ARQUIVAMENTO E COMPRESSÃO

7.1 Introdução

Neste capítulo, discutiremos como gerenciar arquivos *archive* na linha de comando.

O **arquivamento de arquivos** é usado quando um ou mais arquivos precisam ser transmitidos ou armazenados da forma mais eficiente possível. Existem dois aspectos para isso:

- **Arquivamento** - Combinando vários arquivos em um, o que elimina a sobrecarga em arquivos individuais e facilita a transmissão
- **Compressão** - Diminuindo os arquivos removendo informações redundantes

Você pode arquivar vários arquivos em um único arquivo e compactá-lo ou compactar um arquivo individual. O primeiro ainda é chamado de arquivamento, enquanto o segundo é chamado de compactação. Quando você pega um arquivo, descompacta e extrai um ou mais arquivos, você está desarquivando-o.

Mesmo que o espaço em disco seja relativamente barato, o arquivamento e a compactação ainda têm valor:

- Se você quiser disponibilizar um grande número de arquivos, como o código-fonte para um aplicativo ou uma coleção de documentos, será mais fácil para as pessoas baixarem um arquivo compactado do que baixar arquivos individuais.
- Os arquivos de registro têm o hábito de preencher discos, por isso é útil dividi-los por data e compactar versões mais antigas.
- Quando você faz o backup de diretórios, é mais fácil mantê-los todos em um arquivo do que na versão de cada arquivo.
- Alguns dispositivos de streaming, como fitas, têm melhor desempenho se você estiver enviando um fluxo de dados em vez de arquivos individuais.
- Muitas vezes, pode ser mais rápido compactar um arquivo antes de enviá-lo para uma unidade de fita ou em uma rede mais lenta e descompactá-lo na outra extremidade do que seria para enviá-lo descompactado.

Como administrador do Linux, você deve se familiarizar com as ferramentas de arquivamento e compactação de arquivos.

7.2 Compactando arquivos

A *compactação de arquivos* os torna menores removendo a duplicação de um arquivo e armazenando-o de forma que o arquivo possa ser restaurado. Um arquivo com texto legível por humanos pode ter palavras usadas com frequência substituídas por algo menor, ou uma imagem com um plano de fundo sólido pode representar manchas dessa cor por um código. Você geralmente não usa a versão compactada do arquivo, mas a descompacta antes de usar. O algoritmo de compactação é um procedimento que o computador faz para codificar o arquivo original e, como resultado, torná-lo menor. Os cientistas da computação pesquisam esses algoritmos e obtêm os melhores que podem trabalhar mais rápido ou tornar o arquivo de entrada menor.

Quando se fala em compressão, existem dois tipos:

- **Sem perda**: nenhuma informação é removida do arquivo. Compactar um arquivo e descompactá-lo deixa algo idêntico ao original.
- **Com perdas**: as informações podem ser removidas do arquivo à medida que são compactadas, de modo que a descompactação de um arquivo resulte em um arquivo ligeiramente diferente do original. Por exemplo, uma imagem com dois tons de verde sutilmente diferentes pode ser menor, tratando essas duas tonalidades como iguais. Muitas vezes, os olhos não conseguem perceber a diferença.

Geralmente os olhos e ouvidos humanos não notam pequenas imperfeições nas imagens e no áudio, especialmente quando são exibidos em um monitor ou tocados em alto-falantes. A compactação com perdas geralmente beneficia a mídia, pois resulta em tamanhos menores de arquivos e as pessoas não podem dizer a diferença entre o original e a versão com os dados alterados. Para coisas que devem permanecer intactas, como documentos, logs e software, você precisa de compactação sem perdas.

A maioria dos formatos de imagem, como GIF, PNG e JPEG, implementa algum tipo de compactação com perdas. Você geralmente pode decidir quanta qualidade deseja preservar. Uma qualidade inferior resulta em um arquivo menor, mas após a descompressão você pode perceber artefatos como bordas ásperas ou descolorações. A alta qualidade será muito parecida com a imagem original, mas o tamanho do arquivo será mais próximo do original.

