



Apostila

Programação

Avançada em

Shell Script

Por Reinaldo Marques de Lima

Esta Apostila tem como finalidade orientar e mostrar, à aqueles que tem interesse em aprender a desenvolver scripts, que a criação de script em shell não se limitam somente a sua execução em linha de comando, e que é possível criar aplicações robustas e de fácil interação com o usuário.

Veremos algumas dicas para uma boa elaboração de scripts, faceis de serem compreendidos, visualizados e de simples manutenção.

Também veremos maneiras de criar scripts visuais com as caixas de dialogo do Gnome e do KDE.

E por fim veremos a interação de shell Scripts com programas utilizados no dia-a-dia, como o navegador em modo texto lynx e o suite de escritório Open Office, e também com aplicações administrativas, como Servidor Apache, Bancos de Dados (Oracle e MySQL).

Índice

<i>Introdução.....</i>	<i>6</i>
<i>Dicas e técnicas para um bom script.....</i>	<i>7</i>
<i>Dicas de Scripts integrados.....</i>	<i>20</i>
<i>Scripts com ferramentas administrativas..</i>	<i>40</i>
<i>Criação de Programas visuais.....</i>	<i>50</i>
<i>Alguns scripts criados pelo autor.....</i>	<i>66</i>
<i>Considerações Finais.....</i>	<i>98</i>

*Aproveite a leitura,
Reinaldo*



Publicação Digital

- **Introdução**
- **Capítulo 1 - Dicas e técnicas para um bom script:**
 - Comentários são só comentários;
 - Portabilidade em scripts para vários sistemas;
 - Principais variáveis de ambiente do bash e suas funções;
 - Formatação de linhas para scripts com dialog e similares;
 - Funções e Case, uma dupla dinâmica;
 - Tudo sobre redirecionamento;
 - Aprendendo a usar o hold-space do sed;
 - Técnica de recursividade em Shell Script;
 - Algumas dicas aleatórias.
- **Capítulo 2 - Dicas de Scripts integrados com:**
 - Navegador Lynx
 - conhecendo o navegador lynx;
 - Onde conseguir o Lynx;
 - comandos mais usuais;
 - Opções do Menu Inicial;
 - Algumas opções de Parâmetros;
 - Variáveis de Ambiente;
 - Executando cgi's em Lynx, você sabia?
 - O arquivo lynx.cfg
 - Algumas Dicas;
 - Efetuando pesquisas;
 - Exemplos de Shell-script com Lynx;
 - Open Office
 - recebendo parâmetros pela linha de comando;
 - redirecionando saídas de scripts para o Open Office.
- **Capítulo 3 - Scripts com ferramentas administrativas:**
 - Servidor Apache;
 - Um pouco sobre o Servidor Web Apache;
 - Onde conseguir;
 - Meu Apache está rodando?
 - Script que testa
 - Logs de acesso;
 - Script que processa logs de acesso;
 - Banco de Dados MySQL
 - dicas de criação de scripts;
 - expressões regulares em MySQL;
 - Banco de Dados Oracle
 - dicas de criação de scripts;
 - exemplos prontos;
- **Capítulo 4 - Criação de Programas visuais com:**
 - gmessage / xmessage;
 - exemplos das caixas;
 - zenity;
 - exemplos das caixas;
 - kdialog;
 - exemplos das caixas.
- **Apêndice - Alguns scripts criados pelo autor (EU).**
 - plaspkg - Meu primeiro pacote de "programas" em shell.
 - plaspkg_dialog - Mesmo pacote, mas todo em dialog.
 - plaspkg_zenity - Mais um, agora todo em zenity.
 - plasconvert - converte texto para html e vice-versa (em crecinemto).
 - plasinfo - busca informação em alguns sites da internet.
 - sysinfo - script simples que passa informação do sistema.
 - sysinfo_gmessage - mesmo programa escrito em gmessage.
 - go - faz conexão remota com servidores via ssh.
 - scripts_index - cria uma pagina html que mostra conteúdo do diretório "scripts".
 - gowalk - script que da um alerta visual quando se passam 50 minutos, para evitar

de ficar muito tempo sentado. (script politicamente correto)

- gowalk_xmessage - mesmo programa para outras interfaces.
- meuip - script que mostra o ip e a subnet mask.
- meuip_gmessage - mesmo script em gmessage.

- Considerações Finais.

- Agradecimentos
- Bibliografias e Links

Olá

Este documento foi criado para servir de apoio para criação de programa de alto nível utilizando Shell Script.

Aqui veremos muitas dicas e técnicas de como criar scripts acima da média utilizando recursos que você nem imaginava que poderiam ser inseridos dentro de um script.

O bom aproveitamento deste documento pode ajudar você a subir alguns degraus no mundo da programação em shell, e no final você vai ver que não é nenhum bicho de sete cabeças, se você já achava shell divertido, ao final deste documento você vai sentir-se como se sua cabeça estivesse a ponto de explodir de tantas ideias que irão surgir, para criação de novos scripts, seu leque de possibilidades vai crescer consideravelmente.

Todos os exemplos foram testados com o interpretador bash e funcionavam perfeitamente até o final da concepção deste documento.

Primeira dica: Evite ficar comentando a respeito disso com sua Namorada/Noiva/Esposa para não despertar uma crise de ciúmes achando que foi trocada por um script, como aconteceu comigo 8^S (heheheheh).

Aproveite a leitura, estude e pratique bastante, pois a recompensa no final é grande.

Forte Abraço

Reinaldo Marques de Lima

```
+-----+
|
|  Programas usados nesta apostila:
|
|  Interpretador: bash 3.00.16(1)
|  Navegador: Lynx 2.8.5.rel.1
|  Pacote de Escritório: OpenOffice 1.1.3
|  Servidor Web: Apache 1.3.33 (Debian GNU/Linux)
|  Bancos de Dados: MySQL, Oracle 10g
|  Criador de interface visual: gmessage 2.0.11
|                               xmessage 1.6
|                               zenity 2.10
|                               kdialog (KDE 3.3)
|
+-----+
```



Capítulo 1

Dicas e técnicas para um bom script:

- Comentários são só comentários;
- Portabilidade em scripts para vários sistemas;
- Formatação de linhas para scripts com dialog e similares;
- Funções e Case, uma dupla dinâmica;
- Técnica de recursividade em Shell Script;
- Algumas dicas aleatórias;

Vamos iniciar esta apostila falando de técnicas simples que são de fácil aprendizado, não doem nada e evitam dores de cabeça futuras. Coisas que é sempre bom ter em mente para a concepção de um script que (com certeza) vai precisar de manutenções e adaptações futuras, neste caso é sempre bom saber onde mexer e ainda, mexer o menos possível.

Comentários são só comentários...

Não faça de seu script um lugar de desabafo pessoais, nem coloque "receitas de bolo" ou ainda evite comentar cada linha de código como se as pessoas que irão analisá-lo fossem completos idiotas.

Mantenha a Linguagem simples e direta e em pontos importantes do script para esclarecer um determinado trecho do código.

Por exemplos, reserve as primeiras linhas para definir a "ficha técnica" do script, mais ou menos assim:

```
#!/bin/bash

##[ Ficha ]#####
#
# Nome: Meu script
#
# Escrito por: Mim
#
# Criado em: dd/mm/aaaa
#
# Ultima atualização: dd/mm/aaaa
#
##[ Descrição ]#####
#
# Script que faz um monte de coisas legais e que vai
# revolucionar o mundo da tecnologia
# ao redor do mundo.
#
#####
```

Simples, direto e sem muita firula, você pode também adicionar comentários de evolução, conforme forem sendo adicionados ou melhorados alguns códigos.

```
##[ Versões ]#####...
#
# Versão 0.1   : faz isso, isso e aquilo...
# Versão 0.2   : faz mais isso e mais isso....
# Versão 0.2.1 : melhorei isso e aquilo...
#
#####...
```

Outro exemplo seria com relação aos comentários no meio do código, para esclarecer o que um certo trecho do código faz. vejamos:

Iniciamos o bloco de variáveis básicas do script, aquelas que vão ser usadas em vários pontos do script, aconselha-se declara-las logo após o fim da "ficha técnica".

```
##( Variáveis ).....#

VARIABEL1=$1
VARIABEL2=$2
ARQUIVOS=$(ls *.txt)
.
.
e assim por diante, fica bem legível, fácil de identificar e ajuda bastante a leitura,
sabendo disso, aí vira festa....veja mais exemplos.

##( Funções ).....#

funcao1(){
...
}

funcao2(){
...
}

##( Case ).....#

case $VARIABEL in
....

esac
```

Para alguma linha de código que façam um trabalho importante, ou que seja de difícil entendimento, aconselha-se comentar das seguintes maneiras: assim:

```
linha importante | de difícil entendimento      # comentário que
que é muito | difícil | de visualizar          # ajuda a entender
                                                # a linha difícil

ou assim:
```

```
##[ Comentário sobre a linha difícil ].....#

linha importante | de difícil entendimento
que é muito | difícil | de visualizar
```

Na segunda maneira, a linha fica parecida coma de comentários em blocos "#()...#" diferenciando-se pelos colchetes [].



AVISO: cuidado, o uso abusivo de comentários costuma confundir mais do que ajudar.

Portabilidade em scripts para vários sistemas

Está é a dica mais simples, portanto uma das menores desta apostila, mas não menos

importante.

Uma maneira muito boa e extremamente simples de fazer com que seu script se torne portátil em vários sistemas "Unix Like" é definir em forma de variáveis todos os programas a serem utilizados pelo script com seus caminhos absolutos.



NOTA: Caminho absoluto é o endereço onde está situado o programa a partir do diretório raiz "/", exemplo: "/bin/bash" "/usr/bin/vi" etc..

Tá...eu sei que não é legal ficar procurando caminhos absolutos para cada ls ou wc que você for colocar em seu script, para isso os sistemas "Unix Like" tem um comando muito gente boa chamado **which** que te mostra o lugar certinho onde "mora" o programa/comando que você pretende usar em seu script.

```
prompt> which vi
/usr/bin/vi
```

```
prompt> which bash
/bin/bash
```

A partir daí podemos armazenar em uma variável o caminho absoluto do programa/comando sem precisar ficar caçando uma por uma.

Exemplos:

```
WC=$(which wc)
SED=$(which sed)
```



NOTA: Recomenda-se testar a eficiência do which na linha de comando

Para melhorar a portabilidade do script, você pode inserir uns testes para saber se um determinado programa/comando existe no sistema onde o script está sendo rodado.

Exemplo:

```
CAR=$(which car)
[ -z $CAR ] && echo 'A variável $CAR é nula' && exit
```

Ai você vem e me diz: "Mas espera aí, do que exatamente você está falando, porque eu deveria me preocupar com portabilidade e caminho absoluto?????"

R.: Simples, pelo fato de, pelo menos a grande maioria dos sistemas Unix like ser de código aberto, a padronização dos sistemas se torna algo impraticável, pois cada um organiza seu sistema da maneira que bem entende, e muitas vezes se você não passar o caminho absoluto de onde se encontra um determinado programa/comando, adeus script bem feito.

Exemplo:

O comando date no Linux Ubuntu fica em:

```
/bin/date
```

Já o mesmo comando no SunOS 5.7:

```
/usr/bin/date
```

Mas se você precisa que ele funcione tão bem no Linux e no SunOS você precisa passar o caminho:

/usr/gnu/bin/date

Ou seja, sem caminho absoluto, não tem script portátil em vários sistemas.



NOTA: fique atento, pois, como no exemplo anterior, as vezes o mesmo programa com versões diferentes reside em caminhos diferentes. Portanto faça testes constantes.

Principais variáveis de ambiente do bash e suas funções

Variáveis de Ambiente são as variáveis nativas do Sistema que em geral especificam suas configurações, as vezes precisamos destas informações do sistema para criação de scripts que vão atuar diretamente na administração do Sistema, por exemplo. Ter conhecimento de como elas se comportam e de como obter informações sobre o Sistema, as vezes, evita algumas dores de cabeça. Portanto vou listar algumas das mais importantes e uma breve descrição sobre elas:

BASH - exibe o caminho absoluto do interpretador bash.

COLUMNS - exibe a quantidade de colunas existente no terminal corrente.

EUID - exibe o número do ID do usuário corrente.

GROUPS - lista os nomes dos grupos a qual o usuário corrente pertence.

HOME - exibe o diretório home do usuário corrente.

HOSTNAME - exibe o hostname corrente.

HOSTTYPE - indica em qual máquina o bash está sendo executado.

IFS - exibe a lista de caracteres utilizados para separação de campos. vazia por default.

LANG - mostra a(s) linguagem(ns) reconhecidas pelo sistema.

LINES - exibe a quantidade de linhas existentes no terminal corrente.

MAIL - avisa sobre a chegada de e-mails em um arquivo especificado. vazia por default.

PATH - exibe uma lista de diretórios em que o bash procura comandos para serem executados.

PIPESTATUS - lista processos mais recentes executados em foreground.

RANDOM - cria números randomicos de 0 até 32767.

SHELL - o pathname do shell em execução.

TERM - tipo de terminal em uso.

Para visualizar o conteúdo da variável, na linha de comando digite:

```
prompt> echo $VARIABLE
```

Uma lista completa das variáveis pode se exibida digitando:

```
prompt> echo $<TAB><TAB>
```

Formatação de linhas para scripts com dialog e similares

Esta é mais uma dica relâmpago para quem tem interesse em criar script usando as caixas de dialogo do Linux (dialog, zenity, kdialog, gmessage, xmessage, ...).

Para quem conhece os parâmetros passados na linha de comando desses aplicativos sabe que eles são meio extensos podendo, em alguns casos, usar mais de uma linha, para aqueles que não conhecem, eis um exemplo:

```
prompt> zenity --text="Terceiro exemplo de Caixa List - Radiolist" --list --radiolist --column selecionar FALSE primeiro FALSE segundo FALSE terceiro --column opcao
```

Neste exemplo a caixa de dialogo zenity (nativa do Gnome) irá criar uma caixa de dialogo Radiolist com três opções em duas colunas, lindo...mas imagina inserir uma linha destas em um script, uma outra pessoa que não entende tanto assim vai querer arrancar os cabelos para descobrir o que foi escrito ai, além de ficar um tanto feio visualmente. E se nós escrevermos desta forma:

```
zenity --title="$TITLE" \
      --text="Terceiro exemplo de Caixa List - Radiolist" \
      --list \
      --radiolist \
      --column selecionar FALSE primeiro FALSE segundo FALSE terceiro \
      --column opcao
```

Concorda que tanto no quesito visual quanto no quesito entendimento, este exemplo tirou nota 10, já o outro.....

Isto foi possível graças ao "caracter mágico '\'" que dentro de um script, ou até mesmo na linha de comando do terminal tem a finalidade de informar ao interpretador que a linha de comando ainda não terminou, pois no caso do zenity, por exemplo, colocar uma instrução na próxima linha vai provocar a ira do interpretador que não vai saber o que está acontecendo.

O "\" pode ser utilizado em outro casos, não somente neste, mas nem sempre isso é preciso. Existem ainda casos onde ele não se aplica, mais uma vez tudo se resolve testando.



DICA EXTRA: coloque o "\" como no exemplo acima, no final da linha, depois de alguns <TABS> para que seu alinhamento facilite na visualização. No capítulo 3 veremos mais exemplos

Funções e Case, uma dupla dinâmica

Aqui eu mostrarei uma parceria que costuma dar muito certo, principalmente se você quer criar um script que executa várias tarefas diferentes, mas todas relacionadas ao mesmo "assunto".

Explicando com um exemplo:

Vamos supor que você precise de um script que insira, remova, e liste ocorrências ou registros em um arquivo .txt e ele precisa ser acessado várias vezes no dia. Dependendo do porte do script ele ficaria monstruoso e você se perderia numa infinidade de if's elif's else's, como o objetivo é facilitar as coisa, podemos criar várias funções, e com isso ainda ganhar um certo desempenho no script, pois ao em vez de fazer vários testes e rastrear o script inteiro até achar o que ele precisa fazer, ele vai somente executar o conteúdo da função chamada.

E na chamada da função é que entra o tal do case, pois ele é quem vai se encarregar de direcionar a execução do script para uma determinada função.

Acompanhe como ficaria a estrutura do script baseada no exemplo dado:

```
#!/bin/bash

#
# Script que manipula registro do arquivo texto.txt
#

#( Funções ).....#
```

```

insere(){
.....
.....
....
}

remove(){
.....
.....
....
}

lista(){
.....
.....
....
}

#( Case para direcionar a função ).....#

case $1 in

    -i) insere; ;;
    -r) remove; ;;
    -l) lista; ;;

esac

```

O que aconteceu aqui é que o interpretador varreu linha por linha do script e verificou que só existiam funções até que ele achou um case que falou para ele assim: "Caso o conteúdo da variável \$1 for um destes que eu tenho aqui, você deve ir até a função correspondente e executar os comandos dela."

Como o interpretador tinha o conteúdo de \$1 (que foi passado na linha de comando) o case orientou-o para o que ele deveria fazer.

Case...ops...Caso você tenha um script que seja um tanto grande e possa ser dividido em vários pedaços, é recomendadíssimo criar funções desta maneira.

Você pode ainda criar no começo do script uma "sessão de testes" para acusar erros se não for especificado nenhum parâmetro em \$1, se o conteúdo de \$1 é diferente dos parâmetro aceitos pelo programa, ainda se o programa usa um determinado arquivo, executar os mesmos testes com \$2...enfim, basta ter uma lógica boa na cabeça e listar todas as possibilidades de falhas.

Aqui vão alguns exemplos comentados de testes de parâmetros para as variáveis \$1 \$2:

```

#( Testes de parâmetro ).....#

#[ Se não for passado nenhum parâmetro ].....#

[ -z $1 ] && echo "
    $0: Parametro Inválido: -h para ajuda
    " && exit

#[ Se o parâmetro não for reconhecido pelo programa ].....#

if [[ -n $1 && $1 != -w && $1 != -h && $1 != -t ]]; then

    echo "
    $0: Parametro Inválido: -h para ajuda
    " && exit

```

```

else

#[ Se os parâmetros estiverem OK mas o $2 estiver zerado ].....#

    if [[ -z $2 && $1 = -w ]] || [[ -z $2 && $1 = -t ]]; then

        echo "
        $0: Falta de Parametros: -h para ajuda
        " && exit

    fi

fi

```

Existem algumas maneiras mais simplificadas para efetuar esses testes de perguntas múltiplas, são elas:

```

if [[ $1 == -[1-9] ]];then # testa parâmetros numéricos de 1 até 9.

teste=$(echo $1 | sed '/^\(1\|2\|3\|4\|5\|6\|7\|8\|9\)$/!d') # analisando o parâmetro
[ $teste ] && echo "$teste" # antes usando sed

teste=$(echo $1 | sed '/^\([1-9]\)$/!d') # se os parâmetros forem
[ $teste ] && echo "$teste" # somente numéricos, dá para
# simplificar mais ainda

```

Você pode ainda criar uma função de nome help que ira mostrar ao usuário do programa quais as opções de parâmetros que o programa aceita. Todos esses detalhes é que diferencias um bom script de um não tão bom assim.

