecione STDIN porque, com a maioria dos comandos, se você quiser ler os dados de um arquivo em um comando, você pode simplesmente especificar o nome do arquivo como um argumento para o comando. O comando então examinará o arquivo.

Para alguns comandos, se você não especificar um nome de arquivo como argumento, eles serão revertidos para o uso de STDIN para obter dados. Por exemplo, considere o seguinte comando `cat`:

```
sysadmin@localhost:~$ cat
hello
hello
how are you?
how are you?
goodbye
goodbye
sysadmin@localhost:~$
```

No exemplo acima, o comando `cat` não recebeu um nome de arquivo como argumento. Por isso, pediu que os dados fossem exibidos na tela a partir de STDIN. O usuário digitou `hello` e, em seguida, o comando `cat` exibiu `hello` na tela. Talvez isso seja útil para pessoas solitárias, mas não é realmente um bom uso do comando `cat`.

No entanto, talvez se a saída do comando `cat` fosse redirecionada para um arquivo, esse método poderia ser usado para incluir em um arquivo existente ou para colocar um texto em um novo arquivo:

```
sysadmin@localhost:~$ cat > new.txt
Hello
How are you?
Goodbye
sysadmin@localhost:~$ cat new.txt
Hello
How are you?
Goodbye
sysadmin@localhost:~$
```

Embora o exemplo anterior demonstre outra vantagem de redirecionar o STDOUT, ele não aborda por que ou como o STDIN pode ser direcionado. Para entender isso, primeiro considere um novo comando chamado `tr`. Este comando irá pegar um conjunto de caracteres e traduzi-los em outro conjunto de caracteres.

Por exemplo, suponha que você queira capitalizar uma linha de texto. Você poderia usar o comando `tr` da seguinte maneira:

```
sysadmin@localhost:~$ tr 'a-z' 'A-Z'
watch how this works
WATCH HOW THIS WORKS
sysadmin@localhost:~$
```

O comando `tr` pegou o STDIN do teclado (observe como isso funciona) e converteu todas as letras minúsculas antes de enviar STDOUT para a tela.

Parece que um melhor uso do comando `tr` seria executar a tradução em um arquivo, não a entrada do teclado. No entanto, o comando `tr` não suporta argumentos de nome de arquivo:

```
sysadmin@localhost:~$ more example.txt
/etc/ppp:
chap-secrets
ip-down
ip-down.ipv6to4
ip-up
ip-up.ipv6to4
ipv6-down
ipv6-up
options
pap-secrets
peers
sysadmin@localhost:~$ tr 'a-z' 'A-Z' example.txt
tr: extra operand `example.txt'
Try `tr --help' for more information
sysadmin@localhost:~$
```

Você pode, no entanto, dizer ao shell para obter STDIN de um arquivo em vez de usar o caractere < caractere:

```
sysadmin@localhost:~$ tr 'a-z' 'A-Z' < example.txt
/ETC/PPP:
CHAP-SECRETS
IP-DOWN
IP-DOWN.IPV6TO4
IP-UP
IP-UP.IPV6TO4


IPV6-DOWN
IPV6-UP
OPTIONS
PAP-SECRETS
sysadmin@localhost:~$
```

Isso é bastante raro porque a maioria dos comandos aceita nomes de arquivos como argumentos. Mas, para aqueles que não o fazem, esse método pode ser usado para que o shell seja lido no arquivo, em vez de depender do comando para ter essa capacidade.

Uma última nota: na maioria dos casos, você provavelmente quer pegar a saída resultante e colocá-la de volta em outro arquivo:

```
sysadmin@localhost:~$ tr 'a-z' 'A-Z' < example.txt > newexample.txt
sysadmin@localhost:~$ more newexample.txt
/ETC/PPP:
CHAP-SECRETS
IP-DOWN
IP-DOWN.IPV6TO4
IP-UP
IP-UP.IPV6TO4
IPV6-DOWN
IPV6-UP
OPTIONS
PAP-SECRETS
sysadmin@localhost:~$
```

# 8.4 Procurando por arquivos usando o comando Find

Um dos desafios que os usuários enfrentam ao trabalhar com o sistema de arquivos é tentar recuperar o local onde os arquivos estão armazenados. Existem milhares de arquivos e centenas de diretórios em um sistema de arquivos típico do Linux, portanto, lembrar onde esses arquivos estão localizados pode representar desafios.

Tenha em mente que a maioria dos arquivos com os quais você trabalhará são aqueles criados por você. Como resultado, você frequentemente procurará em seu próprio diretório pessoal para localizar arquivos. No entanto, às vezes você pode precisar procurar em outros lugares no sistema de arquivos para localizar arquivos criados por outros usuários.

O comando `find` é uma ferramenta muito poderosa que você pode usar para procurar arquivos no sistema de arquivos. Esse comando pode procurar arquivos por nome, incluindo o uso de caracteres curinga para quando você não tiver certeza do nome exato do arquivo. Além disso, você pode pesquisar arquivos com base em metadados de arquivo, como tipo de arquivo, tamanho de arquivo e propriedade de arquivo.