A compactação de um arquivo já compactado não diminui o tamanho. Isso costuma ser esquecido quando se trata de imagens, já que elas já estão armazenadas em um formato compactado. Com a compactação sem perdas, essa compactação múltipla não é um problema, mas se você compactar e descompactar um arquivo várias vezes usando um algoritmo com perdas, você acabará tendo algo irreconhecível.

O Linux fornece várias ferramentas para compactar arquivos, o mais comum é o `gzip`. Aqui mostramos um arquivo de log antes e depois da compactação.

```
bob:tmp $ ls -l access_log*
-rw-r--r-- 1 sean sean 372063 Oct 11 21:24 access_log
bob:tmp $ gzip access_log
bob:tmp $ ls -l access_log*
-rw-r--r-- 1 sean sean 26080 Oct 11 21:24 access_log.gz
```

No exemplo acima, há um arquivo chamado `access_log` que tem 372.063 bytes. O arquivo é compactado invocando o comando `gzip` com o nome do arquivo como o único argumento. Depois que o comando for concluído, o arquivo original será removido e uma versão compactada com uma extensão de arquivo `.gz` será deixada em seu lugar. O tamanho do

arquivo agora é de 26.080 bytes, fornecendo uma taxa de compactação de cerca de 14: 1, o que é comum em arquivos de log.

O `gzip` lhe dará esta informação se você perguntar, usando o parâmetro `-l`, como mostrado aqui:

```
bob:tmp $ gzip -l access_log.gz
      compressed     uncompressed  ratio uncompressed_name
           26080           372063  93.0% access_log
```

Aqui, você pode ver que a taxa de compressão é dada como 93%, que é o inverso da proporção de 14: 1, ou seja, 13/14. Além disso, quando o arquivo é descompactado, ele será chamado de `access_log`.

```
bob:tmp $ gunzip access_log.gz
bob:tmp $ ls -l access_log*
-rw-r--r-- 1 sean sean 372063 Oct 11 21:24 access_log
```

O oposto do comando `gzip` é o `gunzip`. Alternativamente, o `gzip -d` faz a mesma coisa (o `gunzip` é apenas um script que chama o `gzip` com os parâmetros corretos). Depois que o `gunzip` faz seu trabalho, você pode ver que o arquivo `access_log` está de volta ao seu tamanho original.

O `gzip` também pode funcionar como um filtro, o que significa que ele não lê ou grava nada no disco, mas recebe dados por meio de um canal de entrada e os grava em um canal de saída. Você aprenderá mais sobre como isso funciona no próximo capítulo, portanto, o próximo exemplo dá uma ideia do que você pode fazer ao compactar um fluxo.

```
bob:tmp $ mysqldump -A | gzip > database_backup.gz
bob:tmp $ gzip -l database_backup.gz


         compressed        uncompressed  ratio uncompressed_name
              76866             1028003  92.5% database_backup
```

O comando `mysqldump -A` envia o conteúdo dos bancos de dados MySQL locais para o console. O caractere `|` (pipe) diz “redirecione a saída do comando anterior para a entrada do próximo”. O programa para receber a saída é o `gzip`, que reconhece que nenhum nome de arquivo foi dado, portanto ele deve operar no modo pipe. Finalmente, a parte final do comando `> database_backup.gz` significa “redirecionar a saída do comando anterior para um arquivo chamado `database_backup.gz`. Inspecionar este arquivo com o `gzip -l` mostra que a versão compactada é 7.5% do tamanho do original, com o benefício adicional de que o arquivo maior nunca precisou ser gravado no disco.

Existe outro par de comandos que operam virtualmente de forma idêntica ao `gzip` e ao `gunzip`. Estes são `bzip2` e `bunzip2`. Os utilitários `bzip` usam um algoritmo de compressão diferente (chamado de ordenação de blocos Burrows-Wheeler, versus a codificação Lempel-Ziv usada pelo `gzip`) que pode compactar arquivos menores que gzip às custas de mais tempo de CPU. Você pode reconhecer esses arquivos porque eles têm uma extensão `.bz` ou `.bz2` em vez de `.gz`.