Tudo sobre redirecionamento

Mostrarei um poucos sobre a teoria de redirecionamento e algumas referências rápidas. Dando mais ênfase para as saídas, pois tem um campo maior de trabalho.

Existem 3 descrições de "arquivos", stdin (arquivo de entrada), stdout (saída), stderr (erro), o "std" vem de standard, ou padrão. significa que você pode redirecionar arquivos de entrada, obter saídas ou receber mensagens de erro. Explicando melhor, basicamente você pode:

- Redirecionar o stdout para um arquivo;
- Redirecionar o stderr para um arquivo;
- Redirecionar o stdout para o stderr;
- Redirecionar o stderr para o stdout;
- Redirecionar o stderr e o stdout para um arquivo;
- Redirecionar o stderr e o stdout para o stdout;
- Redirecionar o stderr e o stdout para o stderr.

O trabalho do stdin é bastante fácil de compreender, pois simplesmente redireciona o resultado da entrada padrão (teclado) para outro lugar desejado, por exemplo:

```
read -sp "Entre com um número: " NUM
```

Esta linha declara , dentro de um script, a variável "NUM" já com o valor especificado pelo comando read que redireciona o resultado da entrada padrão. Podemos dar como exemplo de stdin as linhas a seguir:

```
ARQUIVO=$1

echo $ARQUIVO | less
```

Aqui pegamos o arquivo que foi especificado em "\$1" e mostramos gradativamente seu conteúdo com o comando less.

Em resumo, o stdin é quem se encarrega de mandar informações externas para dentro de um script, por exemplo, usando a entrada padrão (teclado).

Seguindo em frente com os conceitos de redirecionamento, veremos agora exemplos de stdout e stderr.

- stdout para um arquivo:

```
ls -l > ls-l.txt
```

Este comando irá redirecionar a saída padrão do comando ls para um arquivo texto, sem exibir nada na tela do terminal, graças ao redirecionador ">".

- stderr para um arquivo:

```
grep da * 2> grep-erro.txt
```

Aqui foi causado um erro proposital do comando grep e a mensagem de erro foi redirecionada para um arquivo texto usando "2>" que pega somente as mensagens de erro e manda para o arquivo "grep-erro.txt", sendo que qualquer outra coisa vai ser mostrada na tela.

- stdout para o stderr:

```
grep da * 1>&2
```

Usando o mesmo exemplo, qualquer resultado apresentado na tela que for considerado stdout, ou saída, será redirecionado para stderr que normalmente não aparece.

- stderr para stdout:

```
grep da * 2>&1
```

Aqui temos uma inversão, qualquer eventual erro que possa ocorrer aparecerá, porque a saída de erro também será enviada para a saída padrão.

- stdout e stderr para um arquivo:

```
rm -f $(find / -name core) &> /dev/null
```

Este comando irá vasculhar os diretórios atrás de um arquivo com nome "core" e irá remove-lo de onde estiver, tendo como qualquer eventual saída, tanto de erro quanto padrão, redirecionada para o famoso "buraco negro" do sistema Linux, o diretório "/dev/null".

Agora vamos dar uma rápida olhada no que fazem os comandos >, >>, < e | nesta brincadeira de redirecionamento.

- um sinal de maior: >

Utilizando somente um sinal de maior o redirecionamento entende que a saída deve ser enviada para um arquivo com nome a ser especificado depois dele.

- dois sinais de maior: >>

Tem a mesma função do sinal sozinho, mas com uma vantagem, o sinal de maior sozinho (caso seja usado para escrever em um arquivo já existente), vai subscrever o arquivo apagando o conteúdo anterior, utilizando ">>" o redirecionamento entende que o texto será escrito a partir da ultima linha do arquivo não perdendo o conteúdo existente.

- sinal de menor: <

Este comando faz com que um caminho inverso seja utilizado, ao invés de redirecionar para um arquivo, ele vai redirecionar o conteúdo de um arquivo, por exemplo, para um comando, vejamos:

```
less < arquivo.txt
```

Esta linha vai jogar o conteúdo de "arquivo.txt" para o comando less, que irá mostra-lo, aparentemente parece bobo, pois obteríamos o mesmo resultado com "less arquivo", mas sabendo desse conceito podemos criar algo mais robusto.

```
- o "pipe": |
```

O bom e velho "pipe" que vemos tantas vezes tem, como o próprio nome diz, a função de servir de encanamento, redirecionando a saída de um comando para outro afim de filtrar, ou melhor dizendo, lapidar o resultado a ser mostrado na saída padrão. Veja o exemplo com o comando ifconfig:

```
prompt> ifconfig eth0
```

```
eth0      Link encap:Ethernet  HWaddr 00:0x:xx:xx:xx:F1
          inet addr:192.168.000.000  Bcast:122.168.000.000  Mask:255.255.0.0
          inet6 addr: fexx::xxx:xxff:fexx:11f1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

Um pouco confuso não é?

Se nós usássemos alguns comando que vão nos mostrar somente o endereço de IP da máquina? Isso só é possível graças ao pipe.

```
prompt> ifconfig eth0 | grep inet | head -1 | awk '{print $2}' | cut -d: -f2
192.168.000.000
```

Bem melhor, não concorda? Vamos ver o que cada um dos comandos fez?

```
O ifconfig mostra a configuração da placa de rede "eth0";
O grep filtra essa informação mostrando apenas as linhas com a ocorrência "inet";
> inet addr:192.168.000.000 Bcast:122.168.000.000 Mask:255.255.0.0
> inet6 addr: fexx::xxx:xxff:fexx:11f1/64 Scope:Link
O head mostra somente , com a opção '-1', a primeira linha das duas que foram
selecionadas;
> inet addr:192.168.000.000 Bcast:122.168.000.000 Mask:255.255.0.0
O awk pega a segunda coluna da linha com a opção '{print $2}';
> addr:192.168.000.000
E o cut faz o último filtro com as opções '-d:' e '-f2', que vão pegar o segundo
campo, separado pelo delimitador ':'.
> 192.168.000.000
```

Portanto podemos concluir que sabendo os comandos corretos e para onde queremos enviar os resultados, os redirecionamentos podem ser uma ferramenta poderosa na confecção de um bom script.

Aprendendo a usar o hold space do sed

O hold space do sed, como o próprio nome diz, é um espaço reservado, uma espécie de buffer que o sed usa para armazenar uma determinada informação, que pode ser exibida ao final de um processo de execução. Vejamos:

Vamos tomar como exemplo o arquivo "/etc/shells"

```
prompt> cat /etc/shells
# /etc/shells: valid login shells
/bin/ash
/bin/bash
/bin/csh
/bin/sh
/usr/bin/es
/usr/bin/ksh
/bin/ksh
/usr/bin/rc
/usr/bin/tcsh
/bin/tcsh
```

```
/usr/bin/zsh
/bin/sash
/bin/zsh
/usr/bin/esh
/bin/rbash
/bin/dash
```

```
prompt> sed '/\bash/H;$g' /etc/shells
# /etc/shells: valid login shells
/bin/ash
/bin/bash
/bin/csh
/bin/sh
/usr/bin/es
/usr/bin/ksh
/bin/ksh
/usr/bin/rc
/usr/bin/tcsh
/bin/tcsh
/usr/bin/zsh
/bin/sash
/bin/zsh
/usr/bin/esh
/bin/rbash

/bin/bash <----
```

Aqui o sed fez o seguinte, armazenou, com a opção H, no hold space a expressão '/bash' e a exibiu o resultado após uma quebra de linha.

"Legal, mas para que serve isso mesmo?"

Serve para capturar informações de linhas diferentes e exibí-las na mesma linha, quem nunca precisou de informações em linhas separadas e apanhou para mostrá-las com o sed?

"Tá, mas você exibiu em linhas diferentes..."

Sim, eu sei, mas isso foi só para mostrar o modo de execução do hold space, o melhor ainda está por vir.

Vamos dar uma olhada nas opções do hold space, e para que elas servem:

h	guarda no espaço reserva;
H	guarda (anexando) no espaço reserva;
g	pega o conteúdo do espaço reserva;
G	pega (anexando) o conteúdo do espaço reserva;
x	troca os conteúdos dos 2 registradores;

Conhecendo estas opções, vamos comentar o comando usado acima:

```
sed '/\bash/H;$g' /etc/shells
```

"/\bash/H" - Pega o resultado da expressão e armazena com a opção H.

"\$g" - Exibe o resultado da expressão ao final do processo.

Lembra que acima foi dito que o hold space exibe o resultado após uma quebra de linha?

Pois é, agora que a coisa fica legal, vamos pegar um arquivo para exemplos:

```
prompt> teste.txt
```

```
=====
Dados do periodo de 01-JAN-06 ate 12-APR-06
TAG: $HSTU
=====
22.983971      01/01/2006 00:01:00
23.029369      01/01/2006 00:02:00
23.037775      01/01/2006 00:03:00
22.997421      01/01/2006 00:08:00
22.962111      01/01/2006 00:09:00
22.962111      01/01/2006 00:29:00
```


Vamos supor que nós queremos pegar a informação contida na linha "TAG" e exibi-la em conjunto com as linhas que começam com "22". Podemos fazer da seguinte maneira.

```
prompt> cat teste.txt | sed -n '3h;G;s/^\(22\..*\)[[:space:]]\n\(.*\)\n\(.*\)/\3 - \1 \2/p'

TAG: $HSTU - 22.983971      01/01/2006 00:01:00
TAG: $HSTU - 22.997421      01/01/2006 00:08:00
TAG: $HSTU - 22.962111      01/01/2006 00:09:00
TAG: $HSTU - 22.962111      01/01/2006 00:29:00
```

O que essa expressão fez aqui foi separar a linha desejada, filtrar as linhas que começam com 22, e mostrar tudo formatado, repare que no final da expressão, antes dos retrovisores foi recuperado o conteúdo da linha "TAG", pois como eu havia dito, ele exibe o hold space ao final de uma quebra de linha "\n".

Uma abordagem mais completa pode ser encontrada em:

<http://aurelio.net/sed/sed-HOWTO/sed-HOWTO-8.html#ss8.4>

Técnica de recursividade em Shell Script

Já que estamos falando em funções e bons scripts esta é uma boa hora para falar a respeito de uma técnica "elegante" de programação que existe em várias linguagens, e claro também existe em shell script. Se chama técnica de recursividade, ou técnica recursiva.

A recursividade existe a muito tempo na matemática, usada em fatoração por exemplo, e consiste em resolver um problema grande quebrando-o em vários problemas menores que vão ser resolvidos separadamente e juntos no final darão o resultado do problema grande...hum...complicou não é?

Vou passar um exemplo matemático para ficar um pouco mais claro:

Queremos saber o resultado de 5! (cinco fatorial), a recursividade usa o princípio de "dividir e conquistar", vejamos:

```
120 <-
5! = 5 x 4! =120<--
      |
      4 x 3! =24<--
          |
          3 x 2! =6<--
              |
              2 x 1! =2 <--
                  |
                  1 x 0! =1 <--
                      |
                      -----> 1
```

Em palavras quer dizer... 5! é igual a 5 x 4! , 4! é igual a 4 x 3!, e assim por diante, até chegar no termo mais fácil de se resolver 0! que é igual a 1, a partir daí tendo o resultado mais fácil, vamos subindo e resolvendo até chegar na resolução do problema desejado, ou seja 5! = 120.

Um algoritmo de função recursiva funciona mais ou menos da mesma maneira para resolver um problema, dentro da função existe uma linha que chama ela mesma e resolve o problema da instancia menor para a maior, mostrando o resultado final. Isto nada mais é do que uma repetição (como for e while) mas implícita, você resolve com menos linhas de código. O objetivo de uma função recursiva é fazer com que ela passe por uma seqüência de chamadas até que um certo ponto seja atingido.

Aplicando o mesmo problema em um script ficaria assim:



Script: fatorialR.bsh

Script que calcula o fatorial de um número usando recursividade.

```
#!/bin/bash

#
# função recursiva calculando fatorial
#

[ -z $1 ] && echo "$0:Erro:use $0 0 [numero]" && exit

fat(){
if [ $1 -ge 1 ]; then
    echo $(( `fat $(( $1 - 1 ))` * $1 ))
else
    echo 1
fi
}

echo `fat $1`
```

Este exemplo é um exemplo clássico de recursividade aplicado em aulas de Estrutura de Dados, assim como o da sequência de Fibonacci. Esta técnica não é muito difundida, pois para scripts simples sua funcionalidade é pouca. Recursividade costuma ser bastante usada em Inteligência Artificial. Mas caso você queira se aprofundar mais nesse assunto e testar algumas coisas, tenha sempre em mente que é preciso ter um condicional de controle "if" para reduzir a recursão até a menor instância, e se você não estiver trabalhando com números, é preciso de um contador, e claro, inserir o nome da função dentro dela mesma, de preferência em uma sub-shell.

Veja agora um exemplo simples de recursividade, um script criado para exibir na tela barras de progresso que mostram a situação da memória e do swap do computador.

```
#!/bin/bash

#####[ Ficha: ]#####
#
# Script: Memory View Recursivo - versão 0.0
# Escrito por: Reinaldo Marques de Lima
# Criado em: 23/06/2006
# Ultima Atualizacao: 23/06/2006
#
#####[ Descricao: ]#####
#
# Script que gera uma barra horizontal que exibe a porcentagem de memoria #
# e swap em uso.
#
#####[ Changelog: ]#####
#
# Versao 0.0 - Nasceu!
#
#####

clear

#( Looping Perpetuo com Recursividade ).....#

funcao(){
```

```

#( Variaveis ).....#

SPACE1=" "
SPACE2=" "
SPACE3=" "
SPACE4=" "
MEM=$(free | grep Mem: | awk '{print $2}')
MEMF=$(free | grep Mem: | awk '{print $3}')
SWAP=$(free | grep Swap: | awk '{print $2}')
SWAPF=$(free | grep Swap: | awk '{print $3}')
CAMPO1=$((MEM/5000))
CAMPO2=$((MEMF/5000))
CAMPO3=$((SWAP/10000))
CAMPO4=$((SWAPF/10000))
SOBRA1=$((CAMPO1-CAMPO2))
SOBRA2=$((CAMPO3-CAMPO4))

#( Sequencia de lacos pra gerar as barras ).....#

for i in `seq $CAMPO1`; do

    SPACE1=${SPACE1/ / }

done
for j in `seq $SOBRA1`; do

    SPACE2=${SPACE2/ / }

done
for i in `seq $CAMPO3`; do

    SPACE3=${SPACE3/ / }

done
for j in `seq $SOBRA2`; do

    SPACE4=${SPACE4/ / }

done

#( Imprime as barras ).....#

echo -e "Situacao da
memoria:\nMemoria\n0%\e[42m[$SPACE1\e[m$SPACE2]100%\nSwap\n0%\e[42m[$SPACE3\e[m$SPACE4]10
0%"

sleep 0.5 && clear

#( Captura o fim do looping e exibe a ultima mensagem ).....#

trap "clear; echo fim da conexão, Tchau; exit" 0 2 3

funcao #[ Aqui entra a recursividade, pois a funcao chama a si mesma ]....#
}
funcao

#( Fim ).....#

```



Algumas dicas aleatórias

Vou agora listar alguns exemplos prontos que você pode usar em seu script, é aquela sessão de consulta rápida que você lê por cima a primeira vez, e volta a ler com mais atenção caso você tenha alguma dúvida ou precise de algo.

Passar senha para script sem imprimir na tela:

```
read -sp "senha: " pw
```

Conexão ssh para se usar em script, nome do servidor vai em \$1

```
xterm -T $1 -e ssh -vv (usuário)@$1
```

Usando a tecla de backspace no case

```
`echo -e "\b"` comando ;;
```

Criando um arquivo de backup tar.gz de um diretório

```
tar -zcf (nome do arquivo).tar.gz (diretorio)
```

Contando quantos arquivos o tar tem

```
tar -ztvf arquivo.tar.gz | wc -l
```

Mostrando somente o ip da maquina, filtra ifconfig

```
ifconfig -a | grep inet | head -1 | awk '{print $2}' | cut -d: -f2
```

Lendo arquivos compactados

```
gunzip -dc arquivo | less
```

Testando consistência da compactação

```
gzip -t
```

Removendo extensão de um arquivo

```
sed 's/\..*$//' arquivo.txt  
arquivo
```

Remover <tags>

```
sed 's/<[^>]*>//g' arquivo
```

Removendo tags do lynx com a opção '-dump'

```
sed 's/\[[^]]*\]//g'
```

Criando Barra de Progresso

```
t=0;echo -n " 0 ";while [ $t -lt 100 ];do t=$((t+1));\  
echo -ne "\e[${#t}D#t";sleep 1;done
```

Formatando o seq para completar a sequência com zeros

```
seq -f %03.0f 'número'
```

```
# Completando sequência com zeros, em sed e awk

# sed

sed 's/[0-9]/0000&;s/[0-9]\{2\}/000&;s/[0-9]\{3\}/00&;s/[0-9]\{4\}/0&' arquivo.txt

# awk

cat arquivo.txt | awk '{printf("%.5d\n",$1)}'

# Acompanhando o crescimento de um arquivo

tail -f

# Para criar uma lista randomica não linear de cem números

for i in `seq 100`; do echo $(( RANDOM % $i )); done
```



Capítulo 2

Dicas de Scripts integrados com:

- Navegador Lynx
 - conhecendo o navegador lynx;
 - Onde conseguir o Lynx;
 - comandos mais usuais;
 - Opções do Menu Inicial;
 - Algumas opções de Parâmetros;
 - Variáveis de Ambiente;
 - Executando cgi's em Lynx, você sabia?
 - O arquivo lynx.cfg
 - Algumas Dicas;
 - Efetuando pesquisas;
 - Exemplos de Shell-script com Lynx;
 - Outros Navegadores;
- Open Office
 - recebendo parâmetros pela linha de comando;
 - redirecionando saídas de scripts para o Open Office.



Navegador

Lynx

Conhecendo o Navegador Lynx

O Lynx é um navegador em modo texto onde você pode se conectar a qualquer site da internet pelo seu terminal de trabalho por exemplo, ele foi desenvolvido na Universidade do Kansas - Estados Unidos para, dentre outras tarefas, possibilitar uma navegação rápida caso se queira ler o conteúdo de uma página sem precisar abrir outro navegador. A seguir veremos um pouco mais que o lynx pode oferecer.

Onde conseguir o Lynx

Caso você não tenha o Lynx instalado em seu computador ele pode ser baixado da internet gratuitamente em instalado manualmente através dos links:

Site oficial -

<http://lynx.browser.org/>

Última versão estável (até a ultima atualização deste documento) -

<http://lynx.isc.org/lynx2.8.5/index.html>

Versão em testes (não estável) -

<http://lynx.isc.org/current/>

Mirrors -

<http://lynx.isc.org/mirrors.html>

Se você utiliza o Debian, ou alguma variação como por exemplo o Ubuntu, basta, na linha de comando como root, digitar:

```
prompt> apt-get install lynx
```

Usando o "smart":

```
prompt> smart install lynx -y
```

Depois de finalizada a instalação, basta chama-lo pela linha de comando:

```
prompt> lynx
```

E Para obter ajuda utilize:

man lynx, lynx -help e info lynx

E seu arquivo de configuração fica em:

/etc/lynx.cfg

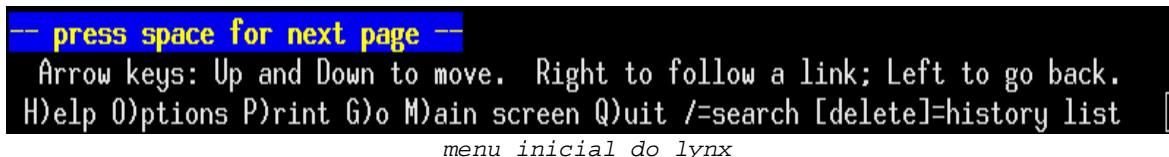
Existe ainda a versão do lynx para DOS, bom para quem trabalho com os dois ambientes e precisa desse recurso. Ele pode ser conseguido em:

<http://www.fdisk.com/doslynx/lynxport.htm>

Comandos mais usuais

Opções do Menu Inicial

Ao iniciar o lynx chamando o programa na linha de comando aparecerá uma página com alguns links de ajuda do navegador, para escolher algum deles basta mover a seta do teclado para cima ou para baixo, seta para direita para seguir o link e seta para esquerda para voltar.