A sintaxe do comando `find` é:

```
find [diretório inicial] [opções de busca] [critérios de busca] [opçõe
s de resultado]
```

A description of all of these components:

| Componente | Descrição |
|---|---|
| [diretório inicial] | É aqui que o usuário especifica onde começar a pesquisar. O comando `find` irá procurar neste diretório e em todos os seus subdiretórios. Se nenhum diretório inicial for fornecido, o diretório atual será usado para o ponto inicial. |
| [opções de pesquisa] | É aqui que o usuário especifica uma opção para determinar que tipo de metadados pesquisar; Existem opções para o nome do arquivo, tamanho do arquivo e muitos outros atributos de arquivo. |
| [critérios de pesquisa] | Este é um argumento que complementa a opção de pesquisa. Por exemplo, se o usuário usar a opção para procurar um nome de arquivo, o critério de pesquisa será o nome do arquivo. |
| [opções de resultado] | Essa opção é usada para especificar qual ação deve ser executada quando o arquivo for encontrado. Se nenhuma opção for fornecida, o nome do arquivo será impresso em STDOUT. |

# 8.4.1 Pesquisa por nome de arquivo

Para procurar um arquivo por nome, use a opção `-name` para o comando `find`:

```
sysadmin@localhost:~$ find /etc -name hosts
find: `/etc/dhcp': Permission denied
find: `/etc/cups/ssl': Permission denied
find: `/etc/pki/CA/private': Permission denied
find: `/etc/pki/rsyslog': Permission denied
find: `/etc/audisp': Permission denied
find: `/etc/named': Permission denied
find: `/etc/lvm/cache': Permission denied
find: `/etc/lvm/backup': Permission denied
find: `/etc/lvm/archive': Permission denied
/etc/hosts
find: `/etc/ntp/crypto': Permission denied
find: `/etc/polkit-1/localauthority': Permission denied
find: `/etc/sudoers.d': Permission denied
find: `/etc/sssd': Permission denied
/etc/avahi/hosts
find: `/etc/selinux/targeted/modules/active': Permission denied
find: `/etc/audit': Permission denied

sysadmin@localhost:~$
```

Note que dois arquivos foram encontrados: `/etc/hosts` e `/etc/avahi/hosts`. O restante da saída foi mensagens STDERR porque o usuário que executou o comando não tinha permissão para acessar determinados subdiretórios.

Lembre-se de que você pode redirecionar o STDERR para um arquivo para que você não precise ver essas mensagens de erro na tela:

```
sysadmin@localhost:~$ find /etc -name hosts 2> errors.txt
/etc/hosts
/etc/avahi.hosts
sysadmin@localhost:~$
```

Embora a saída seja mais fácil de ler, realmente não há propósito em armazenar as mensagens de erro no arquivo `error.txt`. Os desenvolvedores do Linux perceberam que seria bom ter um "arquivo de lixo" para enviar dados desnecessários; qualquer arquivo que você envia para o arquivo `/dev/null` é descartado:

```
sysadmin@localhost:~$ find /etc -name hosts 2> /dev/null
/etc/hosts
/etc/avahi/hosts
sysadmin@localhost:~$
```

# 8.4.2 Exibindo o Detalhe do Arquivo

Pode ser útil obter detalhes do arquivo ao usar o comando `find`, porque apenas o nome do arquivo pode não ser suficiente para você encontrar o arquivo correto.

Por exemplo, pode haver sete arquivos chamados *hosts*; Se você soubesse que o arquivo *host* que você precisava foi modificado recentemente, o registro de data e hora da modificação do arquivo seria útil para ver.

Para ver esses detalhes do arquivo, use a opção `-ls` para o comando `find`:

```
sysadmin@localhost:~$ find /etc -name hosts -ls 2> /dev/null
    41    4 -rw-r--r--   1 root     root      158 Jan 12 2010 /etc/host
s
  6549    4 -rw-r--r--   1 root     root      1130 Jul 19 2011 /etc/ava
hi/hosts
sysadmin@localhost:~$
```

**Nota:** As duas primeiras colunas da saída acima são o número do inode do arquivo e o número de blocos que o arquivo está usando para armazenamento. Ambos estão além do escopo do tópico em questão. O restante das colunas são saídas típicas do comando `ls -l`: tipo de arquivo, permissões, contagem de link físico, proprietário do usuário, proprietário do grupo, tamanho do arquivo, registro de data e hora da modificação e nome do arquivo.

# 8.4.3 Procurando arquivos por tamanho

Uma das muitas opções úteis de pesquisa é a opção que permite pesquisar arquivos por tamanho. A opção `-size` permite pesquisar arquivos maiores ou menores que um tamanho especificado, além de procurar um tamanho de arquivo exato.