7.3 Arquivando arquivos

Se você tivesse vários arquivos para enviar a alguém, poderia compactar cada um individualmente. Você teria uma quantidade menor de dados no total do que se enviasse arquivos descompactados, mas ainda teria que lidar com muitos arquivos de uma só vez.

O *arquivamento* é a solução para esse problema. O utilitário tradicional do UNIX para arquivar arquivos é chamado `tar`, que é uma forma abreviada do TApe aRchive. O `tar` foi usado para transmitir muitos arquivos para uma fita para backups ou transferência de arquivos. O `tar` pode ter como entrada vários arquivos e cria um único arquivo de saída que pode ser dividido novamente nos arquivos originais na outra extremidade da transmissão.

O `tar` tem 3 modos que você vai querer estar familiarizado:

- **Criar**: crie um novo arquivo a partir de uma série de arquivos
- **Extrair**: extrai um ou mais arquivos de um arquivo
- **Lista**: mostra o conteúdo do arquivo sem extrair

Lembrar os modos é a chave para descobrir as opções de linha de comando necessárias para fazer o que você deseja. Além do modo, você também deve se lembrar onde especificar o nome do arquivo, já que pode estar digitando vários nomes de arquivo em uma linha de comando.

Aqui, mostramos um arquivo *tar*, chamado de tarballs, criado a partir de vários registros de acesso.

```
bob:tmp $ tar -cf access_logs.tar access_log*
bob:tmp $ ls -l access_logs.tar
-rw-rw-r-- 1 sean sean 542720 Oct 12 21:42 access_logs.tar
```

Criar um arquivo archive requer duas opções nomeadas. O primeiro, `c`, especifica o modo. O segundo, `f`, diz ao `tar` para esperar um nome de arquivo como o próximo argumento. O primeiro argumento no exemplo acima cria um arquivo chamado `access_logs.tar`. Os argumentos restantes são todos tomados como nomes de arquivos de entrada, como um caractere curinga, uma lista de arquivos ou ambos. Neste exemplo, usamos a opção curinga para incluir todos os arquivos que começam com `access_log`.

O exemplo acima faz uma listagem longa do diretório do arquivo criado. O tamanho final é de 542.720 bytes, que é um pouco maior que os arquivos de entrada. Os tarballs podem ser comprimidos para facilitar o transporte, seja compactando o arquivo ou fazendo o `tar` com a opção `z` da seguinte maneira:

```
bob:tmp $ tar -czf access_logs.tar.gz  access_log*
bob:tmp $ ls -l access_logs.tar.gz
-rw-rw-r-- 1 sean sean 46229 Oct 12 21:50 access_logs.tar.gz
bob:tmp $ gzip -l access_logs.tar.gz
         compressed        uncompressed  ratio uncompressed_name
              46229              542720  91.5% access_logs.tar
```

O exemplo acima mostra o mesmo comando do exemplo anterior, mas com a adição do parâmetro `z`. A saída é muito menor que o próprio tarball, e o arquivo resultante é compatível

com o `gzip`. Você pode ver no último comando que o arquivo descompactado tem o mesmo tamanho que seria se você o tivesse executado o `tar` em uma etapa separada.

Embora o UNIX não trate as extensões de arquivo especialmente, a convenção é usar `.tar` para arquivos `tar` e . `.tar.gz` ou `.tgz` para arquivos `tar` compactados. Você pode usar `bzip2` em vez de `gzip` substituindo a letra `z` por `j` e usando `.tar.bz2`, `.tbz` ou `.tbz2` para uma extensão de arquivo (por exemplo, `tar -cjf file.tbz access_log*`).

Dado um arquivo `tar`, compactado ou não, você pode ver o que está nele usando a opção `t`:

```
bob:tmp $ tar -tjf access_logs.tbz
logs/
logs/access_log.3
logs/access_log.1
logs/access_log.4
logs/access_log
logs/access_log.2
```

Este exemplo usa 3 opções:

- `t`: lista de arquivos no arquivo
- `j`: descomprima com o `bzip2` antes de ler
- `f`: opera no nome de arquivo fornecido `access_logs.tbz`

O conteúdo do arquivo compactado é exibido. Você pode ver que um diretório foi prefixado nos arquivos. O Tar recorrerá aos subdiretórios automaticamente ao compactar e armazenará as informações do caminho dentro do archive.