```
-- press space for next page --
Arrow keys: Up and Down to move. Right to follow a link; Left to go back.
H)elp O)ptions P)rint G)o M)ain screen Q)uit /-search [delete]=history list
menu inicial do lynx
```

Caso queira que seja exibido alguns tópicos de ajuda digite no teclado **"h"** (de help) que, caso você esteja navegando em algum site, mostrará a tela inicial do lynx como se você tivesse ativado ele sem nenhum endereço.

Para buscar algum conteúdo na página a qualquer momento digite **"/"** seguida do que se deseja achar e de **<ENTER>**, esta opção é muito parecida com a do **VI**.

Para acessar algum site primeiro digitamos **"g"** (de Go) no teclado e em seguida digitamos o endereço da URL.

Digitando a letra **"p"** mostramos as opções de impressão, onde podemos salvar a página, envia-la por e-mail, etc.

Caso não tenha sido especificado nenhum parâmetro na chamada do programa, digitando **"o"** teremos as opções de configuração do navegador.

Quando a navegação está bastante avançada e você precisa voltar a página inicial basta digitar **"m"** para voltar tudo.

Podemos também especificar uma URL logo na chamada do programa, como por exemplo:

```
user@linux~$ lynx www.google.com
```

Pagina do Google

```
Google
Pgna inicial personalizada | Efetuar login

Google

Web  Imagens  Grupos  Diretro  Noticias?Novo!  mais ^J
-----
Pe
Pesquisa Google Estou com sorte
Ferramentas de idiomas
Pesquisar: ( ) a web ( ) pgnas em portugus( ) pgnas do Brasil
            (*)      ( )
Solue de publicidade - Tudo sobre o Google - Google.com in English

206 Google

(NORMAL LINK) Use right-arrow or <return> to activate.
Arrow keys: Up and Down to move. Right to follow a link; Left to go back.
H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

Caso nenhum parâmetro seja especificado, antes de acessar o site o navegador pergunta se quer aceitar cookies da página à ser acessada.

Algumas Opções de Parâmetros

Aqui veremos alguns opções de parâmetros que podem ser inseridos na linha de comando junto com a chamada do programa para que sejam realizadas determinadas ações. São elas:

-accept_all_cookies - Use este parâmetro para que o lynx aceite todos os cookies que a página tenta enviar, sabemos que cookies são sempre um assunto polêmico se tratando de internet, vai de cada um aceitar os cookies da página ou não.

-book - Habilita uma página em html como bookmark, sendo a primeira página chamada ao iniciar o navegador.

-case - Habilita o modo case-sensitive enquanto o lynx estiver ativo nessa navegação.

-center - Centraliza o texto da página.

-color - Usa, se estiver disponível, o modo colorido que diferencia por cores os links, imagens, textos, etc...

-connect_timeout=N - Define um tempo de conexão, onde N é o tempo em segundos.

-dump - Modo que retira o texto da página desabilitando os tags do HTML, muito bom para ser usado em scripts.

-emacskeys - Usa o padrão do editor emacs para navegação.

-error_file=FILE - Define um arquivo onde o lynx vai reportar os códigos de erro HTTP.

-homepage=URL - Define um endereço URL como página inicial.

-index=URL - Define um endereço URL como página inicial, não necessariamente a index da página web.

-justify - Justifica o texto da página.

-nolist - Desabilita a lista de links destacados no dump.

-source - Faz a mesma coisa que o dump, mas mantém as características de HTML da página.

-use_mouse - Permite, se disponível a utilização do mouse na navegação.

-vikeys - Usa o padrão do editor vi para navegação.

-width=NUMBER - Especifica o tamanho de colunas para formatação do dump, por padrão o número é 80.

Variáveis de Ambiente

Neste tópico abordarei algumas das principais variáveis de ambiente do lynx, que podem ser habilitadas para que o navegador adote um determinado comportamento, listarei as variáveis que mais se enquadram no propósito deste tutorial, que é a utilização do navegador lynx em sincronia com scripts em shell.

Podemos dizer que este tópico é triplo, pois ao final dele abordarei um pouco sobre cgi's no lynx, que vai ser visto com mais ênfase no próximo tópico, e como isto está altamente relacionado com o arquivo de configuração do lynx, este também será abordado com calma mais adiante.



NOTA: A maioria das variáveis de ambiente listadas aqui ,por padrão, não estão ativas. Para atribuir um valor basta executar normalmente em linha de comando, use:

```
prompt> VARIABEL=valor
```

E para torna-la global use:

```
prompt> export $VARIABEL
```

Variável	Descrição
LYNX_CFG	Utiliza outro arquivo de configuração para o navegador, diferente do arquivo localizado em /etc/lynx.cfg.
MAIL	Esta variável, se declarada, irá especificar uma caixa de entradas onde o lynx vai verificar a existência de novos emails.

Existem algumas outras variáveis de ambiente, mas como eu havia dito, para o proposito de criação de scripts, não são necessárias muitas configurações no navegador.

Caso as regras de execução de cgi's for configurada no arquivo lynx.cfg, as seguintes variáveis de ambiente em cgi passam a ser aceitas também pelo navegador lynx:

CONTENT_LENGTH

CONTENT_TYPE

DOCUMENT_ROOT

HTTP_ACCEPT_CHARSET

HTTP_ACCEPT_LANGUAGE

HTTP_USER_AGENT

PATH_INFO

PATH_TRANSLATED

QUERY_STRING

REMOTE_ADDR

REMOTE_HOST

REQUEST_METHOD

SERVER_SOFTWARE

Executando cgi's em Lynx, você sabia?

Pois é meus amigos, é a mais pura verdade, quando você está criando seu script, independente da linguagem, para ser usando como cgi para uma página web, as vezes fica um tanto complicado ficar toda hora abrindo o navegador, e testando o cgi, para tornar as coisas mais dinâmicas eis que surge mais uma vez o lynx para o salvamento. A principio a configuração padrão do lynx aceita a execução de qualquer cgi, basta chama-lo pela linha de comando:

```
prompt> lynx http://localhost/cgi-bin/dir\_teste/script.cgi
```

Mas para efeito de segurança, e também para ficar mais fácil de aplicar algumas regras na execução do cgi o lynx aceita, através de seu arquivo de configuração, que se especifique algumas regras. Eis aqui o trecho do arquivo que precisa ser editado para aplicar estas regras:

```
.h1 CGI scripts
# These settings control Lynx's ability to execute various types of scripts.

.h2 LOCAL_EXECUTION_LINKS_ALWAYS_ON
.h2 LOCAL_EXECUTION_LINKS_ON_BUT_NOT_REMOTE
# Local execution links and scripts are by default completely disabled,
# unless a change is made to the userdefs.h file to enable them or
# the configure script is used with the corresponding options
# (--enable-exec-links and --enable-exec-scripts).
# See the Lynx source code distribution and the userdefs.h
# file for more detail on enabling execution links and scripts.
#
# If you have enabled execution links or scripts the following
# two variables control Lynx's action when an execution link
# or script is encountered.
#
# If LOCAL_EXECUTION_LINKS_ALWAYS_ON is set to TRUE any execution
# link or script will be executed no matter where it came from.
# This is EXTREMELY dangerous. Since Lynx can access files from
# anywhere in the world, you may encounter links or scripts that
# will cause damage or compromise the security of your system.
#
# If LOCAL_EXECUTION_LINKS_ON_BUT_NOT_REMOTE is set to TRUE only
# links or scripts that reside on the local machine and are
# referenced with a URL beginning with "file://localhost/" or meet
# TRUSTED_EXEC or ALWAYS_TRUSTED_EXEC rules (see below) will be
# executed. This is much less dangerous than enabling all execution
# links, but can still be dangerous.
#
LOCAL_EXECUTION_LINKS_ALWAYS_ON:FALSE
LOCAL_EXECUTION_LINKS_ON_BUT_NOT_REMOTE:FALSE

.h2 TRUSTED_EXEC
# If LOCAL_EXECUTION_LINK_ON_BUT_NOT_REMOTE is TRUE, and no TRUSTED_EXEC
# rule is defined, it defaults to "file://localhost/" and any lynxexec
# or lynxprog command will be permitted if it was referenced from within
```

```

# a document whose URL begins with that string.  If you wish to restrict the
# referencing URLs further, you can extend the string to include a trusted
# path.  You also can specify a trusted directory for http URLs, which will
# then be treated as if they were local rather than remote.  For example:
#
#     TRUSTED_EXEC:file://localhost/trusted/
#     TRUSTED_EXEC:http://www.wfbr.edu/trusted/
#
# If you also wish to restrict the commands which can be executed, create
# a series of rules with the path (Unix) or command name (VMS) following
# the string, separated by a tab.  For example:
#
# Unix:
# ====
#     TRUSTED_EXEC:file://localhost/<tab>/bin/cp
#     TRUSTED_EXEC:file://localhost/<tab>/bin/rm
# VMS:
# ===
#     TRUSTED_EXEC:file://localhost/<tab>copy
#     TRUSTED_EXEC:file://localhost/<tab>delete
#
# Once you specify a TRUSTED_EXEC referencing string, the default is
# replaced, and all the referencing strings you desire must be specified
# as a series.  Similarly, if you associate a command with the referencing
# string, you must specify all of the allowable commands as a series of
# TRUSTED_EXEC rules for that string.  If you specify ALWAYS_TRUSTED_EXEC
# rules below, you need not repeat them as TRUSTED_EXEC rules.
#
# If EXEC_LINKS and JUMPFILE have been defined, any lynxexec or lynxprog
# URLs in that file will be permitted, regardless of other settings.  If
# you also set LOCAL_EXECUTION_LINKS_ON_BUT_NOT_REMOTE:TRUE and a single
# TRUSTED_EXEC rule that will always fail (e.g., "none"), then *ONLY* the
# lynxexec or lynxprog URLs in JUMPFILE (and any ALWAYS_TRUSTED_EXEC rules,
# see below) will be allowed.  Note, however, that if Lynx was compiled with
# CAN_ANONYMOUS_JUMP set to FALSE (default is TRUE), or -restrictions=jump
# is included with the -anonymous switch at run time, then users of an
# anonymous account will not be able to access the jumps file or enter
# 'j'ump shortcuts, and this selective execution feature will be overridden
# as well (i.e., they will only be able to access lynxexec or lynxprog
# URLs which meet any ALWAYS_TRUSTED_EXEC rules).
#
TRUSTED_EXEC:none

.h2 ALWAYS_TRUSTED_EXEC
# If EXEC_LINKS was defined, any lynxexec or lynxprog URL can be made
# always enabled by an ALWAYS_TRUSTED_EXEC rule for it.  This is useful for
# anonymous accounts in which you have disabled execution links generally,
# and may also have disabled jumps file links, but still want to allow
# execution of particular utility scripts or programs.  The format is
# like that for TRUSTED_EXEC.  For example:
#
# Unix:
# ====
#     ALWAYS_TRUSTED_EXEC:file://localhost/<tab>/usr/local/kinetic/bin/usertime
#     ALWAYS_TRUSTED_EXEC:http://www.more.net/<tab>/usr/local/kinetic/bin/who.sh
# VMS:
# ===
#     ALWAYS_TRUSTED_EXEC:file://localhost/<tab>usertime
#     ALWAYS_TRUSTED_EXEC:http://www.more.net/<tab>show users
#
# The default ALWAYS_TRUSTED_EXEC rule is "none".
#
ALWAYS_TRUSTED_EXEC:none

.h2 TRUSTED_LYNXCGI
# Unix:
# =====

```

```

# TRUSTED_LYNXCGI rules define the permitted sources and/or paths for
# lynxcgi links (if LYNXCGI_LINKS is defined in userdefs.h). The format
# is the same as for TRUSTED_EXEC rules (see above), but no defaults are
# defined, i.e., if no TRUSTED_LYNXCGI rules are defined here, any source
# and path for lynxcgi links will be permitted. Example rules:
#
#     TRUSTED_LYNXCGI:file://localhost/
#     TRUSTED_LYNXCGI:<tab>/usr/local/etc/httpd/cgi-bin/
#     TRUSTED_LYNXCGI:file://localhost/<tab>/usr/local/www/cgi-bin/
#
# VMS:
# ====
# Do not define this.
#
TRUSTED_LYNXCGI:none

.h2 LYNXCGI_ENVIRONMENT
# Unix:
# ====
# LYNXCGI_ENVIRONMENT adds the current value of the specified
# environment variable to the list of environment variables passed on to the
# lynxcgi script. Useful variables are HOME, USER, etc... If proxies
# are in use, and the script invokes another copy of lynx (or a program like
# wget) in a subsidiary role, it can be useful to add http_proxy and other
# *_proxy variables.
#
# VMS:
# ====
# Do not define this.
#
#LYNXCGI_ENVIRONMENT:

.h2 LYNXCGI_DOCUMENT_ROOT
# Unix:
# ====
# LYNXCGI_DOCUMENT_ROOT is the value of DOCUMENT_ROOT that will be passed
# to lynxcgi scripts. If set and the URL has PATH_INFO data, then
# PATH_TRANSLATED will also be generated. Examples:
#     LYNXCGI_DOCUMENT_ROOT:/usr/local/etc/httpd/htdocs
#     LYNXCGI_DOCUMENT_ROOT:/data/htdocs/
#
# VMS:
# ====
# Do not define this.
#
#LYNXCGI_DOCUMENT_ROOT:

```

Para aplicar as regras necessárias para execução de cgi's pelo navegador lynx basta modificar as linhas:

```

LOCAL_EXECUTION_LINKS_ALWAYS_ON:FALSE
LOCAL_EXECUTION_LINKS_ON_BUT_NOT_REMOTE:FALSE
TRUSTED_EXEC:none
ALWAYS_TRUSTED_EXEC:none
TRUSTED_LYNXCGI:none
#LYNXCGI_ENVIRONMENT:
#LYNXCGI_DOCUMENT_ROOT:

```

O arquivo lynx.cfg

Já que estamos falando de configuração, que tal dar um passeio por alguns trechos relevantes do arquivo lynx.cfg e comentar algumas de suas funções?

Então, aperte os cintos 8^).

Antes de mais nada, quero reafirmar que listarei alguns trechos mais relevantes do

arquivo lynx.cfg, unicamente para o propósito de criação de scripts, pois o arquivo é enorme, e não faz sentido colocá-lo aqui, porque tornaria a leitura entediante, e sem sentido, além de fugir do escopo.

.h2 CONNECT_TIMEOUT

Specifies (in seconds) connect timeout. Default value is rather huge.

#CONNECT_TIMEOUT:18000

Determina o tempo de conexão do navegador, se configurado tem tempo default de 5 horas, altere para o tempo desejado, em segundos.

.h2 CASE_SENSITIVE_ALWAYS_ON

The default search type.

This is a default that can be overridden by the user!

#

#CASE_SENSITIVE_ALWAYS_ON:FALSE

Altera a definição de case sensitive, para reconhecer parâmetros maiúsculos e minúsculos.

.h2 DEFAULT_BOOKMARK_FILE

DEFAULT_BOOKMARK_FILE is the filename used for storing personal bookmarks.

It will be prepended by the user's home directory.

NOTE that a file ending in .html or other suffix mapped to text/html

should be used to ensure its treatment as HTML. The built-in default

is lynx_bookmarks.html. On both Unix and VMS, if a subdirectory off of

the HOME directory is desired, the path should begin with "/" (e.g.,

./BM/lynx_bookmarks.html), but the subdirectory must already exist.

Lynx will create the bookmark file, if it does not already exist, on

the first ADD_BOOKMARK attempt if the HOME directory is indicated

(i.e., if the definition is just filename.html without any slashes),

but requires a pre-existing subdirectory to create the file there.

The user can re-define the default bookmark file, as well as a set

of sub-bookmark files if multiple bookmark file support is enabled

(see below), via the 'o'ptions menu, and can save those definitions

in the .lynxrc file.

#

#DEFAULT_BOOKMARK_FILE:[lynx_bookmarks.html](#)

Especifica uma página em html para ser utilizada como bookmark do navegador.

.h2 BLOCK_MULTI_BOOKMARKS

If BLOCK_MULTI_BOOKMARKS is set TRUE, multiple bookmark support will

be forced off, and cannot to toggled on via the 'o'ptions menu. The

compilation setting is normally FALSE, and can be overridden here.

It can also be set via the -restrictions=multibook or the -anonymous

or -validate command line switches.

#

#BLOCK_MULTI_BOOKMARKS:FALSE

Especifica uma página em html para ser utilizada como bookmark do navegador para páginas que NÃO podem ser acessadas.

.h1 Proxy

#http_proxy:http://some.server.dom:port/

#https_proxy:http://some.server.dom:port/

#ftp_proxy:http://some.server.dom:port/

#gopher_proxy:http://some.server.dom:port/

#news_proxy:http://some.server.dom:port/

#newspost_proxy:http://some.server.dom:port/

#newsreply_proxy:http://some.server.dom:port/

#snews_proxy:http://some.server.dom:port/

#snewspost_proxy:http://some.server.dom:port/

#snewsreply_proxy:http://some.server.dom:port/

#nntp_proxy:http://some.server.dom:port/

#wais_proxy:http://some.server.dom:port/

#finger_proxy:http://some.server.dom:port/

#cso_proxy:http://some.server.dom:port/

#no_proxy:host.domain.dom

Define o endereço de servidores proxy, para diversas tecnologias.

Algumas Dicas:

Dica show, e simples. Para baixar um site da Web direto para o VI:

`:r !lynx -dump http://sitedaweb.com` (funciona com servidor local inclusive)

O navegador lynx tem algumas particularidades na sua execução em linha de comando como por exemplo:

- Colocar a linha de comando em background quando existe um endereço com um "&" no meio.