Quando você especifica um tamanho de arquivo, pode dar o tamanho em bytes (c), kilobytes (k), megabytes (M) ou gigabytes (G). Por exemplo, o seguinte irá procurar por arquivos na estrutura de diretório / etc com exatamente 10 bytes:

```
sysadmin@localhost:~$ find /etc -size 10c -ls 2>/dev/null
   432    4 -rw-r--r--   1 root     root           10 Jan 28  2015 /et
c/adjtime
 8814    0 drwxr-xr-x   1 root     root           10 Jan 29  2015 /etc
/ppp/ip-d
own.d


8816    0 drwxr-xr-x   1 root     root           10 Jan 29  2015 /etc/
ppp/ip-u
p.d
 8921    0 lrwxrwxrwx   1 root     root           10 Jan 29  2015 /etc
/ssl/cert
s/349f2832.0 -> EC-ACC.pem
  9234    0 lrwxrwxrwx   1 root     root           10 Jan 29  2015 /et
c/ssl/cert
s/aeb67534.0 -> EC-ACC.pem
 73468    4 -rw-r--r--   1 root     root           10 Nov 16 20:42 /et
c/hostname
sysadmin@localhost:~$
```

Se você deseja pesquisar arquivos maiores que um tamanho especificado, coloque um caractere + antes do tamanho. Por exemplo, o seguinte procurará todos os arquivos na estrutura de diretório `/usr` com mais de 100 megabytes de tamanho:

```
sysadmin@localhost:~$ find /usr -size +100M -ls 2> /dev/null
574683 104652 -rw-r--r--   1 root      root      107158256 Aug  7 11:0
6 /usr/share/icons/oxygen/icon-theme.cache
sysadmin@localhost:~$
```

Para procurar arquivos menores que um tamanho especificado, coloque um caractere – antes do tamanho do arquivo.

# 8.4.4 Opções adicionais de pesquisa úteis

Existem muitas opções de pesquisa. A tabela a seguir ilustra algumas dessas opções:

| Opção | Significado |
|---|---|
| `-maxdepth` | Permite que o usuário especifique a profundidade da estrutura de diretório a ser pesquisada. Por exemplo, `-maxdepth 1` significaria apenas pesquisar o diretório especificado e seus subdiretórios imediatos. |
| `-group` | Retorna arquivos pertencentes a um grupo especificado. Por exemplo, `-group payroll` retornaria arquivos pertencentes ao grupo payroll. |
| `-iname` | Retorna arquivos que correspondem ao nome de arquivo especificado, mas diferentemente de `-name`, `-iname` não faz distinção entre maiúsculas e minúsculas. Por exemplo, `-iname hosts` corresponderiam os arquivos denominados `hosts`, `Hosts`, `HOSTS`, etc. |
| `-mmin` | Retorna arquivos que foram modificados com base no tempo de modificação em minutos. Por exemplo, `-mmin 10` corresponderia aos arquivos que foram modificados há 10 minutos. |
| `-type` | Retorna arquivos que correspondem ao tipo de arquivo. Por exemplo, `-type f` retornaria arquivos que são arquivos regulares. |
| `-user` | Retorna arquivos pertencentes a um usuário especificado. Por exemplo, `-user bob` retornaria arquivos pertencentes ao usuário `bob`. |

# 8.4.5 Usando várias opções

Se você usar várias opções, elas funcionam como um "*and*", o que significa que para que uma correspondência ocorra, todos os critérios devem corresponder, não apenas um. Por exemplo, o comando a seguir exibirá todos os arquivos na estrutura de diretório `/etc` com 10 bytes de tamanho e arquivos simples:

```
sysadmin@localhost:~$ find /etc -size 10c -type f -ls 2>/dev/null


432    4 -rw-r--r--   1 root     root           10 Jan 28  2015 /etc/a
djtime
73468    4 -rw-r--r--   1 root     root           10 Nov 16 20:42 /etc
/hostname
sysadmin@localhost:~$
```

# 8.5 Exibindo arquivos usando o comando less

Enquanto visualizar pequenos arquivos com o comando `cat` não apresenta problemas, não é uma escolha ideal para arquivos grandes. O comando `cat` não fornece nenhuma maneira de pausar e reiniciar a exibição facilmente, portanto, todo o conteúdo do arquivo é despejado na tela.

Para arquivos maiores, você desejará usar um comando pager para visualizar o conteúdo. Os comandos do pager exibirão uma página de dados por vez, permitindo que você avance e recue no arquivo usando as teclas de movimento.

Existem dois comandos de pager comumente usados:

- O comando `less`: Esse comando fornece um recurso de paginação muito avançado. Normalmente, é o pager padrão usado por comandos como o comando `man`.
- O comando `more`: Esse comando existe desde os primeiros dias do UNIX. Embora tenha menos recursos do que o comando `less`, ele tem uma vantagem importante: o comando `less` nem sempre é incluído em todas as distribuições do Linux (e, em algumas distribuições, não é instalado por padrão). O comando `more` está sempre disponível.

Quando você usa os comandos `more` ou `less`, eles permitem que você "movimente" um documento usando comandos de pressionamento de tecla. Como os desenvolvedores do comando `less` basearam o comando na funcionalidade do comando `more`, todos os comandos de pressionamento de tecla disponíveis no comando `more` também funcionam no comando `less`.

Para o propósito deste manual, o foco estará no comando mais avançado (`less`). O comando `more` ainda é útil para lembrar quando o comando `less` não estiver disponível. Lembre-se de que a maioria dos comandos de teclas fornecidos funcionam nos dois comandos.