Apenas para mostrar que este arquivo ainda não é nada especial, listaremos o conteúdo do arquivo em duas etapas usando um pipeline.

```
bob:tmp $ bunzip2 -c access_logs.tbz | tar -t
logs/
logs/access_log.3
logs/access_log.1
logs/access_log.4
logs/access_log
logs/access_log.2
```

O lado esquerdo do pipeline é `bunzip -c access_logs.tbz`, que descompacta o arquivo, mas a opção `-c` envia a saída para a tela. A saída é redirecionada para `tar -t`. Se você não especificar um arquivo com `-f`, o `tar` será lido na entrada padrão, que nesse caso é o arquivo descompactado.

Finalmente, você pode extrair o arquivo com a opção `-x`:

```
bob:tmp $ tar -xjf access_logs.tbz
bob:tmp $ ls -l
total 36
-rw-rw-r-- 1 sean sean 30043 Oct 14 13:27 access_logs.tbz
drwxrwxr-x 2 sean sean  4096 Oct 14 13:26 logs
bob:tmp $ ls -l logs
total 536
-rw-r--r-- 1 sean sean 372063 Oct 11 21:24 access_log
-rw-r--r-- 1 sean sean    362 Oct 12 21:41 access_log.1
-rw-r--r-- 1 sean sean 153813 Oct 12 21:41 access_log.2
-rw-r--r-- 1 sean sean   1136 Oct 12 21:41 access_log.3
-rw-r--r-- 1 sean sean    784 Oct 12 21:41 access_log.4
```

O exemplo acima usa o padrão similar como antes, especificando a operação (eXtract), a compactação (a opção `j`, significando `bzip2`) e um nome de arquivo (`-f access_logs.tbz`). O arquivo original é intocado e o novo diretório de logs é criado. Dentro do diretório estão os arquivos.

Adicione a opção `-v` e você terá uma saída detalhada dos arquivos processados. Isso é útil para você ver o que está acontecendo:

```
bob:tmp $ tar -xjvf access_logs.tbz
logs/
logs/access_log.3
logs/access_log.1
logs/access_log.4
logs/access_log
logs/access_log.2
```

It is important to keep the `-f` flag at the end, as `tar` assumes whatever follows it is a filename. In the next example, the `f`and `v` flags were transposed, leading to `tar` interpreting the command as an operation on a file called "`v`" (the relevant message is in italics.)

```
bob:tmp $ tar -xjfv access_logs.tbz
tar (child): v: Cannot open: No such file or directory
tar (child): Error is not recoverable: exiting now
tar: Child returned status 2
tar: Error is not recoverable: exiting now
```

Se você quiser apenas alguns arquivos fora do arquivo, você pode adicionar seus nomes ao final do comando, mas, por padrão, eles devem corresponder exatamente ao nome no arquivo ou usar um padrão:

```
bob:tmp $ tar -xjvf access_logs.tbz logs/access_log
logs/access_log
```

O exemplo acima mostra o mesmo arquivo de antes, mas extrai apenas o arquivo `logs/access_log`. A saída do comando (como o modo detalhado foi solicitado com a opção `v`) mostra apenas que um arquivo foi extraído.

O `tar` tem muito mais recursos, como a capacidade de usar padrões ao extrair arquivos, excluir determinados arquivos ou exibir os arquivos extraídos na tela, em vez de no disco. A documentação do `tar` tem informações detalhadas.

7.4 Arquivos ZIP

O utilitário de arquivamento de fato no mundo da Microsoft é o arquivo ZIP. Não é tão prevalente no Linux, mas é bem suportado pelos comandos `zip` e `unzip`. Com o `tar` e o `gzip`/`gunzip`, os mesmos comandos e opções podem ser usados para fazer a criação e a extração, mas este não é o caso do `zip`. A mesma opção tem significados diferentes para os dois comandos diferentes.