Exemplo: <http://www4.climatempo.com.br/site/espelho.php?estados=SP&pg=capitais&pc=estadao>

Para resolver isto basta colocar uma barra invertida antes do "&" negando-o como se faz em expressões regulares por exemplo, ficando desta maneira:

<http://www4.climatempo.com.br/site/espelho.php?estados=SP\&pg=capitais\&pc=estadao>

Assim a linha de comando reconhece o "&" como um caracter do próprio endereço web.

Efetuando pesquisas

Uma maneira bem legal de efetuar pesquisas em páginas como a do google por exemplo, é passar todo o preenchimento dos campos de pesquisa pelo endereço que vai na linha de comando, por exemplo:

Pelo navegador (no meu caso o Mozilla Firefox), vamos pesquisar a palavra "Linux" no site do google usando as opções de pesquisa, (Palavra Linux, páginas em português do Brasil) e mandar pesquisar. Note que a barra de endereço mostra algo assim:


"<http://www.google.com.br/search?hl=pt-BR&q=Linux&btnG=Pesquisa+Google&meta=cr%3DcountryBR>"

Agora se dermos um **Ctrl+i** (ou no navegador de sua preferência, encontre a opção "Propriedades da Página") na página inicial do google aparecerá uma caixa com informações sobre a página, selecionando a aba **Form** podemos ver umas colunas onde mostra-se algo mais ou menos assim:

Label	Field Name	Type	Current Value
	hl	hidden	pt-BR
	q	text	Linux
	btnG	submit	Pesquisa Google
	btnI	submit	Estou com Sorte
a web	meta	radio	
páginas em português	meta	radio	lr=lang_pt
páginas do Brasil	meta	radio	cr=countryBR

Ao analisar os campos que o google usa e a linha que aparece no navegador podemos perceber que o mecanismo de busca do google simplesmente completa os campos para a pesquisa, onde no campo **text** vai a palavra ou expressão que se está procurando, nada mais natural. A partir daí conseguimos criar um script para efetuar esta mesma pesquisa sem precisar do navegador, não conseguimos? "Claro que sim..."

Então vamos lá:

	<p>Script: busca_google.bsh</p> <p>Script simples que efetua pesquisas no site do google.</p>
<pre>#!/bin/bash # # Script simples que faz pesquisas no site 'http://www.google.com' # # Criado para simples didática, sem tratativas de erro ou melhorias similares # LYNX="/usr/bin/lynx" WORD=\$1 SEARCH=\$((\$LYNX -dump http://www.google.com.br/search?hl=pt-BR&q=\$WORD&btnG=Pesquisa+Google&meta=cr%3DcountryBR sed 's/\[[^\]]*\]/ /') # Um 'sedzinho' no final para remover os lixos deixados pelo lynx echo "\$SEARCH"</pre>	

Exemplos de Shell-script com Lynx

Talvez o mais conhecido exemplo que circula na internet de como utilizar o poder do lynx em um script sejam as **FunçõesZZ** <http://funcoeszz.net/> escritas em conjunto entre

o Aurélio e o Thobias, um programa que usa e abusa do lynx para buscar informações em sites e mostra-las na tela do terminal. Claro que não vou nem arriscar a fazer algo equivalente, mas tentarei dar alguns exemplos de utilização dessas duas ferramentas (lynx e shell-script) em conjunto.

Aqui mostrarei dois códigos de scripts que eu fiz para buscar notícias de sites da Web usando lynx, o script "apinfo.bsh" e o "pesquisa_grupo.bsh", e um script criado em parceria com o Julio Neves que tenta acessar um site congestionado a força bruta:

```
#!/bin/bash

####[ Ficha: ]#####

#
# Nome: Apinfo.bsh
# Escrito por: Reinaldo Marques de Lima
# Criado em: 03/2006
# Ultima atualização: 24/04/2006
#
####[ Descrição: ]#####
#
# Script que acessa o site www.apinfo.com (site de ofertas de emprego para#
# profissionais de informatica) e gera uma pagina html com as ofertas do #
# dia.
#
#####

#( Variaveis ).....#

LYNX=$(which lynx)
GREP=$(which grep)
HEAD=$(which head)
SED=$(which sed)
CUT=$(which cut)

#( Funcao apinfo, acessa o site e gera um tmp com links das ofertas ).....#

apinfo(){

URL=http://www.apinfo3.com/pp7w.htm

####[ Filtra usando o canivete ( grep, head, cut...) ]#####
#$LYNX -source $URL | $GREP \<p>\<a | $HEAD -2 | $CUT -d\" -f2 > /tmp/apinfo.tmp
#
####[ Aqui usa só o sed ]#####
```



```

$LYNX -source $URL | $SED '/<p><a href=.*!/d;
s/<p><a href="\(.*\)">/\1/g' | sed '1,2!d' > /tmp/apinfo.tmp

txt
}

#( Fucao txt acessa os link usando array, filtra a pagina e gara um txt)..#

txt(){

LINKS=$(cat /tmp/apinfo.tmp)

for i in ${LINKS[*]}; do

    $LYNX -dump $i | $SED 's/\[[^\]]*\]/g;
    /References/q' >> /tmp/apinfo.txt

done

html
}

#( Funcao html pega o txt e converte em um html, facil de visualizar )....#

html(){

DATA=$(date +%d/%m/%Y)
DATA2=$(date +%d_%m_%Y)
TEXTO=$( $SED 's/[_]\{10,\}/<HR NOSHADE SIZE=1>/g;
/References/d' /tmp/apinfo.txt)

TITLE="<TITLE>
Ofertas de emprego do dia $DATA
</TITLE>"

HEAD="<HEAD>
$title
<H1>
<CENTER>
<B>
Script que acessa o Site Apinfo.com
</B>
</CENTER>

```

```

</H1>
<H2>
<CENTER>
<B>
Ofertas de emprego do dia $DATA
<HR NOSHADE SIZE=1>
</B>
</CENTER>
</H2>
</HEAD> "

BODY="<BODY bgcolor="white">
<FONT SIZE=3 FACE=verdana>
<PRE>
$TEXT0
</PRE>
</FONT>
</BODY> "

HTML="<HTML>
$HEAD
$BODY
</HTML> "

printf "$HTML" > apinfo_$DATA2.html

rm /tmp/apinfo.tmp
rm /tmp/apinfo.txt

}

#( ultima linha, chama a primeira funcao que desencadeia o processo ).....#

apinfo

#( fim ).....#

```

```

#!/bin/bash

#####[ Ficha: ]#####
#
#

```

```

# Script: Pesquisa Grupo #
# Escrito por: Reinaldo Marques de Lima #
# Criado em: 05/03/2006 #
# Ultima Atualizacao: 05/03/2006 #
# #
#####[ Descricao: ]#####
# #
# Script que acessa a pagina do grupo de shell-script e gera uma pagina #
# em html com o resultado da pesquisa. #
# O nome do arquivo gerado eh 'pesquisa_(chave de pesquisa).html'. #
# #
# TODO: #
# Gerar pagina com todos os resultados da pesquisa, atualmente o script #
# so pega a primeira pagina de resultados. #
# #
#####

#( Teste ).....#

[ -z $1 ] && echo "erro: use $0 [palavra]: sed ( por exemplo)" && exit

#( Variaveis ).....#

WORD=$1
SED=$(which sed)
CAT=$(which cat)
LYNX=$(which lynx)
URL=http://search.gmane.org/search.php?group=gmane.org.user-
groups.programming.shell.brazil\&query=

#( Funcao 'grupo', acessa a pagina e gera um tmp ).....#

grupo(){

$LYNX -source $URL$WORD | $SED -n '/<A HREF=.*>/p' | $SED 's/<A
HREF="(\.*\)">.*\/\1\/;^http:\\\/\.*\/d;s/<B\\(\.*\)\1/' > /tmp/links.tmp

txt
}

#( Funcao 'txt', usa o tmp criado como array para gerar a pagina ).....#

txt(){

```

```

LINKS=$(CAT /tmp/links.tmp)

for i in ${LINKS[*]}; do

    $LYNX -dump $i | $SED '1,27d;
    /\[paint-list-id\.php?group=gmane\.org\.user-
groups\.programming\.shell\.b$/q;
    s/\[[^]]*\]//g' | $SED 's/ \[30\].*/<HR NOSHADE SIZE=1>/' >> /tmp/links.txt

#[ Caso aqui ^ de erro, coloque tudo em uma linha soh ].....#

done

html
}

#( Funcao 'html', pega o txt e insere tags para gerar a pagina html ).....#

html(){

TEXTO=$( $SED 's/\[_]\{10,\}/<HR NOSHADE SIZE=1>/g' /tmp/links.txt)

TITLE="<TITLE>
Resultado da Pesquisa com a palavra: $WORD
</TITLE>"

HEAD="<HEAD>
$TITLE
<H1>
<CENTER>
<B>
Script que acessa o site do grupo de Shell-Script
</B>
</CENTER>
</H1>
<H2>
<CENTER>
<B>
Resultado da Pesquisa com a palavra: $WORD
<HR NOSHADE SIZE=1>
</B>
</CENTER>

```

```

</H2>
</HEAD> "

BODY="<BODY bgcolor="white">
<FONT SIZE=3 FACE=verdana>
<PRE>
$TEXT0
</PRE>
</FONT>
</BODY> "

HTML="<HTML>
$HEAD
$BODY
</HTML> "

echo "$HTML" > pesquisa_$WORD.html

rm /tmp/links.tmp
rm /tmp/links.txt

}

#( Aqui eh chamada a primeira funcao ).....#

grupo

#( fim ).....#

```

```

#!/bin/bash

####[ Ficha: ]#####
#
# Script: force_lynx.bsh
# Escrito por: Reinaldo Marques de Lima ( Plastico ) e Julio Cezar Neves
# Criado em: 24/05/2006
# Ultima Atualizacao: 26/05/2006
#
####[ Descricao: ]#####
#
# Script que tenta acessar um determinado site repetidas vezes, ideal
# para acessar sites congestionados atraves do navegador lynx, que demora #

```

```

# menos para carregar uma pagina. #
# #
#####[ How: ]#####
# #
# O script recebe na linha de comando uma URL a ser acessada, com numero #
# definido de vezes ou nao (em caso negativo inicia-se um loopin eterno), #
# que abre um novo terminal, ja chamando o navegador lynx e a URL pela #
# linha de comando, tentando acessar a URL atraves de forza bruta. #
# Para isso o usuário precisa finalizar o terminal ao perceber a demora #
# no acesso. #
# #
#####
# #
#( Variaveis ).....#
# #
PING=$(which ping) #####
KEY=0 # indice inicial que controla o while;
URL=$1 # endereco a ser acessado;
LYNX=$(which lynx) # teste de portabilidade;
#####
# #
#( Testes ).....#
# #

[ $1 ] || {
    echo "$0: Erro: Use: $0 URL [N]: Onde N eh o numero de tentativas
(vazio=infinito).\"
    exit 1; }

ping -c1 \"$1\" > /dev/null || { echo \"URL inválida ou sem rede\"; exit 1; }

TRY=${2:-1} # Se $2 estiver definido TRY recebe $2, senao -1

trap \"clear; echo fim da conexão, Tchau; exit\" 0 2 3 # Para interromper loop
infinito

# #
#( Funcao ).....#
# #
tentar()
{
    while [ $KEY != $TRY ]; do
        xterm -T $URL -e $LYNX --accept_all_cookies $URL &> /dev/null

```

```

        let KEY++
done

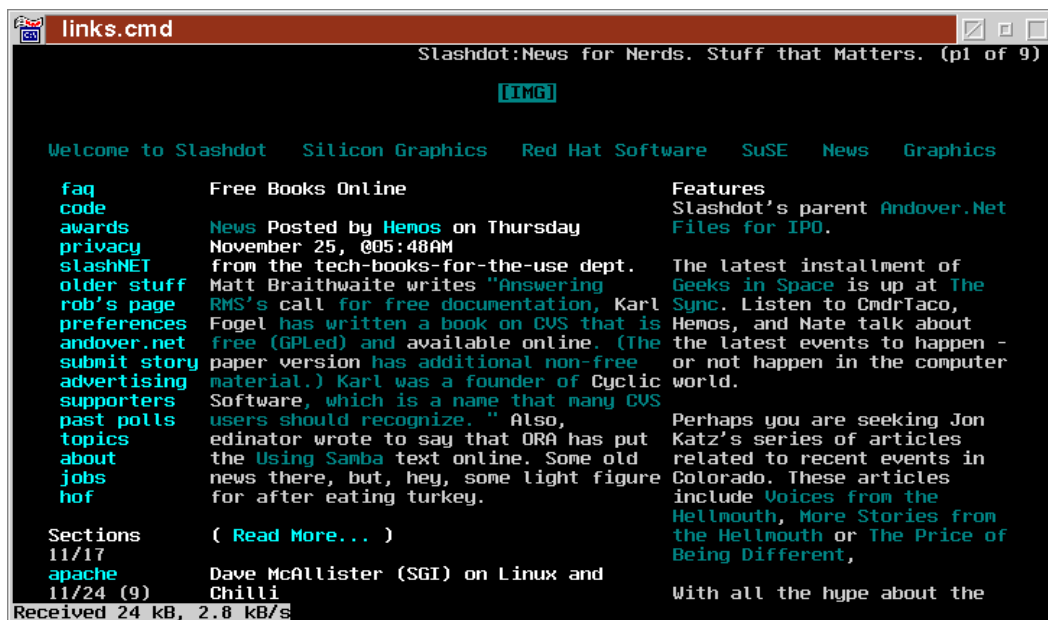
read -p "Nao foi possivel acessar $URL. Quer tentar novamente? (N/s) " RESPOSTA
RESPOSTA=${RESPOSTA:-n}
[ `echo $RESPOSTA | tr N n` = n ] &&
{
    echo "fim da conexão, Tchau"
    exit
}
KEY=0      # reinicia o indice do while
tentar     # e chama novamente a funcao
}
tentar

```

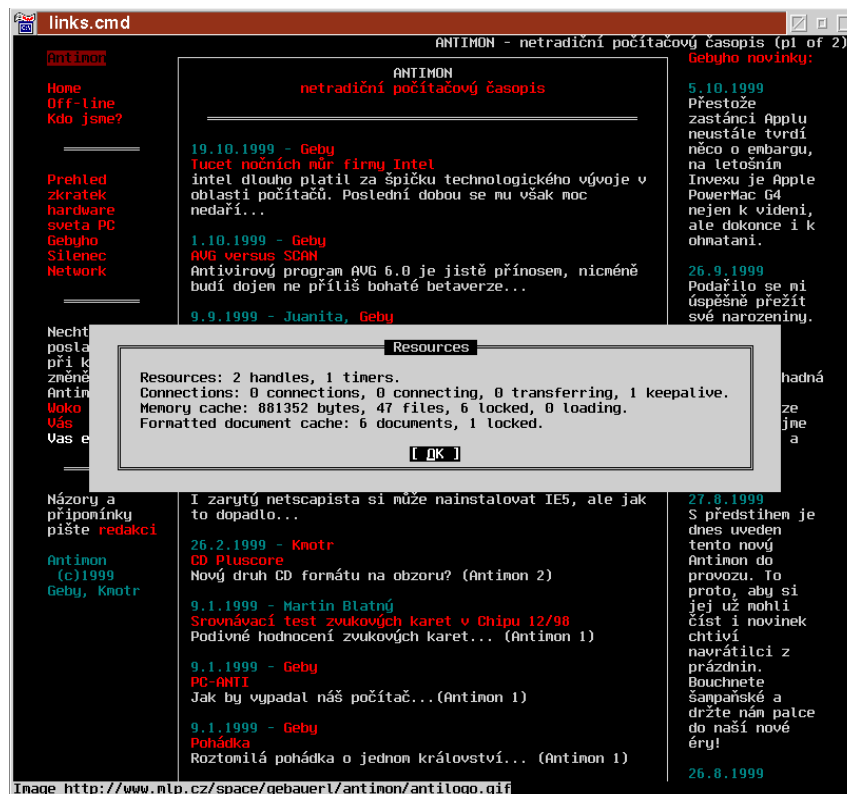
Outros Navegadores

Existem também outros navegadores em modo texto muito difundidos e comentados na internet, são o w3m e o Navegador Links (qualquer semelhança é mera coincidência?). Neste documento daremos apenas uma rápida olhada no navegador Links, pois seu conteúdo é tão grande ou até maior do que o Lynx, e uma abordagem completa daria outra apostila.

O navegador Links é um pouco mais robusto do que seu primo, com melhor visualização de tela, suporte a imagens, java-script dentre outras coisas. Vejamos dois screenshots:



A formatação da Página foi bem mais trabalhada



Inclusive ele abre até pop-up's, com a aparência do dialog

Os criadores deste navegador tiveram ótimas ideias para deixá-lo bastante robusto e com uma infinidade de opções, mas acaba revertendo isso em desvantagem, pois se ele dá suporte a java-script, visualização de imagens e outras perfumarias, por que não utilizar um navegador convencional?

Essa evolução toda fez com que se perdesse o charme e a praticidade de navegar com rapidez na web, para aqueles que tem curiosidade em conhecer mais sobre essa ferramenta, eis os "links".

Páginas:

<http://artax.karlin.mff.cuni.cz/~mikulas/links/>

<http://links.sourceforge.net/>

Source Forge

<http://links.sourceforge.net/docs/manual-0.90-en/index.html>

Manual

Download:

<http://artax.karlin.mff.cuni.cz/~mikulas/links/download/>

Grupo de Discussão:

<http://groups.yahoo.com/group/links-browser/>



A Suite de escritório Open Source mais comentada atualmente também tem sua vez através da linha de comando, e também pode ser sincronizado com um script, por exemplo, que exiba a saída de um texto diretamente no editor de texto do Open Office.

Recebendo parâmetros pela linha de comando

Primeiramente vamos ver algumas chamadas simples diretamente da linha de comando do terminal para abrir documentos no Open Office.

chamando arquivos texto

As extensões de arquivos em formato texto que o Open Office reconhece são:

- Estrangeiros (.txt .doc .vor .rtf .html)
- Nativos da própria Suit (.sxw .stw .sdw)
- Formatos Unix-like que ele reconhece como texto (.bsh .sh .ksh ...etc)



Aviso: O Open Office sempre exige que, para ser aberto, o arquivo necessite de uma extensão.

Agora sim, podemos chamar o programa pela linha de comando. Vamos supor que você queira abrir um texto que você está editando, executamos o comando assim:

```
prompt> openoffice -oowriter texto.txt
```

Ao executar este comando o Open Office começa a executar o programa (openoffice) especificando o editor de texto (-oowriter) e também o arquivo a ser aberto (texto.txt). Legal não é?

Sabendo disso você pode associar outros aplicativos do Open Office baseado nesse comando, e de lambuja eu ainda vou te dar a lista dos aplicativos e as extensões que eles reconhecem.