# 8.5.1 Tela de ajuda em less

Quando você visualiza um arquivo com o comando `less`, você pode usar a tecla **h** para exibir uma tela de ajuda. A tela de ajuda permite que você veja quais outros comandos estão disponíveis. No exemplo a seguir, o comando `less /usr/share/dict/words` é executado. Uma vez que o documento é exibido, a tecla **h** foi pressionada, exibindo a tela de ajuda:

```
                  SUMMARY OF LESS COMMANDS

      Commands marked with * may be preceded by a number, N.
      Notes in parentheses indicate the behavior if N is given.



  h  H                 Display this help.
  q  :q  Q  :Q  ZZ     Exit.
 ---------------------------------------------------------------------------

                           MOVING

  e  ^E  j  ^N  CR  *  Forward  one line   (or N lines).
  y  ^Y  k  ^K  ^P  *  Backward one line   (or N lines).
  f  ^F  ^V  SPACE  *  Forward  one window (or N lines).
  b  ^B  ESC-v      *  Backward one window (or N lines).
  z                 *  Forward  one window (and set window to N).
  w                 *  Backward one window (and set window to N).
  ESC-SPACE         *  Forward  one window, but don't stop at end-of-f
ile.
  d  ^D             *  Forward  one half-window (and set half-window t
o N).
  u  ^U             *  Backward one half-window (and set half-window t
o N).
  ESC-)  RightArrow *  Left  one half screen width (or N positions).
  ESC-(  LeftArrow  *  Right one half screen width (or N positions).
HELP -- Press RETURN for more, or q when done
```

# 8.5.2 Comandos de movimento do less

Existem muitos comandos de movimento para o comando `less`, cada um com várias chaves ou combinações de teclas possíveis. Embora isso possa parecer intimidante, lembre-se de que você não precisa memorizar todos esses comandos de movimento; Você sempre pode usar a tecla **h** sempre que precisar de ajuda.

O primeiro grupo de comandos de movimento nos quais você pode querer se concentrar são os mais usados. Para tornar isso ainda mais fácil de aprender, as chaves que são idênticas em `more` e `less` serão resumidas. Desta forma, você estará aprendendo a se mover em `more` e `less` ao mesmo tempo:

| Movimento | Tecla |
|---|---|
| Avançar página | **Barra de espaço** |
| Voltar página | **B** |
| Avançar linha | **Enter** |
| Sair | **Q** |
| Ajuda | **H** |

Quando simplesmente usar `less` como um pager, a maneira mais fácil de avançar uma página é pressionar a barra de espaço.

# 8.5.3 Comandos de pesquisa less

Existem duas maneiras de pesquisar no comando `less`: você pode pesquisar para frente ou para trás a partir de sua posição atual usando padrões chamados expressões regulares. Mais detalhes sobre expressões regulares são fornecidos posteriormente neste capítulo.

Para iniciar uma pesquisa a partir da sua posição atual, use a tecla ***/***. Em seguida, digite o texto ou padrão para corresponder e pressione a tecla **Enter**.

Se uma correspondência puder ser encontrada, o cursor se moverá no documento para a correspondência. Por exemplo, no gráfico a seguir, a expressão " frog" foi pesquisada no arquivo `/usr/share/dict/words`:

```
bullfrog
bullfrog's
bullfrogs
bullheaded
bullhorn
bullhorn's
bullhorns
bullied
bullies
bulling
bullion
bullion's
bullish
bullock
bullock's
bullocks
bullpen
bullpen's
bullpens
bullring
bullring's
bullrings
bulls
:
```

Observe que " frog " não precisa ser uma palavra por si só. Observe também que, enquanto o comando `less` levou você para a primeira partida da posição atual, todas as correspondências foram destacadas.

Se não for possível encontrar partidas a partir de sua posição atual, a última linha da tela informará " `Pattern not found` ":

```
Pattern not found (press RETURN)
```

Para iniciar uma pesquisa para olhar para trás a partir da sua posição atual, pressione a tecla **?**, digite o texto ou padrão para combinar e pressione a tecla **Enter**. Seu cursor voltará para a primeira correspondência que encontrar ou informará que o padrão não pode ser encontrado.

Se mais de uma correspondência puder ser encontrada por uma pesquisa, o uso da tecla **n** permitirá que você mude para a próxima partida e, usando a tecla **N**, você poderá ir para uma partida anterior.

# 8.6 Revisitando os Comandos head e tail

Lembre-se de que os comandos `head` e `tail` são usados para filtrar arquivos para mostrar um número limitado de linhas. Se você quiser visualizar um número selecionado de linhas na parte superior do arquivo, use o comando `head` e, se desejar visualizar um número selecionado de linhas na parte inferior de um arquivo, use o comando `tail`.

Por padrão, os dois comandos exibem dez linhas do arquivo. A tabela a seguir fornece alguns exemplos:

| Command Example | Explanation of Displayed Text |
|---|---|
| `head /etc/passwd` | Primeiras 10 linhas de `/etc/passwd` |
| `head -3 /etc/group` | Primeiras 3 linhas de `/etc/group` |
| `head -n 3 /etc/group` | Primeiras 3 linhas de `/etc/group` |
| `help \| head` | Primeiras 10 linhas da saída por pipe do comando `help` |
| `tail /etc/group` | Últimas 10 linhas do `/etc/group` |
| `tail -5 /etc/passwd` | Últimas 5 linhas do `/etc/passwd` |
| `tail -n 5 /etc/passwd` | Últimas 5 linhas do `/etc/passwd` |
| `help \| tail` | Últimas 10 linas da saída por pipe do comando `help` |

Como visto nos exemplos acima, ambos os comandos produzirão texto a partir de um arquivo regular ou da saída de qualquer comando enviado por meio de um pipe. Ambos usam a opção -n para indicar quantas linhas serão produzidas.