O modo padrão do `zip` é adicionar arquivos a um arquivo e compactá-lo.

```
bob:tmp $ zip logs.zip logs/*
  adding: logs/access_log (deflated 93%)
  adding: logs/access_log.1 (deflated 62%)
  adding: logs/access_log.2 (deflated 88%)
  adding: logs/access_log.3 (deflated 73%)
  adding: logs/access_log.4 (deflated 72%)
```

O primeiro argumento no exemplo acima é o nome do arquivo a ser operado, neste caso é `logs.zip`. Depois disso, é uma lista de arquivos a serem adicionados. A saída mostra os arquivos e a taxa de compactação. Deve-se notar que o `tar` requer a opção `-f` para indicar que um nome de arquivo está sendo passado, enquanto o `zip` e o `unzip` exigem um nome de arquivo e, portanto, não é necessário que você diga explicitamente que um nome de arquivo está sendo passado.

O `zip` não recorre aos subdiretórios por padrão, o que é um comportamento diferente do `tar`. Ou seja, apenas adicionar `logs` em vez de `logs/*` somente adicionará o diretório vazio e não os arquivos abaixo dele. Se você quiser um comportamento do tipo `tar`, você deve usar o comando `-r` para indicar que a recursão deve ser usada:

```
bob:tmp $ zip -r logs.zip logs
  adding: logs/ (stored 0%)
  adding: logs/access_log.3 (deflated 73%)
  adding: logs/access_log.1 (deflated 62%)

  adding: logs/access_log.4 (deflated 72%)
  adding: logs/access_log (deflated 93%)
  adding: logs/access_log.2 (deflated 88%)
```

No exemplo acima, todos os arquivos no diretório de `logs` são adicionados porque usam a opção `-r`. A primeira linha de saída indica que um diretório foi adicionado ao archive, mas, caso contrário, a saída é semelhante ao exemplo anterior.

A listagem de arquivos no `zip` é feita pelo comando `unzip` e pela opção `-l` (list):

```
bob:tmp $ unzip -l logs.zip
Archive:  logs.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  10-14-2013 14:07   logs/
     1136  10-14-2013 14:07   logs/access_log.3
      362  10-14-2013 14:07   logs/access_log.1
      784  10-14-2013 14:07   logs/access_log.4
    90703  10-14-2013 14:07   logs/access_log
   153813  10-14-2013 14:07   logs/access_log.2
---------                     -------
   246798                     6 files
```

Extrair os arquivos é como criar o arquivo, pois a operação padrão é extrair:

```
bob:tmp $ unzip logs.zip
Archive:  logs.zip
   creating: logs/
  inflating: logs/access_log.3
  inflating: logs/access_log.1
  inflating: logs/access_log.4
  inflating: logs/access_log
  inflating: logs/access_log.2
```

Aqui, extraímos todos os arquivos do archive para o diretório atual. Assim como o `tar`, você pode passar nomes de arquivos na linha de comando:

```
bob:tmp $ unzip logs.zip access_log
Archive:  logs.zip
caution: filename not matched:  access_log
bob:tmp $ unzip logs.zip logs/access_log
Archive:  logs.zip
  inflating: logs/access_log
bob:tmp $ unzip logs.zip logs/access_log.*
Archive:  logs.zip
  inflating: logs/access_log.3
  inflating: logs/access_log.1
  inflating: logs/access_log.4
  inflating: logs/access_log.2
```

O exemplo acima mostra três tentativas diferentes para extrair um arquivo. Primeiro, apenas o nome do arquivo é passado sem o componente de diretório. Como o `tar`, o arquivo não é correspondido.

A segunda tentativa passa o componente de diretório junto com o nome do arquivo, que extrai apenas esse arquivo.

A terceira versão usa um curinga, que extrai os 4 arquivos correspondentes ao padrão, assim como o `tar`.

As páginas man `zip` e `unzip` descrevem outras coisas que você pode fazer com essas ferramentas, como substituir arquivos dentro do archive, usar diferentes níveis de compactação e até mesmo usar criptografia.