São eles:

ooffice

(Este abre a suite sem nenhum aplicativo nem arquivo específico)

oocalc

(Programa para criação de planilhas)

Extensões:

- Estrangeiras (.dif .dbf .vor .slk .csv .txt .html)
- Nativas (.sdc .sxc .stc)
- Proprietárias (.xls .xlw .xlt)

oodraw

(Editor de imagens)

Extensões:

- Estrangeiras (.vor)
- Nativas (.sxd .std. sda .sdd)
- Mais conhecidas (.gif .jpg .jpeg .png .bmp .png ...etc)

ooweb

(Editor de páginas de internet)

Extensões:

- Estrangeiras (.vor .txt .htm .html)
- Nativa (.stw)

As opções ooimpress, oomath e oofrontemplate são executadas pelo próprio Open Office, sendo que se chamarmos estas opções pela linha de comando todas irão abrir o Open Office Writer. Neste exemplo não foram listados os comandos e extensões para o

gerenciador de banco de dados, nem o editor de apresentações.



Dica: Todos os documentos editados pelo Open Office podem ser convertidos para documentos em formato .pdf usando a opção exportar.

Por analogia formulamos a pergunta: Se o Open Office abre documentos pela linha de comando, meu visualizador de pdfs (xpdf, gpdf, kpdf) também consegue fazer isso?

R.: Claaaaaaaro, basta usar:

```
prompt> [gkx]pdf arquivo.pdf
```

Redirecionando saídas de scripts para o Open Office

Já que a finalidade maior desta apostila é criar scripts que facilitem nossa vida, vamos encerrar todo o falatório chato e ir direto ao ponto.

Podemos fazer scripts que interajam com o Open Office?

R.: Sim, vamos aos exemplos...



Script: abre_arquivo_oo.bsh

Script que filtra um arquivo html, converte para texto e abre-o no OpenOffice.

```
#!/bin/bash

#####
#
# Exemplo de script que interage com o Open Office
#
#####

sed 's/<[^>]*>/g' arquivo.html > arquivo.txt; openoffice -oowriter arquivo.txt &

#(fim).....#
```

O script acima filtra com o sed todos os tags em html de um arquivo e escreve em um arquivo .txt que posteriormente vai ser aberto pelo Open Office já para ser editado e formatado.

Você pode melhorar o script fazendo com que o arquivo seja especificado diretamente pela linha de comando, para que o script pegue o mesmo nome do arquivo e salve com a extensão desejada, por exemplo.



Script: abre_arquivo_oo.bsh (melhorado)

Script que filtra um arquivo html, converte para texto e abre-o no OpenOffice

```
#!/bin/bash

#####
#
# Exemplo de script que interage com o Open Office melhorado
#
#####

#(variaveis).....#
```



Script: abre_arquivo_oo.bsh (melhorado)

Script que filtra um arquivo html, converte para texto e abre-o no OpenOffice

```
ARQUIVO=$1
LIMPO=$(echo $1 | sed 's/\.*//')

sed 's/<[^>]*>///g' $ARQUIVO > $LIMPO.txt; openoffice -oowriter $LIMPO.txt &

#(fim).....#
```

Você ainda pode fazer testes para que o programa não aceite parâmetros vazios em \$1, passar mensagens de erro se o arquivo não for encontrado, criar funções para que o script abra mais de um tipo de arquivo no Open Office...e por ai vai...aos poucos seu script pode se tornar um pequeno monstinho de vários tentáculos.



Capítulo 3

Scripts com ferramentas administrativas:

- Servidor Apache;
 - Um pouco sobre o Servidor Web Apache;
 - Onde conseguir;
 - Meu Apache está rodando?
 - Script que testa
 - Logs de acesso;
 - Script que processa logs de acesso;
- Banco de Dados MySQL;
 - dicas de criação de scripts;
 - expressões regulares em MySQL;
- Banco de Dados Oracle;
 - dicas de criação de scripts;
 - exemplos prontos;



Servidor

Apache

Um pouco sobre o Servidor Web Apache

O Servidor Apache surgiu de um projeto desenvolvido pela National Center for Supercomputing Applications que fica na universidade de Illinois, desenvolvido por Rob McCool nos anos 90. É uma opção OpenSource de aplicação para gerenciamento de páginas Web, e se tornou tão popular que hoje é usado por mais de 80% dos servidores Web ao redor do mundo. Pode-se dizer que o Apache foi um dos softwares que contribuíram para a popularização do Linux.

Onde conseguir

Para conseguir a ultima versão estável do Apache, basta baixar diretamente da página oficial do projeto, escolhendo um dos mirrors:

<http://www.apache.org/dyn/closer.cgi>

Ou para usuários do Debian ou Ubuntu, basta digitar na linha de comando, como root:


```
prompt> apt-get install apache
```

Ou é bem provável que, dependendo da distribuição que você estiver usando, o Apache já seja instalado junto com o sistema, ou pelo menos exista no cd de instalação.


Meu Apache está rodando?

Para verificar a execução do Apache, podemos verificar pelo terminal dando um simples comando "ps -ef | grep apache".

Mas como a finalidade desta apostila é a criação de scripts que facilitem a vida do usuário/administrador/analista/(coloque aqui seu cargo), vamos ao primeiro exemplo básico de script para se utilizar com Apache.

	<p>Script: <code>checa_apache.bsh</code></p> <p>Script que testa se o Apache esta rodando, caso negativo ele executa o comando para "startar" o Apache.</p>
<pre>#!/bin/bash ####[Nome:]##### # # Script checa_apache.bsh # ####[Descricao:]##### # # Script que testa se o Apache esta rodando, caso # negativo ele executa o comando para "startar" o # Apache. ##### RUN=\$(ps -ef grep apache grep -v grep awk '{print \$8}' head -1 cut -d/ -f4) if ["\$RUN"]; then echo "Apache esta Rodando" else apachectl start fi # [\$run] && echo "Apache is running" apachectl start</pre>	

Este script executa o teste do Apache, caso ele não esteja ativo, o próprio script se encarrega de "subir" o Apache. Percebe que, com algumas modificações, essa simples solução pode muito bem ser inserida no cron do servidor onde vai ser executado o Apache e ele vai fazer a verificação diária do seu servidor Web, caso o Apache de algum problema, ele não ficará mais de 24 horas fora do ar. Claro que certos casos se tornam mais críticos, mas já temos um começo.

	<p>AVISO: Somente o root pode executar os comandos do Apache, para que mais usuários possam administrar o servidor, recomenda-se estudar sobre o comando sudo, que foge ao escopo desta apostila.</p>
---	---

Logs de acesso

Depois que um servidor Apache é devidamente configurado podemos dizer que ele "está no ar", então ele inicia uma de suas funções que é a de armazenar informações sobre todo os computadores que acessaram a(s) página(s) nele contida(s), e essas informações são armazenadas em um arquivo de log, o arquivo se chama "access.log" e, a não ser que seja definido um local de armazenamento destes arquivos, ele pode ser encontrado no diretório "/var/log/apache/", o formato de armazenamento das informações de acesso ao servidor são feitos diariamente, das 00:00:00 as 23:59:59. Vejamos a seguir um exemplo de formatação do arquivo access.log:

```
192.168.0.0 - - [09/Jun/2006:11:29:44 -0300] "GET /mlmmj-php-web-admin/ HTTP/1.1" 500 641
"http://www.site.com.br/" "Mozilla/5.0 (X11; U; Linux i686; pt-BR; rv:1.8.0.4)
Gecko/20060508 Firefox/1.5.0.4"
```

Onde:

192.168.0.0 - É o endereço de IP do equipamento que acessou o Servidor;

09/Jun/2006:11:29:44 - Dia e Hora do acesso;

"<http://www.site.com.br/>" - Endereço que foi acessado no Servidor;
 "Mozilla/5.0 (X11; U; Linux i686; pt-BR; rv:1.8.0.4) Gecko/20060508 Firefox/1.5.0.4" - São informações sobre o tipo de Navegador, e o Sistema Operacional do equipamento que acessou o Servidor;



NOTA: Tanto os arquivos de logs quanto o diretório padrão onde são armazenados os arquivos podem ser alterados para serem armazenados em um filesystem distinto do que foi criado como default pelo Apache, por exemplo.

Basta para isso editar o arquivo /usr/local/apache/conf/httpd.conf e os módulos do Apache.

Conhecendo essas informações podemos traçar algumas estatísticas de acesso para obter um controle melhor do Apache, podemos criar, por exemplo, um script que verifique os arquivos de log e exiba a quantidade de acessos que ocorreram por hora naquele Servidor. Vejam um exemplo:

```
#!/bin/bash

#....[ Ficha: ].....#
#
# Script: Conta acessos
# Escrito por: Reinaldo Marques de Lima (Plastico)
# Criado em: 05/2006
# Ultima atualizacao: 23/06/2006
#
#....[ Descricao: ].....#
#
#      Script que conta a quantidade de acessos de um Servidor Apache
#
#....[ Notas: ].....#
#
# - Este script foi criado para filtrar os logs de acessos a partir da
# formatacao padrao do arquivo de log, que segue o padrao:
# "%h %l %u %t \"%r\" %>s %b"
#
# +----Legenda-----+
# | %h - host      | Endereco da Maquina que acessou o servidor; |
# | %l - ident     | Nome do solicitante;                         |
# | %u - authuser  | Nome do usuario remoto;                     |
# | %t - date      | Tempo no formato especifico;                 |
# | %r - request   | Primeira linha da solicitacao;               |
# | %s - status    | Status da ultima solicitacao;                |
# | %b - bytes     | Numero de bytes enviados como resposta;     |
# +-----+
#
# Para fomatacao diferente, analise uma linha do log, ou entao o arquivo
# httpd.conf e altere aqui as linhas das variaveis DATA1, DATA2 e ACESSOS
# se necessario.
#
# - O filtro foi criado para contabilizar somente as requisicoes de acesso
# bem sucedidas ( codigos 20x e 30x ), caso queira contabilizar todos os
# acessos, inclusive erros basta remover o comando $GREP '\" [23]0[0-9]'
# da linha da variavel ACESSOS.
#
#....[ Variaveis: ].....#
#
#
TOTAL=0
LS=$(which ls)
WC=$(which wc)
AWK=$(which awk)
CUT=$(which cut)
SED=$(which sed)
GZIP=$(which gzip)

# Inicio da contagem total de acessos;
# Testes de portabilidade para que nao
# ocorra nenhum problema na hora de
# executar os comandos sejam qual for
# o sistema operacional unix-like que
# se esteja tentando executar este
# script.
```

```

GREP=$(which grep)          # Recomenda-se testar estas linhas      #
HEAD=$(which head)          # diretamente no terminal para efeito #
TAIL=$(which tail)          # de controle.                #
                             #                                     #
                             #                                     #
#....[ Local do arquivo: ].....#
#
#
if [ $1 ]; then
ARQUIVO=$1
else
ARQUIVO=$(ls -lrt /var/log/apache2/access.log*.gz | $TAIL -1 | $AWK '{print $9}')
fi
#
#
#....[ Recupera datas: ].....#
#
#
DATA2=$(($GZIP -dc $ARQUIVO | $AWK '{print $4}' | $CUT -d: -f1 | $SED 's/\([^\(.*\))/\1/' |
$HEAD -1)
DATA1=$(($GZIP -dc $ARQUIVO | $AWK '{print $4}' | $CUT -d: -f1 | $SED 's|
\[^\(.*\)/\([^\(.*\))/\([^\(.*\)]\1\2\3|' | $HEAD -1)
#
#
#....[ Looping que conta acessos por hora: ].....#
#
#
echo "   Acessos para o dia $DATA2" > acessos_$DATA1.txt
echo "+-----+" >> acessos_$DATA1.txt

for i in 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23; do

    ACESSOS=$(($GZIP -dc $ARQUIVO | $GREP "\" [23]0[0-9]" | $AWK '{print $4}' | $CUT -d:
-f2 | $GREP $i | $WC -l)
    echo "   >>> Hora: $i - $ACESSOS" >> acessos_$DATA1.txt
    TOTAL=$((TOTAL + $ACESSOS))

done

echo "+-----+" >> acessos_$DATA1.txt
echo "   Total de Acessos = $TOTAL" >> acessos_$DATA1.txt
#
#
#....[ Fim ].....#

```

Para quem está começando ou tem interesse em atuar na administração de servidores Web Apache, creio que estes dois exemplos de scripts ajudam bastante, didaticamente falando. Claro que a criação de scripts para administrar está ferramenta depende muito de que tipo de informação se quer obter.

No caso acima, como os exemplos são para contagem de horas de acesso, pode-se criar um filtro para que seja contabilizado somente os acessos válidos (códigos 200 e 300), aqui temos uma tabela simples com a lista dos códigos.

200	solicitação bem sucedida
302	redirecionamento ocorrido
401	autorização negada
403	proibido
404	não encontrado
5xx	erros do servidor

Existe também a possibilidade de formatar o log para facilitar sua leitura. No arquivo

httpd.conf encontre as linhas com a ocorrência "LogFormat" e veja qual é o padrão de formato dos logs. Onde:

```
%h = host
%l = ident
%u = authuser
%t = date
%r = request
%s = status
%b = bytes
```



Aqui abordarei de forma simples e direta a maneira mais simples de se criar scripts em shell que executem determinadas tarefas diretamente no banco de dados, não entrarei a fundo nos conceitos de bancos de dados MySQL, pois existem publicações de ótimo conteúdo para esse assunto, além de fugir ao escopo desta apostila.

Dicas de criação de scripts:

Partiremos aqui do principio de que já tenhamos o MySQL instalado em com algumas tabelas e usuários criados já com as devidas permissões, e que o banco está rodando normalmente.

A manipulação do banco através de linha de comando é bastante rápida e amigável, mas dependendo do comando a ser executado, ele torna-se um tanto extenso e cansativo para o usuário que executa tarefas diariamente no banco de dados.

Por sorte os comandos do banco de dados MySQL usam um padrão bastante fácil de executar, o que facilita na criação de script para automatizar estas tarefas, e é aí que entra a parte divertida da historia.

Veja um exemplo de comando do MySQL:

```
prompt> mysql -u usuariol -psenha -e "COMANDOS" basel
```

O comando acima é bem genérico e segue o padrão de execução em linha de comando do banco MySQL, onde "mysql" foi a chamada do banco, a opção "-u" define o usuário "usuariol", a opção "-p" especifica a senha "senha" (exatamente como mostrada acima, sem espaço), a opção "-e" indica que a seguir serão executados os comandos "COMANDOS" na base de dados "basel".

Logo, seguindo este padrão, fica fácil pensar em um script que automatize esta tarefa, não é?

"Sim, mas antes uma coisa, se qualquer usuário der um "ps" para ver os processos, irá ver a senha que eu digitei na linha de comando..."

Lá isso é verdade, para resolver isto basta editar o arquivo /etc/mysql/my.cnf removendo o comentário da linha:

```
[client]
#password            = my_password
```

E definindo uma senha padrão para todos os usuários que administram o banco, ou, para gerar uma senha para cada usuário, em sua home crie um arquivo chamado .my.cnf com as unicas duas linhas:

```
[client]
password=senha
```

Dessa forma não será mais necessário inserir a senha pela linha de comando. Vamos agora a um exemplo bastante simples:



Script: insere_dados.bsh
Insere dados na tabela "tabela"

```
#!/bin/bash  
IFS=:  
while read NOME TEL EMAIL;do  
  
mysql -u usuario -e "INSERT INTO tabela VALUES('$NOME','$TEL','$EMAIL')" basel  
  
done < $1
```

Este script simples executa uma inserção na tabela "tabela" da base de dados "basel", onde foram inseridos os dados "NOME", "TEL" e "EMAIL", lidos a partir de um arquivo externo, que teve como separados de campos o caractere ":" especificado na variável "IFS".

Aqui o "IFS" foi usado porque, para facilitar a inserção, recomenda-se que os dados seja escritos da seguinte forma:

```
prompt> cat dados.txt  
Reinaldo:555-2001:reinaldo@meuemail.com  
Plastico::plastico@emaildoplastico.com  
Julia::julia@emaildajulia.com  
Jana:555-4004:jana@emaildajana.com
```

Note que nem todos os registros que irão ser cadastrados tem número de telefone, o script com auxilio da variável "IFS" vai interpretar automaticamente os campos.

"Legal, mas como sabemos se a tarefa foi bem sucedida?"

Simples, como fazemos com a verificação de qualquer tarefa em shell, damos um echo na variável "\$?", se ela retornar 0 (zero) foi bem sucedida, se retornar outro valor significa que ocorreu algum erro durante a execução.

"E como eu sei quais os dados que foram inseridos?"

Mais simples ainda, antes de inserir basta pedir para que os dados sejam exibidos na tela:

```
echo "Nome:$NOME Telefone:$TEL email:$EMAIL"
```


Neste caso o script vai ler linha por linha do arquivo e executar a tarefa solicitada, mas imagine um arquivo de dez mil linhas por exemplo, esta tarefa vai levar um tempo muito grande de execução, pois vai chamar o banco dez mil vezes e inserir os dados nesta mesma quantidade, para agilizar este processo, podemos criar um arquivo do próprio banco ".SQL" e deixar o próprio banco se virar.

Vejamos um exemplo:



Script: insere_dados.bsh
Insere dados na tabela "tabela" melhorado.

```
#!/bin/bash  
IFS=:  
while read NOME TEL EMAIL;do  
  
mysql -u usuario -e "INSERT INTO tabela VALUES('$NOME','$TEL','$EMAIL');" >> dados.SQL  
  
done < $1
```

	Script: insere_dados.bsh Insere dados na tabela "tabela" melhorado.
mysql -u usuario -f base1 < dados.SQL	

A opção "-f" vai assegurar que o script tentou executar todos os comandos, no caso de haver algum dado já existente na base.

Como vimos a criação de scripts para trabalhar em conjunto com o banco de dados MySQL é bastante simples, pois os comandos do banco são bastante flexíveis e de fácil compreensão e não se limitam a estes dois exemplos, mas também a todos os conceitos de bancos de dados, como SELECT, DELETE, etc...

Expressões regulares em MySQL

Você sabia que o banco de dados MySQL da suporte a expressões regulares em suas pesquisas???? Não??? Pois é, e adivinha, nesta apostila eu vou listar todos os meta caracteres que são aceitos e ainda vou passar alguns exemplos de pesquisa. Sim, eu sei...eu sou um amor de pessoa 8^).

São eles:

```
+-----+
|   ^ $ . * + ? | ( ) [ ] [^]   |
+-----+
```

E claro aceitam combinações de meta caracteres, como:

^[]\$, (|) , ([]) ...etc

A sintaxe utilizada diretamente na linha de comando é:

```
mysql> SELECT "o que você está procurando" REGEXP "expressão";
```

ATENÇÃO: Os exemplos abaixo utilizam a numeração "0" para erro e "1" para acerto.