# 8.6.1 Valor negativo com a opção -n

Tradicionalmente, no UNIX, o número de linhas para a saída seria especificado como uma opção com qualquer comando, então `-3` significava mostrar três linhas. Para o comando `tail`, `-3` ou `-n -3` ainda significa mostrar três linhas. No entanto, a versão GNU do comando `head` reconhece `-n -3` como mostra todas exceto as últimas três linhas, e ainda assim o comando `head` ainda reconhece a opção `-3` como mostra as três primeiras linhas.

# 8.6.2 Positive Value With the tail Command

The GNU version of the `tail` command allows for a variation of how to specify the number of lines to be printed. If you use the `-n` option with a number prefixed by the plus sign, then the `tail` command recognizes this to mean to display the contents starting at the specified line and continuing all the way to the end.

For example, the following will display line #22 to the end of the output of the `nl` command:

```
sysadmin@localhost:~$ nl /etc/passwd | tail -n +22


    22  sshd:x:103:65534::/var/run/sshd:/usr/sbin/nologin
    23  operator:x:1000:37::/root:/bin/sh
    24  sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/
bin/bash
sysadmin@localhost:~$
```

# 8.6.3 Seguindo alterações em um arquivo

Você pode visualizar as alterações do arquivo ao vivo usando a opção `-f` no comando `tail`. Isso é útil quando você deseja ver as alterações em um arquivo enquanto elas estão acontecendo.

Um bom exemplo disso seria ao visualizar arquivos de log como administrador do sistema. Os arquivos de log podem ser usados para solucionar problemas e os administradores os verão "interativamente" com o comando `tail`, já que estão executando os comandos que estão tentando solucionar em uma janela separada.

Por exemplo, se você fizesse login como usuário `root`, poderia solucionar problemas com o servidor de email, visualizando as alterações ao vivo em seu arquivo de log com o seguinte comando: `tail -f /var/log/mail.log`

# 8.7 Classificando arquivos ou entrada

O comando `sort` pode ser usado para reorganizar as linhas de arquivos ou entradas em um dicionário ou ordem numérica com base no conteúdo de um ou mais campos. Os campos são determinados por um separador de campo contido em cada linha, cujo padrão é o espaço em branco (espaços e tabulações).

O exemplo a seguir cria um arquivo pequeno, usando o comando `head` para capturar as primeiras 5 linhas do arquivo `/etc/passwd` e enviar a saída para um arquivo chamado `mypasswd`.

```
sysadmin@localhost:~$ head -5 /etc/passwd > mypasswd
sysadmin@localhost:~$
sysadmin@localhost:~$ cat mypasswd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
sysadmin@localhost:~$
```

Agora vamos classificar o arquivo mypasswd com o comando `sort`:

```
sysadmin@localhost:~$ sort mypasswd
bin:x:2:2:bin:/bin:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
root:x:0:0:root:/root:/bin/bash
sync:x:4:65534:sync:/bin:/bin/sync
sys:x:3:3:sys:/dev:/bin/sh
sysadmin@localhost:~$
```

# 8.7.1 Campos e opções do sort

Caso o arquivo ou entrada possa ser separado por outro delimitador, como uma vírgula ou dois pontos, a opção `-t` permitirá que outro separador de campo seja especificado. Para especificar campos para o `sort`, use a opção `-k` com um argumento para indicar o número do campo (começando com 1 para o primeiro campo).

As outras opções comumente usadas para o comando `sort` são o `-n` para executar uma ordenação numérica e `-r` para executar uma ordenação inversa.

No próximo exemplo, a opção `-t` é usada para separar campos por um caractere de dois-pontos e executa uma classificação numérica usando o terceiro campo de cada linha:

```
sysadmin@localhost:~$ sort -t: -n -k3 mypasswd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
sysadmin@localhost:~$
```

Observe que a opção `-r` pode ter sido usada para reverter a classificação, fazendo com que os números mais altos no terceiro campo apareçam no topo da saída:

```
sysadmin@localhost:~$ sort -t: -n -r -k3 mypasswd
sync:x:4:65534:sync:/bin:/bin/sync
sys:x:3:3:sys:/dev:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
root:x:0:0:root:/root:/bin/bash
sysadmin@localhost:~$
```

Por fim, você pode querer executar classificações mais complexas, como classificar por um campo primário e depois por um campo secundário. Por exemplo, considere os seguintes dados:

```
bob:smith:23


nick:jones:56
sue:smith:67
```

Você pode querer classificar primeiro pelo sobrenome (campo #2) e depois pelo primeiro nome (campo #1) e depois pela idade (campo #3). Isso pode ser feito com o seguinte comando:

```
sysadmin@localhost:~$ sort -t: -k2 -k1 -k3n filename
```

# 8.8 Exibindo estatísticas de arquivos com o comando wc

O comando `wc` permite que até três estatísticas sejam impressas para cada arquivo fornecido, bem como o total dessas estatísticas, se mais de um nome de arquivo for fornecido. Por padrão, o comando `wc` fornece o número de linhas, palavras e bytes (1 byte = 1 caractere em um arquivo de texto):

```
sysadmin@localhost:~$ wc /etc/passwd /etc/passwd-


  35   56 1710 /etc/passwd
  34   55 1665 /etc/passwd-
  69  111 3375 total
sysadmin@localhost:~$
```

O exemplo acima mostra a saída da execução: `wc /etc/passwd /etc/passwd-`. A saída possui quatro colunas: número de linhas no arquivo, número de palavras no arquivo, número de bytes no arquivo e o nome do arquivo ou `total`.