Caracteres/ Construtores especiais:

^ - Combina com o início de uma string.

```
mysql> SELECT "fo\nfo" REGEXP "^fo$";          -> 0
mysql> SELECT "fofo" REGEXP "^fo";             -> 1
```

\$ - Combina com o fim de uma string.

```
mysql> SELECT "fo\no" REGEXP "^fo\no$";        -> 1
mysql> SELECT "fo\no" REGEXP "^fo$";           -> 0
```

. - Combina com qualquer caracter (incluindo novas linhas)

```
mysql> SELECT "fofo" REGEXP "^f.*";            -> 1
mysql> SELECT "fo\nfo" REGEXP "^f.*";         -> 1
```

a* - Combina com qualquer seqüência de zero ou mais caracteres a.

```
mysql> SELECT "Ban" REGEXP "^Ba*n";            -> 1
mysql> SELECT "Baaan" REGEXP "^Ba*n";         -> 1
mysql> SELECT "Bn" REGEXP "^Ba*n";            -> 1
```

a+ - combina com qualquer seqüência de um ou mais caracteres a.

```
mysql> SELECT "Ban" REGEXP "^Ba+n";           -> 1
```

```
mysql> SELECT "Bn" REGEXP "^Ba+n";          -> 0
```

a? - Combina com zero ou um caracter a.

```
mysql> SELECT "Bn" REGEXP "^Ba?n";          -> 1
mysql> SELECT "Ban" REGEXP "^Ba?n";          -> 1
mysql> SELECT "Baan" REGEXP "^Ba?n";         -> 0
```

de|abc - Combina tanto com a sequência de como com abc.

```
mysql> SELECT "pi" REGEXP "pi|apa";          -> 1
mysql> SELECT "axe" REGEXP "pi|apa";          -> 0
mysql> SELECT "apa" REGEXP "pi|apa";          -> 1
mysql> SELECT "apa" REGEXP "^(pi|apa)$";      -> 1
mysql> SELECT "pi" REGEXP "^(pi|apa)$";      -> 1
mysql> SELECT "pix" REGEXP "^(pi|apa)$";      -> 0
```

(abc)* - Combina com zero ou mais instâncias da sequência abc.

```
mysql> SELECT "pi" REGEXP "^(pi)*$";          -> 1
mysql> SELECT "pip" REGEXP "^(pi)*$";         -> 0
mysql> SELECT "pipi" REGEXP "^(pi)*$";        -> 1
```

Existe um modo mais geral de se escrever regexp que combinam com muitas ocorrências de um átomo anterior.

a* - Pode ser escrito como a{0,}.

a+ - Pode ser escrito como a{1,}.

a? - Pode ser escrito como a{0,1}.

Para ser mais preciso, um átomo seguido por um limite contendo um inteiro i e nenhuma vírgula casa com uma sequência de exatamente i combinações do átomo. Um átomo seguido por um limite contendo i e uma virgula casa com uma sequência de i ou mais combinações do átomo. Um átomo seguido por um limite contendo dois inteiros i e j casa com uma sequência de i até j (inclusive) combinações de átomos.

Ambos os argumentos devem estar na faixa de 0 até 255, inclusive. Se houver dois argumentos, o segundo deve ser maior ou igual ao primeiro.

Como funciona a lista:

[a-dX], [^a-dX]

Combina com qualquer caracter que seja (ou não, se ^ é usado) a, b, c, d ou X. Para incluir um caracter literal], ele deve ser imediatamente seguido pelo colchete de abertura [. Para incluir um caracter literal -, ele deve ser escrito primeiro ou por ultimo. Assim o [0-9] encontra qualquer dígito decimal. Qualquer caracter que não tenha um significado definido dentro de um para [] não tem nenhum significado especial e combina apenas com ele mesmo.

```
mysql> SELECT "aXbc" REGEXP "[a-dXYZ]";      -> 1
mysql> SELECT "aXbc" REGEXP "^[a-dXYZ]$";    -> 0
mysql> SELECT "aXbc" REGEXP "[a-dXYZ]+$";    -> 1
mysql> SELECT "aXbc" REGEXP "^[^a-dXYZ]+$";  -> 0
mysql> SELECT "gheis" REGEXP "^[^a-dXYZ]+$"; -> 1
mysql> SELECT "gheisa" REGEXP "^[^a-dXYZ]+$"; -> 0
```

Usando o padrão POSIX:

O banco de dados MySQL é compatível com o padrão POSIX, com algumas particularidades próprias, como veremos a seguir, aqui vai uma listinha da classe POSIX para expressões regulares.

```
+-----+
| Classe:          Similar à:      Significa:          |
```

[[:upper:]]	[A-Z]	letras maiúsculas
[[:lower:]]	[a-z]	letras minúsculas
[[:alpha:]]	[A-Za-z]	maiúsculas/minúsculas
[[:alnum:]]	[A-Za-z0-9]	letras e números
[[:digit:]]	[0-9]	números
[[:xdigit:]]	[0-9A-Fa-f]	números hexadecimais
[[:punct:]]	[.,!?:...]	sinais de pontuação
[[:blank:]]	[\t]	espaço e TAB
[[:space:]]	[\t\n\r\f\v]	caracteres brancos
[[:cntrl:]]	-	caracteres de controle
[[:graph:]]	[^ \t\n\r\f\v]	caracteres imprimíveis
[[:print:]]	[^ \t\n\r\f\v]	imprimíveis e o espaço

E aqui vão algumas expressões de pesquisa que podem ser usadas em conjunto com o POSIX:

```
[[:character:]]
```

A sequência de caracteres daquele elemento ordenado. A sequência é um único elemento da lista de expressões entre colchetes. Um expressão entre colchetes contendo um elemento ordenado multi-caracter pode então combinar com mais de um caracter, por exemplo, se a sequência ordenada inclui um elemento ordenado ch, então a expressão regular `[[:ch:]]*c` casa com os primeiros cinco caracteres de `chchcc`.

```
[[:classe_caracter=]]
```

Uma classe equivalente, procura pela sequência de caracteres de todos elementos ordenados equivalentes àquele, incluindo ele mesmo.

Por exemplo, se o e (+) são os membros de uma classe equivalente, então `[[:o=]]`, `[[:(+)=]]` e `[o(+)]` são todos sinônimos. Uma classe equivalente não pode ser o final de uma escala.

```
[[:character_class:]]
```

Dentro de colchetes, o nome de uma classe de caracter entre `[:` e `:` procura pela lista de todos os caracteres pertencentes a esta classe. Os nomes de classes de caracteres padrões são:

Nome	Nome	Nome
alnum	digit	punct
alpha	graph	space
blank	lower	upper
cntrl	print	xdigit

Ele procura pelas classes de caracteres definidas na página `ctype(3)` do manual. Um local pode fornecer outros. Uma classe de caracter não pode ser usada como o final de uma escala.

```
mysql> SELECT "justalnums" REGEXP "[[:alnum:]]+";      -> 1
mysql> SELECT "!!" REGEXP "[[:alnum:]]+";             -> 0
```

```
[[:<:]], [[:>:]]
```

Combina com a string null no inicio e no fim de uma palavra, respectivamente. Uma palavra é definida como uma sequência de caracteres de palavra os quais não são nem precedido e nem seguidos por caracteres de palavras. Um caracter de palavra é um caracter alfa numérico (como definido por `ctype(3)`) ou um underscore (`_`).

```
mysql> SELECT "a word a" REGEXP "[[:<:]]word[[:>:]]";      -> 1
mysql> SELECT "a xword a" REGEXP "[[:<:]]word[[:>:]]";      -> 0
mysql> SELECT "weeknights" REGEXP "^(wee|week)(knights|nights)$"; -> 1
```

Acesse a página do Henry Spencer e absorva um pouco mais de informação a respeito:

<http://arglist.com/regex/regex7.html> (em inglês)

A maior parte deste tópico de Expressões Regulares em bancos MySQL foi retirada da página do MySQL (<http://dev.mysql.com/doc/refman/4.1/pt/regexp.html>), eu adicionei algumas coisas que achei estarem faltando para um bom entendimento deste conteúdo.



Dicas de criação de scripts:

Exemplos prontos

Eis aqui uma lista de scripts prontos que foram extraídos da página da Oracle. Simples de serem adaptados para o dia-a-dia e de fácil entendimento, analise, entenda e aproveite.

```
#!/bin/bash

output=`sqlplus -s "/ as sysdba" <<EOF
    set heading off feedback off verify off
    select distinct machine from v\\$session;
    exit
EOF
`

echo $output
```

```
#!/bin/bash

cat /etc/oraInst.loc | grep inventory_loc > tmp
cat tmp | awk -F= '{print $2}'
rm tmp
```

```
#!/bin/bash
echo "`ps -ef | grep smon|grep -v grep|awk '{print $8}' | awk -F \"_\" '{print$3}'`"
```

```
#!/bin/bash
dblist=`cat /etc/oratab | awk -F: '{print $2 }'`

for ohome in $dblist ; do
    echo $ohome
done
```

```
#!/bin/bash
#
# For example:
#
# 11i Financials instance
#
# cat $ORACLE_HOME/admin/$CONTEXT_NAME/bdump/alert_$ORACLE_SID.log | grep ORA-
#
# 10g
#
# cat $ORACLE_BASE/admin/$ORACLE_SID/bdump/alert_$ORACLE_SID.log | grep ORA-
#
#
# $APPL_TOP/admin/$TWO_TASK/log

cat $1 | grep ORA- > alert.err

if [ `cat alert.err|wc -l` -gt 0 ]
then
    mail -s "$0 $1 Errors" Casimir.Saternos@buzziunicemusa.com < alert.err
fi

rm alert.err
```

```
#!/bin/bash
echo "Physical RAM must be greater than 512 MB (524288 KB) "
grep MemTotal /proc/meminfo
echo ""

echo "Swap space must be greater than 1 GB (1048576 KB) or twice the size of RAM"
echo "On systems with 2 GB or more of RAM, the swap space can be between one and two
times the size of RAM "
grep SwapTotal /proc/meminfo
echo ""

echo "Disk space in /tmp must be greater than 400 MB (409600 KB) "
df -k /tmp
echo ""

echo "Disk space for software files 2.5 GB (2621440 KB) and"
echo "Disk space for database files 1.2 GB (1258290 KB) "
df -k
echo ""

echo "OS:"
cat /etc/issue
echo ""

echo "Required packages:"
echo "Min: gcc-3.2.3-2"
rpm -q gcc
echo ""
echo "Min: make-3.79"
rpm -q make
echo ""
echo "Min: binutils-2.11"
rpm -q binutils
echo ""
echo "Min: openmotif-2.2.2-16"
rpm -q openmotif
echo ""
echo "Min: setarch-1.3-1"
rpm -q setarch
echo ""
echo "Min: compat-db-4.0.14.5"
```

```

rpm -q compat-db
echo ""
echo "Min: compat-gcc-7.3-2.96.122"
rpm -q compat-gcc
echo ""
echo "Min: compat-gcc-c++-7.3-2.96.122"
rpm -q compat-gcc-c++
echo ""
echo "Min: compat-libstdc++-7.3-2.96.122"
rpm -q compat-libstdc++
echo ""
echo "Min: compat-libstdc++-devel-7.3-2.96.122"
rpm -q compat-libstdc++-devel
echo ""

```

```

#!/bin/bash
echo "+-----"
echo "|/proc/sys/kernel/sem "
echo "|semmsl 250 "
echo "|semmsl 32000 "
echo "|semopm 100 "
echo "|semmni 128 "
/sbin/sysctl -a | grep sem
echo ""
echo "|/proc/sys/kernel/shmall "
echo "|shmall 2097152 "
/sbin/sysctl -a | grep shmall
echo ""

echo "|/proc/sys/kernel/shmmax "
echo "|shmmax Half the size of physical memory "
/sbin/sysctl -a | grep shmmax
echo "|physical memory total:"
grep MemTotal /proc/meminfo
echo ""
echo "|/proc/sys/kernel/shmmni "
echo "|shmmni 4096 "
/sbin/sysctl -a | grep shmmni
echo ""

echo "|/proc/sys/fs/file-max "
echo "|file-max 65536 "
/sbin/sysctl -a | grep file-max
echo ""
echo "|/proc/sys/net/ipv4/ip_local_port_range "
echo "|ip_local_port_range 1024 65000 "
/sbin/sysctl -a | grep ip_local_port_range
echo ""

```

```

#!/bin/bash
rman target / <<EOF
shutdown immediate;
startup mount;
backup spfile;
backup database;
alter database open;
DELETE NOPROMPT OBSOLETE;
quit;
EOF

```

```

#!/bin/bash
if [ -z "$1" -o -z "$2" ]; then

```

```

    echo ""
    echo " ERROR : Invalid number of arguments"
    echo " Usage : run_bk.sh <sid> <backup_type>"
    exit
fi

#
# Set the variables related to the log message to be sent to the DBA
#
MAIL_TO=Casimir.Saternos@buzziunicemusa.com
export MAIL_TO
ORACLE_SID=$1;
export ORACLE_SID
BACKUP_TYPE=$2;
export BACKUP_TYPE
HOST=`hostname`
export HOST
TODAY=$(date)
export TODAY

sh $BACKUP_TYPE.sh | mail -s "$HOST $ORACLE_SID $BACKUP_TYPE Started: $TODAY " $MAIL_TO

```




Capítulo 4

Criação de Programas visuais com:

- gmessage / xmessage
- zenity
 - exemplos das caixas
- kdialog
 - exemplos das caixas



Interface Gráfica

Gnome

Gmessage / Xmessage

Gmessage

O gmessage é uma opção de caixa de dialogo para gnome, não sendo oficial não é encontrado instalado em todas as distribuições, porém é um programa fácil e gostoso de usar muito bom para agrega-lo a scripts que usem interface gráfica.

Para usuários do Gnome que queiram utilizar este recurso, o programa requer no mínimo o GTK+ 1.2 ou superior para rodar.

A página oficial do projeto é

<http://computacion.cs.cinvestav.mx/~lgallardo/gmessage/index.html> (estava com uma mensagem de "Forbidden" da ultima vez que tentei acessa-la).

Para baixar o programa você pode acessar o link do "tarball" <http://computacion.cs.cinvestav.mx/~lgallardo/gmessage/gmessage-1.0.7.tar.gz> e instala-lo manualmente, mas se você for usuário do Debian ou do Ubuntu basta dar um apt-get install gmessage como root para que o sistema já se encarregue de instala-lo para você.

Sua sintaxe de uso é bastante simples:

```
prompt> gmessage [ opção 1 ] [ opção 2 ] [ opção N ] "Texto"
```

E aqui veremos uma lista das principais funções deste programa:

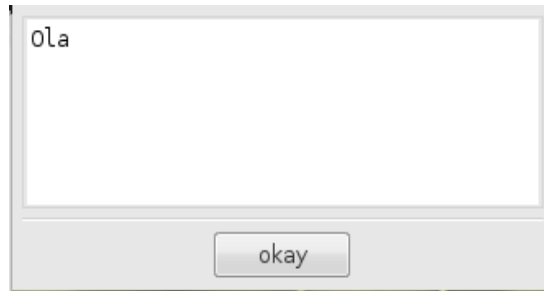
-bg "COR" - esta opção define a cor de fundo que será usada na caixa. (deve-se especificar a cor em inglês off course).

```
prompt> gmessage -bg "yellow" "Ola"
```



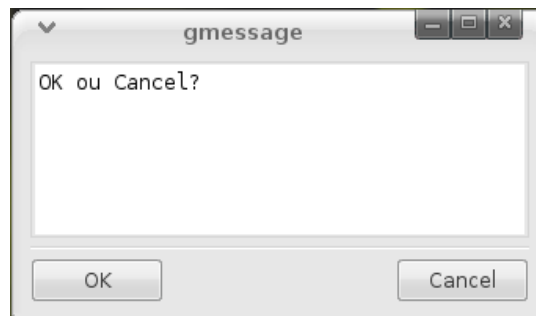
`-borderless` - desabilita a visualização da bordas de janela (Metacity no Gnome) da caixa.

```
prompt> gmessage -borderless
```



`-buttons` - define a legenda do botão, podemos adicionar mais botões passando os nomes separados por vírgula, e podemos ainda definir valores aos botões definindo-os logo após o nome separando por dois pontos ":" o resultado é mostrado na saída padrão.

```
prompt> gmessage -buttons "OK:0,Cancel:1" "OK ou Cancel?"
```



(não esqueça de dar `echo $?` para que o programa mostre o resultado)

`-center` - posiciona a caixa no centro da tela.

`-entry` - habilita a entrada de dados e envia automaticamente para o stdout

```
prompt> gmessage -entry "Digite algo"
```



`-print` - opção que imprime na saída padrão o conteúdo do botão escolhido, por exemplo: `gmessage -print -buttons "OK,Cancel"`; caso escolhido o botão OK a saída padrão mostrará como resultado "OK".

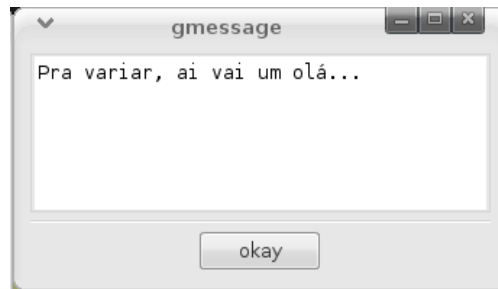
`-fg "COR"` - define a cor da fonte usada na caixa.

```
prompt> gmessage -fg "Blue" "Ola"
```



-file - mostra na caixa o arquivo especificado neste parâmetro.

```
prompt> gmessage -file ola.txt
```



-geometry COMPRIMENTOxALTURA - define o tamanho da caixa a ser mostrada.

-nearmouse - mostra a caixa próxima ao ponteiro do mouse.

-timeout - fecha a caixa no tempo especificado em segundos.

-title - define o título da caixa.

```
prompt> gmessage title Ola "Ola denovo"
```



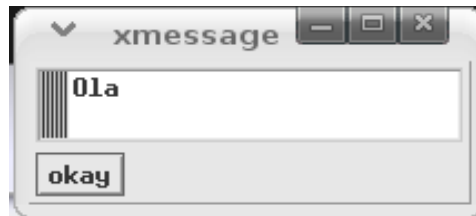
Xmessage (Defalut em qualquer 'Xwindow')

Semelhanças e diferenças:

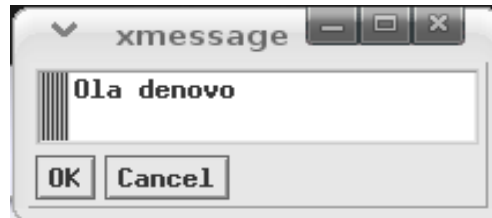
Xmessage é um tipo de caixa de dialogo compatível com qualquer distribuição Linux que tenha o XWINDOW instalado, sua sintaxe também é fácil de trabalhar mas é um tanto pobre se comparada com a gmessage, tornando isso uma desvantagem por falta de recursos, em contrapartida ela pode ser usada fora do Gnome abrangendo um alcance maior de usuários, dando uma vantagem boa sobre o seu primo de pé grande.

Alguns exemplos de caixas xmessage:

```
prompt> xmessage -geometry 200x70 "Ola"
```



```
prompt> message -geometry 200x70 -buttons "OK,Cancel" "Ola denovo"
```



Como vimos nestes dois exemplos, o xmessage precisa a toda hora que sejam especificados parâmetros de tamanho, caso contrario as caixas aparecem desta maneira:

```
ptompt> xmessage "Ola"
```



Ou seja, do tamanho do texto especificado, de certa forma um ponto negativo caso se pense em padronização das caixas.