Se você estiver interessado na visualização de estatísticas específicas, você pode usar `-l` para mostrar apenas o número de linhas, `-w` para mostrar apenas o número de palavras e, `-c` para mostrar apenas o número de bytes.

O comando `wc` pode ser útil para contar o número de linhas geradas por algum outro comando através de um pipe. Por exemplo, se você quisesse saber o número total de arquivos no diretório `/etc`, você poderia executar `ls /etc | wc -l`:

```
sysadmin@localhost:~$ ls /etc/ | wc -l
136
sysadmin@localhost:~$
```

# 8.9 Usando o comando cut para filtrar o conteúdo do arquivo

O comando `cut` pode extrair colunas de texto de um arquivo ou entrada padrão. Um uso principal do comando `cut` é para trabalhar com arquivos de banco de dados delimitados. Esses arquivos são muito comuns em sistemas Linux.

Por padrão, ele considera sua entrada como separada pelo caractere de tabulação **Tab**, mas a opção `-d` pode especificar delimitadores alternativos, como os dois pontos ou a vírgula.

Usando a opção opção `-f`, você pode especificar quais campos exibir, como um intervalo com hífen ou uma lista separada por vírgulas.

No exemplo a seguir, o primeiro, quinto, sexto e sétimo campos do arquivo de banco de dados `mypasswd` são exibidos:

```
sysadmin@localhost:~$ cut -d: -f1,5-7 mypasswd
root:root:/root:/bin/bash
daemon:daemon:/usr/sbin:/bin/sh
bin:bin:/bin:/bin/sh
sys:sys:/dev:/bin/sh
sync:sync:/bin:/bin/sync
sysadmin@localhost:~$
```

Usando o comando `cut`, você também pode extrair colunas de texto com base na posição do caractere com a opção `-c`. Isso pode ser útil para extrair campos de arquivos de banco de dados de largura fixa. Por exemplo, o seguinte mostrará apenas o tipo de arquivo (caractere #1), permissões (caracteres #2-10) e nome do arquivo (caracteres #50+) da saída do comando `ls -l`:

```
sysadmin@localhost:~$ ls -l | cut -c1-11,50-
total 12
drwxr-xr-x Desktop
drwxr-xr-x Documents
drwxr-xr-x Downloads
drwxr-xr-x Music
drwxr-xr-x Pictures
drwxr-xr-x Public
drwxr-xr-x Templates
drwxr-xr-x Videos
-rw-rw-r-- errors.txt
-rw-rw-r-- mypasswd
```

```
-rw-rw-r-- new.txt

sysadmin@localhost:~$
```

# 8.10 Usando o comando grep para filtrar o conteúdo do arquivo

O comando `grep` pode ser usado para filtrar linhas em um arquivo ou a saída de outro comando com base na correspondência de um padrão. Esse padrão pode ser tão simples quanto o texto exato que você deseja combinar ou pode ser muito mais avançado através do uso de expressões regulares (discutido mais adiante neste capítulo).

Por exemplo, você pode querer encontrar todos os usuários que podem efetuar login no sistema com o shell BASH, assim você pode usar o comando `grep` para filtrar as linhas do arquivo `/etc/passwd` para as linhas que contêm os caracteres `bash`:

```
sysadmin@localhost:~$ grep bash /etc/passwd
root:x:0:0:root:/root:/bin/bash
sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash
sysadmin@localhost:~$
```

Para tornar mais fácil ver o que exatamente corresponde, use a opção `--color`. Esta opção realçará os itens correspondentes em vermelho:

```
sysadmin@localhost:~$ grep --color bash /etc/passwd
root:x:0:0:root:/root:/bin/bash
sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/bash
sysadmin@localhost:~$
```

Em alguns casos, você não se importa com as linhas específicas que correspondem ao padrão, mas com quantas linhas correspondem ao padrão. Com a opção `-c`, você pode contar quantas linhas correspondem:

```
sysadmin@localhost:~$ grep -c bash /etc/passwd
2
sysadmin@localhost:~$
```

Quando você está visualizando a saída do comando `grep`, pode ser difícil determinar os números de linha originais. Essas informações podem ser úteis quando você voltar ao arquivo (talvez para editar o arquivo), pois você pode usar essas informações para encontrar rapidamente uma das linhas correspondentes.