Zenity

O zenity é atualmente o "Dialog" oficial do gnome, forçando a aposentadoria do gDialog, este novo conceito de caixa de dialogos do gnome vem com a promessa de ser mais leve e fácil de usar e ainda mais funcional. Não podemos duvidar disto levando-se em conta que temos a frente do projeto nomes como Alan Cox e Glynn Foster que são seus idealizadores.

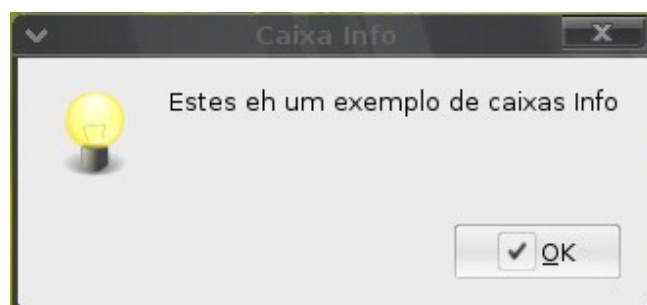
A versão utilizada para confecção deste documento foi: zenity 2.10.0

Exemplos das Caixas

Info

A caixa de dialogo Info como o próprio nome já diz, serve para passar alguma informação ao usuário. veja no exemplo.

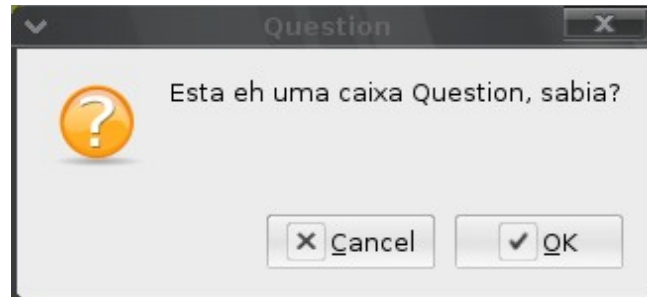
```
prompt> zenity --title="Caixa Info" --info --text="Estes eh um exemplo de caixas Info"
```



Question

A caixa Question tem a mesma função da caixa yesno do dialog, no qual o usuário responde a uma pergunta e dependendo da resposta retorna-se 0 (zero) para SIM (OK) e 1 (um) para NÃO (Cancel) na saída padrão, podendo a resposta ser redirecionada normalmente como em qualquer script.

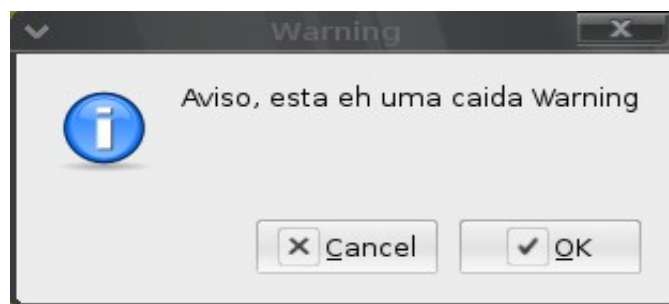
```
prompt> zenity --title="Question" --question --text="Esta eh uma caixa Question, sabia?"
```



Warning

Esta é uma caixa de aviso do sistema comunicando um determinado aviso ao usuário.

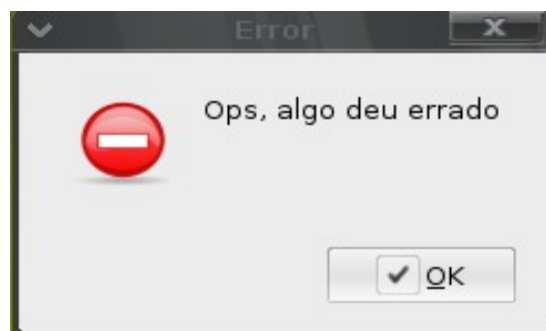
```
prompt> zenity --title="Warning" --warning --text="Aviso, esta eh uma caixa Warning"
```



Error

Caixa utilizada pelo sistema para comunicar ao usuário quando alguma coisa não deu certo. Bastante conhecida em "outros sistemas operacionais" :P.

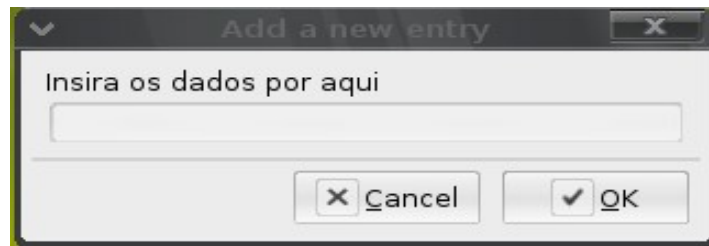
```
prompt> zenity --title="Error" --error --text="Ops, algo deu errado"
```



Entry

Esta caixa é usada para fazer interface com o usuário onde serão inseridos certos dados que um script pode aproveitar alimentando uma variável ou um array por exemplo.

```
prompt> zenity --title="Entry" --entry --text="insira os dados por aqui"
```



File Info

Uma caixa bem versátil, que mostra o conteúdo de um arquivo especificado e passando-se a opção `--editable`, por exemplo, você pode editar o arquivo diretamente desta caixa.

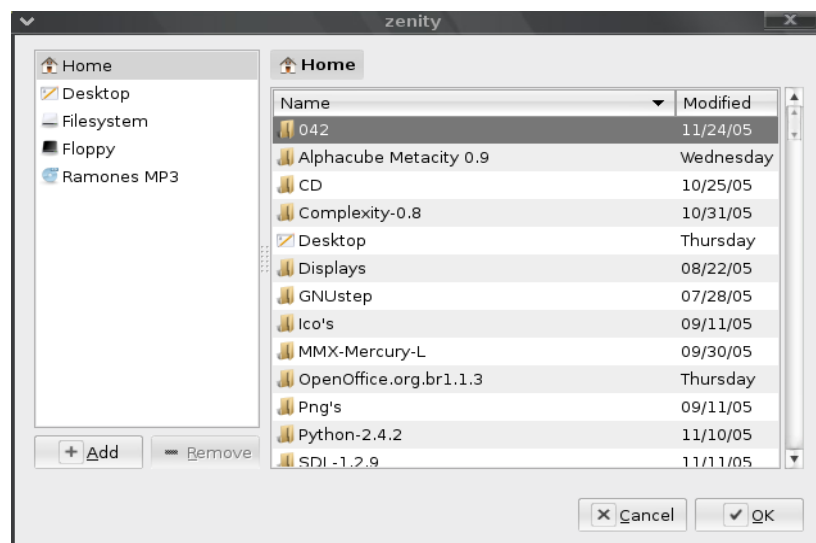
```
prompt> zenity --title="File Info" --text-info --filename ola.txt
```



File Selection

Esta é uma caixa bem popular, que mostra a lista de arquivos contidos em sua máquina para serem selecionados para abrir em algum aplicativo ou fazer upload por exemplo.

```
prompt> zenity --file-selection
```

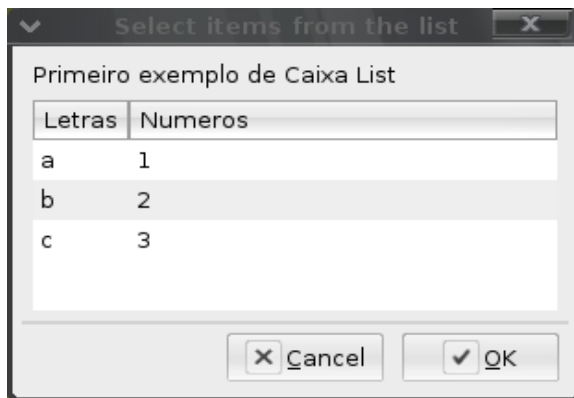


List

A caixa 'list' tem várias vertentes, talvez sendo uma das mais trabalhadas no zenity, ela pode mostrar um conteúdo definido ou uma relação de arquivos texto por exemplo.

Estas caixas tem uma particularidade que se deve dar bastante atenção para que os dados sejam inseridos corretamente nas caixas. A entrada de dados precisa ficar de duas formas.

```
prompt> zenity --text="Primeiro exemplo de Caixa List" --list --column "Letras" a 1 b 2 c 3 --column "Numeros"
```

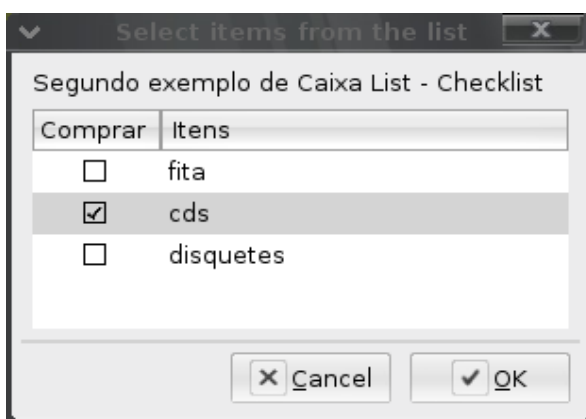


Neste exemplo os dados aparecem em duas colunas (Letras e Números), que foram declaradas no começo e no final como colunas e os dados ficaram no meio intercalados entre Letras (a b c) e Números (1 2 3). A ordem dos dados precisa ser mantida nesse padrão para que os dados sejam inseridos corretamente tendo-se três ou mais colunas, já a declaração das colunas também pode ser feita como no exemplo a seguir.

Checklist

Aqui a caixa checklist é usada para se selecionar uma ou mais opções para que sejam alimentadas num array por exemplo. Pode-se definir alguns campo pré selecionados trocando FALSE por TRUE.

```
prompt> zenity --text="Segundo exemplo de Caixa List - Checklist" --list --checklist --column "Comprar" --column "Itens" FALSE fita FALSE cds FALSE disquetes
```



Repare que a declaração de todas as colunas foram feitas primeiro e depois vem a declaração dos dados sempre intercalados.

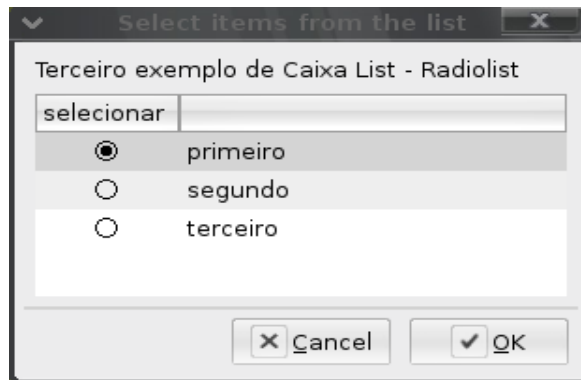
A definição é sempre a mesma, se você declara duas colunas o programa pega os dados em pares para inserir na caixa, para três colunas um trio e assim por diante. Por exemplo, se declararmos três colunas (Letra, Numero e Posição) os dados precisam ser

inseridos em trios também, exemplo: a 1 primeiro.

Radiolist

A caixa radiolist a exemplo da checklist também dá a opção de selecionar dados na caixa, mas somente um dos dados.

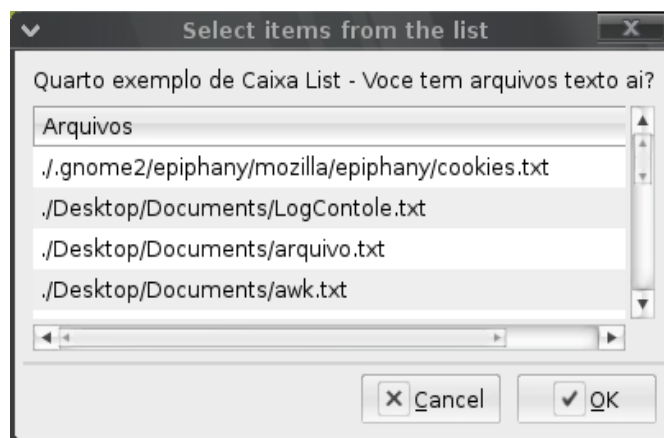
```
prompt> zenity --text="Terceiro exemplo de Caixa List - Radiolist" --list --radiolist --column selecionar FALSE primeiro FALSE segundo FALSE terceiro --column opcao
```



List (Arquivos)

E este é um exemplo de caixa list que mostra o resultado da pesquisa de um determinado tipo de arquivos na máquina.

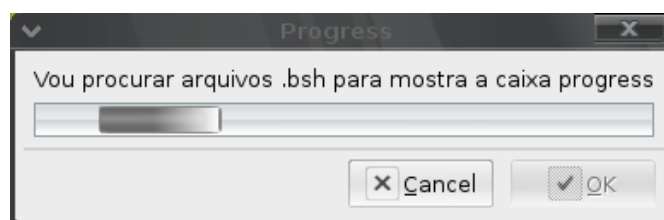
```
prompt> find . -name '*.txt' /(troque aqui por pipe) zenity --text="Quarto exemplo de Caixa List - Voce tem arquivos texto ai?" --list --column "Arquivos"
```



Progress

Outra caixa clássica, que mostra ao usuário o progresso gradativo de um arquivo que está sendo baixado, ou a instalação de um software por exemplo. Esta talvez seja uma das caixas mais difíceis de se implementar.

```
prompt> find `echo $HOME` '*.bsh' /(troque aqui por pipe) zenity --text="Vou procurar arquivos .bsh para mostra a caixa progress" --progress --pulsate
```

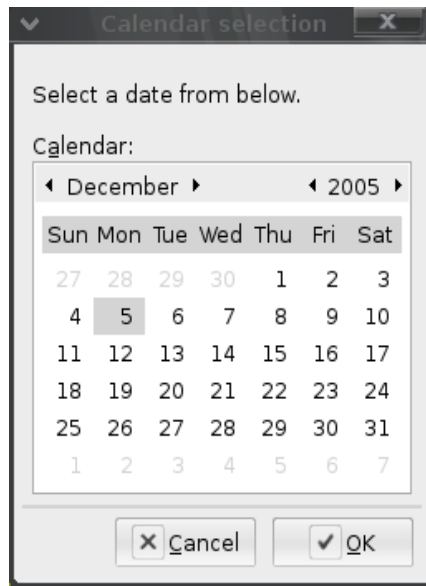


Neste exemplo mostramos a caixa com a opção `--pulsate` que mostra uma barra que fica indo e voltando enquanto não finaliza o processo.

Calendar

Assim como no dialog, também aqui temos uma opção de caixa que mostra um calendário, caso não seja passado nenhum parâmetro na chamada desta caixa, ela tem por default mostrar o dia e mês correntes.

```
prompt> zenity --calendar
```



Vamos ver agora dois exemplos de scripts utilizando caixas de dialogo Zenity.

Script que gera um arquivo de cadastros:

```
#!/bin/bash

#####
#
# Exemplo de Cadastro com zenity                                     #
# Escrito por Reinaldo Marques de Lima ( Plastico )                 #
# Criado em 07/12/2005                                              #
# Ultima Atualização 06/12/2005                                     #
#                                                                    #
#####

#(variaveis)-----#

TITLE="Cadastro de Pessoas"

#(nome)-----#

NOME=$(zenity --title="$TITLE - Nome"                                \
            --text="Digite seu nome: "                             \
            --entry )

[ $? -ne 0 ] && --zenity --text="Esc ou CANCELAR apertado" --error && exit

#(idade)-----#
```

```

IDADE=$(zenity --title="$TITLE - Idade" \
--text="Digite sua idade: " \
--entry )

[ $? -ne 0 ] && --zenity --text="Esc ou CANCELAR apertado" --error && exit

#(sexo)-----#

SEXO=$(zenity --title="$TITLE - Sexo" \
--text="Escolha o sexo: " \
--list \
--radiolist \
--column escolha FALSE masculino FALSE feminino \
--column )

[ $? -ne 0 ] && --zenity --text="Esc ou CANCELAR apertado" --error && exit

#(estado civil)-----#

CIVIL=$(zenity --title="$TITLE - Etado Civil" \
--text="Escolha o estado Civil: " \
--list \
--radiolist \
--column escolha FALSE solteiro FALSE casado FALSE separado FALSE viuvo \
--column )

[ $? -ne 0 ] && --zenity --text="Esc ou CANCELAR apertado" --error && exit

#(telefone)-----#

TELEFONE=$(zenity --title="$TITLE - Telefone" \
--text="Digite o telefone residencial: " \
--entry )

[ $? -ne 0 ] && --zenity --text="Esc ou CANCELAR apertado" --error && exit

#(escrevendo num arquivo texto)-----#

echo -e \
"Nome:      $NOME\nIdade:      $IDADE\nSexo:      $SEXO\nEstado      Civil:      $CIVIL\nTelefone: \
$TELEFONE\n\n-----\n\n" \

>> cadastro.txt

#(mensagem de acerto)-----#

zenity --title="TITLE - Obrigado" \
--text="Dados cadastrados com sucesso" \
--info

```

Script que executa programas através da opção escolhida.

```

#! /bin/bash

#####
#
# script que executa programas pelo zenity
# Escrito por: Reinaldo Marques de Lima ( Plastico )
# criado em: 07/12/2005
# ultima atualização: 07/12/2005
#

```

```
#
#####

OPCAO=$(zenity --title "Qual aplicativo voce quer rodar"
--list
--checklist
--column "Escolha" FALSE firefox FALSE amsn FALSE xmms
--column "Programas" )

$OPCAO & > /dev/null
```



Kdialog

A galera do KDE também conta com um programa que cria caixas de dialogo para sua interface, denominada Kdialog. Peço desculpas ao pessoal do KDE pela falta de informação e a criação de scripts, mas sabe como é não é...eu uso Gnome.

Pesquisando a respeito do Kdialog na internet achei um tópico relacionado falando sobre um tal de kommander, programa que personaliza as caixas de dialogo do KDE já interagindo com scripts...analizando atentamente achei muito parecido com o (ECA :P) Visual Studio. Para quem se interessar o link está aqui:

<http://kde-apps.org/content/show.php?content=12865>

E a página usada como base para elaboração desta parte da apostila foi:

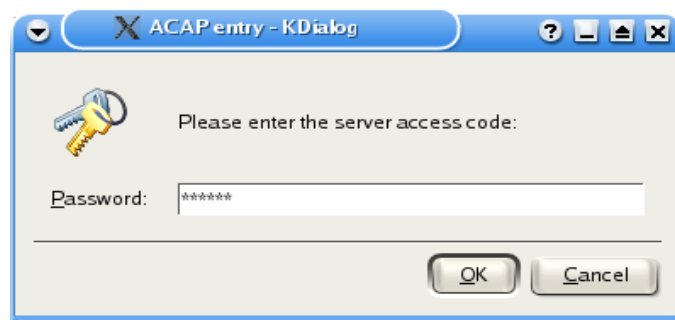
<http://developer.kde.org/documentation/tutorials/kdialog/x85.html>

Neste site pode-se encontrar alguns exemplos de scripts que interagem com o kdialog, recomendo acessa-lo para um melhor aproveitamento desta parte da apostila.