A opção `-n` para o comando `grep` exibirá números de linha originais:

```
sysadmin@localhost:~$ grep -n bash /etc/passwd
1:root:x:0:0:root:/root:/bin/bash
24:sysadmin:x:1001:1001:System Administrator,,,,:/home/sysadmin:/bin/b
as
sysadmin@localhost:~$
```

Algumas opções adicionais para utilizar com o `grep` :

| Exemplos | Saída |
| --- | --- |
| `grep -v nologin /etc/passwd` | Todas as linhas que não tenham `nologin` no arquivo `/etc/passwd` |
| `grep -l linux /etc/*` | Lista os arquivos no diretório `/etc` directory contendo `linux` |
| `grep -i linux /etc/*` | Listagem de linhas de arquivos no diretório `/etc` contendo qualquer case (maiúscula ou minúscula) do padrão `linux` |
| `grep -w linux /etc/*` | Listagem das linhas dos arquivos no diretório `/etc` directory contendo a palavra padrão `linux` |

# 8.11 Expressões Regulares Básicas

Uma *Expressão Regular* é uma coleção de caracteres "normais" e "especiais" usados para corresponder a padrões simples ou complexos. Caracteres normais são caracteres alfanuméricos que correspondem a si mesmos. Por exemplo, um `a` corresponderia a um `a`.

Alguns caracteres têm significados especiais quando usados em padrões por comandos como o comando `grep`. Há duas: *Expressões Regulares Básicas* (disponíveis para uma grande variedade de comandos do Linux) e *Expressões Regulares Estendidas* (disponíveis para comandos Linux mais avançados). Expressões regulares básicas incluem o seguinte:

| Expressão Regular | Correspondência |
|---|---|
| `.` | Qualquer caractere único |
| `[ ]` | Uma lista ou intervalo de caracteres para corresponder a um caractere, a menos que o primeiro caractere seja o acento circunflexo ^ e, em seguida, significa qualquer caractere que não esteja na lista |
| `*` | Caractere anterior repetido zero ou mais vezes |
| `^` | O texto a seguir deve aparecer no início da linha |
| `$` | Texto precedente deve aparecer no final da linha |

O comando `grep` é apenas um dos muitos comandos que suportam expressões regulares. Alguns outros comandos incluem os comandos `more` e `less`. Embora algumas das expressões regulares sejam citadas desnecessariamente com aspas simples, é uma boa prática usar aspas simples em torno de suas expressões regulares para evitar que o shell tente interpretar um significado especial delas.

# 8.11.1 Expressões Regulares Básicas - o. caracter .

No exemplo abaixo, um arquivo simples é criado primeiro usando o redirecionamento. Em seguida, o comando `grep` é usado para demonstrar uma correspondência de padrão simples:

```
sysadmin@localhost:~$ echo 'abcddd' > example.txt
sysadmin@localhost:~$ cat example.txt


abcddd
sysadmin@localhost:~$ grep --color 'a..' example.txt
abcddd
sysadmin@localhost:~$
```

No exemplo anterior, você pode ver que o padrão `a..` correspondeu `abc` . O primeiro . caractere combinou o `b` e o segundo combinou o `c`.

No próximo exemplo, o padrão `a..c` não corresponderá a nada, portanto, o comando `grep` não produzirá nenhuma saída. Para a correspondência ser bem sucedida, precisaria haver dois caracteres entre `a` e `c` no `example.txt`:

```
sysadmin@localhost:~$ grep --color 'a..c' example.txt
sysadmin@localhost:~$
```

# 8.11.2 Expressões Regulares Básicas – os caracteres [ ]

Se você usar o caractere `.`, então qualquer caractere possível poderia corresponder. Em alguns casos, você deseja especificar exatamente quais caracteres deseja corresponder. Por exemplo, talvez você queira apenas combinar um caractere alfabético minúsculo ou um caractere numérico. Para isso, você pode usar os caracteres de expressão regular `[ ]` e especificar os caracteres válidos dentro dos caracteres `[ ]`.

Por exemplo, o comando a seguir corresponde a dois caracteres, o primeiro é `a` ou a `b`, enquanto o segundo é `a`, `b`, `c` ou `d`:

```
sysadmin@localhost:~$ grep --color '[ab][a-d]' example.txt
abcddd
sysadmin@localhost:~$
```

Observe que você pode listar cada caractere possível `[abcd]` ou fornecer um intervalo `[a-d]`, desde que o intervalo esteja na ordem correta. Por exemplo, `[d-a]` não funcionaria porque não é um intervalo válido:

```
sysadmin@localhost:~$ grep --color '[d-a]' example.txt
grep: Invalid range end
sysadmin@localhost:~$
```

O intervalo é especificado por um padrão chamado tabela ASCII. Esta tabela é uma coleção de todos os caracteres imprimíveis em uma ordem específica. Você pode ver a tabela ASCII com o comando `man ascii`. Um pequeno exemplo:

```
      041   33   21   !                              141   97   61   a
      042   34   22   "                              142   98   62   b
      043   35   23   #                              143   99   63   c
      044   36   24   $                              144   100  64   d
      045   37   25   %                              145   101  65   e
      046   38   26   &                              146   102  66   f
```

Como `a` tem um valor numérico menor (`141`) então `d` (`144`), o intervalo `a-d` inclui todos os caracteres de `a` até `d`.

E se você quiser combinar um caractere que pode ser qualquer coisa menos um `x`, `y` ou `z`? Você não precisaria fornecer um conjunto `[ ]` com todos os caracteres, exceto `x`, `y` ou `z`.

Para indicar que você deseja corresponder a um caractere que não seja um dos caracteres listados, inicie seu `[ ]` conjunto com um símbolo `^`. Por exemplo, o seguinte demonstrará a correspondência de um padrão que inclua um caractere que não seja `a`, `b` ou `c` seguido por `d`:

```
sysadmin@localhost:~$ grep --color '[^abc]d' example.txt
abcddd
sysadmin@localhost:~$
```

# 8.11.3 Expressões regulares básicas – o caractere *

O caractere `*` pode ser usado para combinar "zero ou mais do caractere anterior". Por exemplo, o seguinte corresponderá a zero ou mais caracteres `d`:

```
sysadmin@localhost:~$ grep --color 'd*' example.txt
abcddd


sysadmin@localhost:~$
```

# 8.11.4 Expressões regulares básicas – os caracteres ^ e $

Quando você executa uma correspondência de padrão, a correspondência pode ocorrer em qualquer lugar na linha. Você pode querer especificar que a correspondência ocorre no início da linha ou no final da linha. Para combinar no início da linha, inicie o padrão com um símbolo ^.

No exemplo a seguir, outra linha é adicionada ao arquivo `example.txt` para demonstrar o uso do símbolo `^`:

```
sysadmin@localhost:~$ echo "xyzabc" >> example.txt
sysadmin@localhost:~$ cat example.txt
abcddd
xyzabc
sysadmin@localhost:~$ grep --color "a" example.txt
abcddd
xyzabc
sysadmin@localhost:~$ grep --color "^a" example.txt
abcddd
sysadmin@localhost:~$
```

Note que na primeira saída do `grep`, ambas as linhas correspondem porque ambas contêm a letra `a`. Na segunda saída do `grep`, apenas a linha que começava com a letra `a` correspondia.

Para especificar a correspondência ocorre no final da linha, finalize o padrão com o caractere `$`. Por exemplo, para encontrar apenas as linhas que terminam com a letra `c`:

```
sysadmin@localhost:~$ grep "c$" example.txt
xyzabc
sysadmin@localhost:~$
```

# 8.11.5 Expressões regulares básicas – o caractere \

Em alguns casos, talvez você queira corresponder a um caractere que seja um caractere especial de expressão regular. Por exemplo, considere o seguinte:

```
sysadmin@localhost:~$ echo "abcd*" >> example.txt
sysadmin@localhost:~$ cat example.txt
abcddd
xyzabc
abcd*
sysadmin@localhost:~$ grep --color "cd*" example.txt
abcddd
xyzabc
abcd*
sysadmin@localhost:~$
```

Na saída do comando `grep` acima, você verá que cada linha corresponde porque você está procurando por um caractere `c` seguido por zero ou mais `d` caracteres. Se você quiser procurar um caractere `*` real, coloque um caractere `\` antes do caractere `*`:

```
sysadmin@localhost:~$ grep --color "cd\*" example.txt


abcd*
sysadmin@localhost:~$
```

# 8.12 Expressões regulares estendidas

O uso de expressões regulares estendidas geralmente exige que uma opção especial seja fornecida ao comando para reconhecê-las. Historicamente, existe um comando chamado `egrep`, que é semelhante ao `grep`, mas é capaz de entender seu uso. Agora, o comando `egrep` está obsoleto em favor do uso do `grep` com a opção `-E` .

As seguintes expressões regulares são consideradas "estendidas":

| RE | Significado |
|---|---|
| ? | Corresponde ao caractere anterior zero ou uma vez, por isso é um caractere opcional |
| + | Corresponde ao caractere anterior repetido uma ou mais vezes |
| \| | ou lógico |

Alguns exemplos de expressões regulares estendidas:

| Comando | Significado | Correspondência |
|---|---|---|
| `grep -E 'colou?r' 2.txt` | Corresponde `colo` seguido de zero ou um caractere `u` | `color colour` |
| `grep -E 'd+' 2.txt` | Corresponde um ou mais caracteres `d` | `d dd ddd dddd` |
| `grep -E 'gray\|grey' 2.txt` | Corresponde ou `gray` ou `grey` | `gray grey` |

# 8.13 Comando xargs

O comando `xargs` é usado para construir e executar linhas de comando a partir da entrada padrão. Esse comando é muito útil quando você precisa executar um comando com uma lista muito longa de argumentos, o que, em alguns casos, pode resultar em erro se a lista de argumentos for muito longa.

O comando `xargs` tem uma opção `-0` que desativa a cadeia de fim de arquivo, permitindo o uso de argumentos que contenham espaços, aspas ou barras invertidas.

O comando `xargs` é útil para permitir que os comandos sejam executados de forma mais eficiente. Seu objetivo é construir a linha de comando para um comando executar o menor número de vezes possível com tantos argumentos quanto possível, em vez de executar o comando várias vezes com um argumento a cada vez.

O comando `xargs` funciona dividindo a lista de argumentos em sub-listas e executando o comando com cada sub-lista. O número de argumentos em cada sub-lista não excederá o número máximo de argumentos para o comando que está sendo executado e, portanto, evita um erro "`Argument list too long`".

O exemplo a seguir mostra um cenário em que o comando `xargs` permitia a remoção de muitos arquivos, em que o uso de um caractere curinga (glob) normal falhou:

```
sysadmin@localhost:~/many$ rm *
bash: /bin/rm: Argument list too long
sysadmin@localhost:~/many$ ls | xargs rm


sysadmin@localhost:~/many$
```