Vamos dar uma olhada nas principais funções das caixas de dialogo do KDE. Algumas caixas que existem neste pacote ainda se encontram em falta no Gnome, o que constitui uma vantagem para o KDE se relacionado com seu primo do pé grande, mas particularmente eu acho que existem caixas que nem precisariam existir. Mas sendo imparcial, fica a critério de cada um. Vamos a elas.

Password

```
prompt> kdialog --title "ACAP entry" --password "Please enter the server access code:"
```



Msgbox

```
prompt> kdialog --msgbox "Password correct.\n About to connect to server"
```



Sorry

```
prompt> kdialog --sorry "Password incorrect.\n Will not connect to server"
```



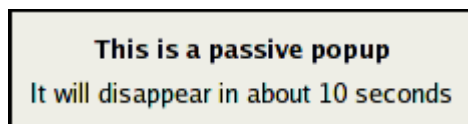
Error

```
prompt> kdialog --error "Server protocol error."
```



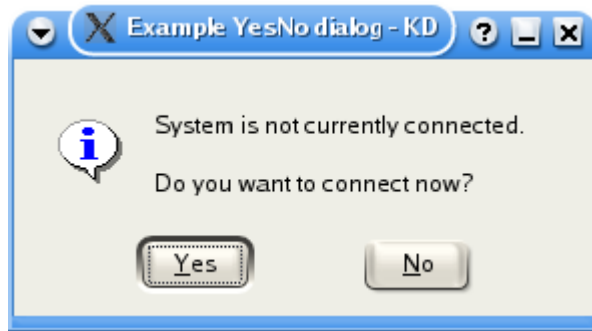
Passive view

```
prompt> ./kdialog --title "This is a passive popup" --passivepopup \
"It will disappear in about 10 seconds" 10
```



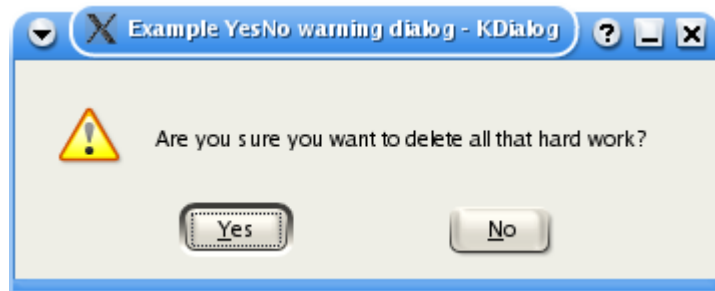
Yesno

```
prompt> kdialog --title "Example YesNo dialog" --yesno "System is not \
currently connected.\n Do you want to connect now?"
```



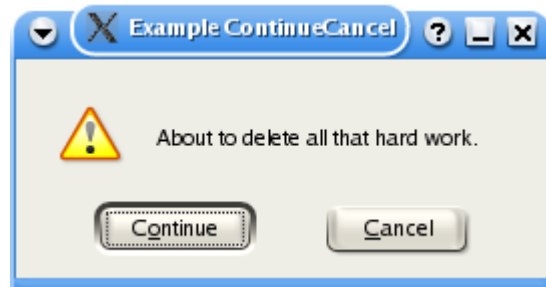
Yesno Warning

```
prompt> kdialog --title "Example YesNo warning dialog" --warningyesno "Are \
you sure you want to delete all that hard work?"
```



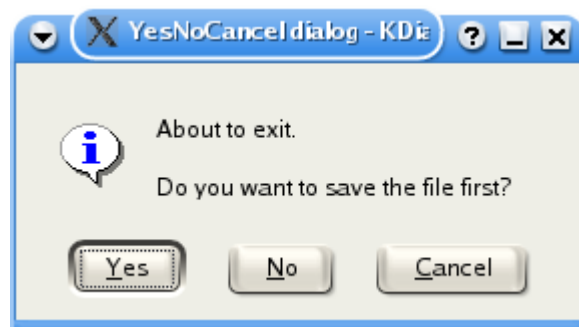
Warning Continue Cancel

```
prompt> kdialog --title "Example ContinueCancel warning dialog" \
--warningcontinuecancel "Are you sure you want to delete all that \
hard work?"
```

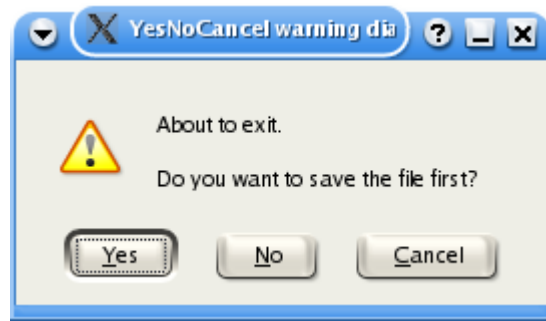


Yesno Cancel

```
prompt> kdialog --title "YesNoCancel dialog" --yesnocancel "About to exit.\n \
Do you want to save the file first?"
```



```
prompt> kdialog --title "YesNoCancel warning dialog" --warningyesnocancel \
"About to exit.\n Do you want to save the file first?"
```



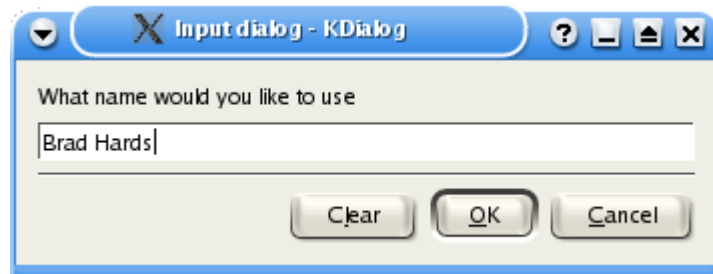
Don't Again

```
prompt> kdialog --dontagain myscript:nofilemsg --msgbox "File not found."
```

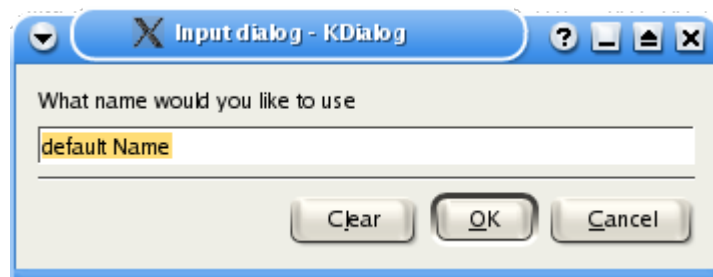


Inputbox

```
prompt> kdialog --title "Input dialog" --inputbox "What name would you like to use"
```



```
prompt> kdialog --title "Input dialog" --inputbox "What name would you like to use" "default Name"
```



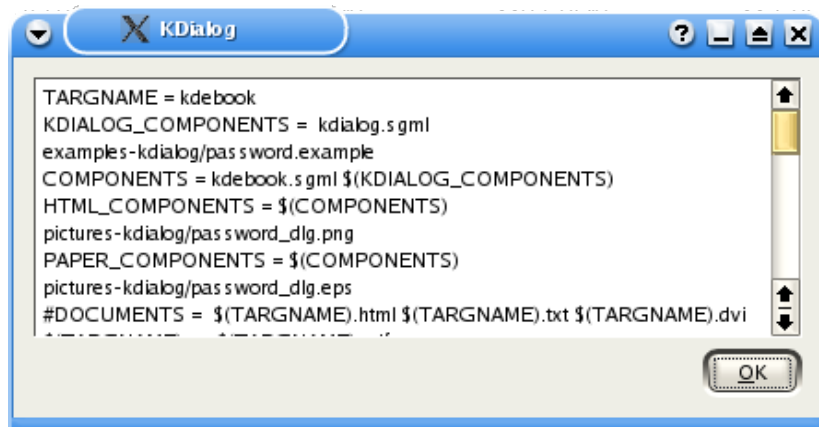
Aqui podemos definir um nome para aparecer na caixa como padrão.

Textbox

```
prompt> kdialog --textbox Makefile
```



```
prompt> kdialog --textbox Makefile 440 200
```



Neste exemplo passamos parâmetros para redimensionar a caixa.
Menu

```
prompt> kdialog --menu "Select a language:" a "American English" b French d "Oz' English"
```



Checklist

```
prompt> kdialog --checklist "Select languages:" 1 "American English" off \
2 French on 3 "Oz' English" off
```



Se alguém achar a diferença entre os dois exemplos acima, por favor me avise.

Combobox

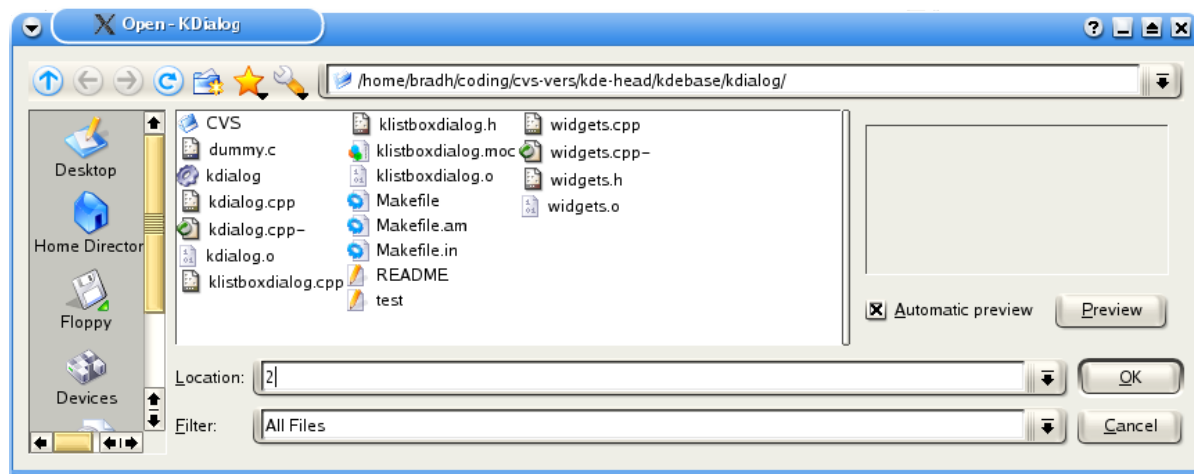
```
prompt> kdialog --combobox "Select a flavour:" "Vanilla" "Chocolate" "Strawberry" "Fudge"
```



Chocolate # olha, ela responde sozinha sem precisar do "echo \$?"...

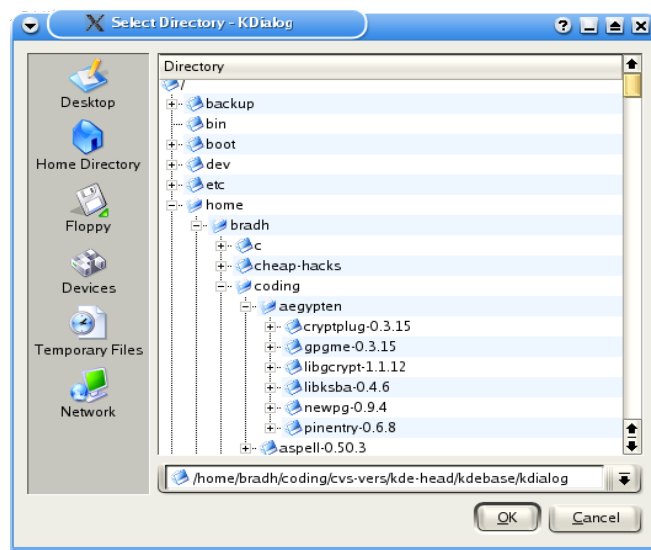
Get open filename

```
prompt> kdialog --getopenfilename
```



Get existing directory

```
prompt> kdialog --getexistingdirectory
```





Apêndice

- Alguns scripts criados pelo autor (EU).

Aqui vai uma lista de alguns scripts que eu criei para, no mínimo, servir de referencia.

Solicite o "tarball" de todos eles em: reimlima@hotmail.com

- plaspkg - Meu primeiro pacote de "programas" em shell.
- plaspkg_dialog - Mesmo pacote, mas todo em dialog.
- plaspkg_zenity - Mais um, agora todo em zenity.
- plasconvert - converte texto para html e vice-versa (em crescimento).
- plasinfo - busca informação em alguns sites da internet.
- sysinfo - script simples que passa informação do sistema.
- sysinfo_gmessage - mesmo programa escrito em gmessage.
- go - faz conexão remota com servidores via ssh.
- scripts_index - cria uma pagina html que mostra conteudo do diretorio "scripts".
- gowalk - script que da um alerta visual quando se passam 50 minutos, para evitar de ficar muito tempo sentado. (script politicamente correto)
- gowalk_xmessage - mesmo programa para outras "distros".
- meuip - script que mostra o ip e a subnet mask.
- meuip_gmessage - mesmo script em gmessage.

Plaspkg

Pacote de "mini-programas" que manipulam uma lista de execução de programas. Você tem alguns programas que são executados diariamente em sua máquina (amsn, firefox...etc), este pacote pode automatizar a tarefa de iniciar estes programas. Ele cria uma lista de execução, onde você pode inserir, remover nomes de programas, matar processos.

```
#!/bin/bash
```

```
#####  
#  
# plaspkg wrote by Reinaldo Marques de Lima (Plastico) #  
# criação iniciada em: 09/2005 #  
# concluido em: 10/2005 #  
# ultima atualização: 25/11/2005 #  
# descrição: #  
# pacote de scripts para automatizar a execução de programas #  
# roda aplicativos de uma lista, inclui e exclui aplicativos #  
# da lista, mostra o conteudo da lista e mata processos que #  
# estejam travados. #  
# #  
# abaixo, segue a documentação de cada um individualmente #  
# e de acordo com cada parte dos programas existem alguns #  
# comentarios sobre o que cada rotina faz. #  
# #  
# Atualizações: #  
# 13/10/2005 - padronização das variaveis e acertos nas #  
# rotinas. #  
# #  
# 17/10/2005 - adicionado o codigo da opção plasrm, e #  
# alterados, a opção 6 de saída e a apresentação. #  
# - inserida uma mensagem de erro caso o usuario digite #
```

```

# qualquer coisa na linha de comando junto com a chamada #
# do programa. #
# - melhorada a tratativa de erro do menu se o usuario #
# digita algo diferente de 1 até 6. #
# (comentario) o antigo codigo era: #
# "if [ $OPCAO -lt 1 ] || [ $OPCAO -gt 6 ]; then" essa #
# opção ficou bastante vaga pois o usuario poderia usar #
# algum caracter invalido que não estivesse na lista do #
# menu, isso causava um erro do proprio sistema, ao inves #
# disso, resolvi tratar esse erro para que o programa #
# fosse "independente" do sistema para identificar e #
# reportar um erro se for digitado um parametro invalido #
# usando a opção de comando ^1-6. #
# #
# 20/10/2005 - adicionada uma nova opção no menu do programa #
# a setima opção é 'plassee' que mostra o conteudo da lista #
# para uma simples conferencia, bem basico, mas achei que #
# seria funcional. #
# - melhorada a apresentação da Ajuda do programa por estar #
# meio 'vaga'. #
# 25/10/2005 - inserido finalmente o codigo da função plasall#
# #
#####
#
# Documentação de cada aplicativo separado #
# #
#####
#
# plassee wrote by Reinaldo Marques de Lima #
# criado em: 10/2005 #
# descrição: #
# faz uma simples conferencia printando na tela o conteudo #
# do arquivo 'plasall.txt'. #
# #
##### comentarios sobre alguns problemas #####
#
# Nenhum, dar cat em um arquivo não é a coisa mais dificil #
# do mundo. #
# #
#####
#
# plasadd wrote by Reinaldo Marques de Lima #
# criado em: 08/2005 #
# descrição: #
# insere nomes de aplicativos a um arquivo externo que #
# vai ser lido pelo programa "plasall.bsh" para ser #
# executado. adiciona aplicativos a uma lista de execução. #
# #
##### comentarios sobre alguns problemas #####
#
# - Esse foi um pouco mais dificil do que eu pensei para #
# fazer, pois pensei..."Ah, eh soh dar um cat com ">>" pra #
# concatenar e tah valendo..." naum foi bem assim. tive #
# varios problemas pra conseguir fazer o cat rolar e quando #
# onseguir o script naum finalizava...ai naum teve jeito, #
# tive que apelar pra galera do grupo de discucao do yahoo #
# sobre shell para se associar - #
# (shell-script-subscribe@yahoogrupos.com.br ), o povo ajuda #
# de verdade, naum eh que nem esses grupos que tem aos #
# montes que soh mandam porcarias para o grupo eh soh shell #
# e pronto. Ai percebi que com o cat naum rola e sim com #
# echo redirecionado, blz, ai funcionou redondo. Soh faltou #
# fazer as tratativas de erro. #
# #
# Obs.: Esse foi o primeiro script que eu adicionei um #
# comando de ajuda " -- help " #
# #

```

```
#####
#
# plasall wrote by Reinaldo Marques de Lima
# criado em: 10/2005
# descrição:
# executa aplicativos que estiverem na lista do arquivo
# plasall.txt em forma de comandos e põe-os pra rodar
#
##### comentarios sobre alguns problemas #####
#
# - O problema maior deste script é fazer com que ele
# continue rodando após a execução do primeiro aplicativo
# pois ele se mantem travado enquanto o aplicativo não é
# finalizado.
# - A solução era mais simples do que eu imaginava, pois era
# só adicionar um 'e comercial & ' no final de cada um dos
# programas que estavam sendo iniciados no laço do for para
# que tudo desse certo.
#
#####
#
# plasrm wrote by Reinaldo Marques de Lima
# criado em: 10/2005
# descrição:
# remove os nomes de aplicativos especificados da lista
# gerada em "plasall.txt"
#
##### comentarios sobre alguns problemas #####
#
# - A intenção nesse script é utilizar o comando 'sed' que é
# um comando muito poderoso no que se refere a manipulação
# de texto.
# - Um dos problemas maiores foi acertar qual a maneira
# correta de utilizar o 'sed' e depois conseguir com que
# fosse escrito corretamente na saída do script os outros
# resultado que não seriam removidos, eu manjava muito
# pouco de sed, por tanto recorri denovo a galera do
# grupo, muito deste "pacote" de script se deve a eles
#
#####
#
# plaskill wrote by Reinaldo Marques de Lima
# antigo nome: "matador"
# criado em: 08/2005
# descricao:
# solicita ao usuario o nome de 1 ou 2 programas , que
# ele pesquisa e da um "kill" no programa solicitado, se
# esse programa/processo nao for o "X" claro :P
#
##### comentarios sobre alguns problemas #####
#
# - Inicialmente ele apenas rastreava o processo, fosse ele
# qual fosse, o primeiro obstaculo foi que esse processo
# poderia ser o "X" :-\, ai eh barra, um script matar seu X
# e fechar sua interface grafica nem rola, ai tratei esse
# erro, pensar nisso eh facil, desde que voce saiba como
# fazer.
# - O proximo passo foi mostrar uma mensagem de erro se
# o usuario digita um nome de aplicativo que o sistema
# nao encontra.
# - O passo seguinte foi colocar a opcao de matar mais de um
# processo usando array, e tambem mostrar uma msgem de erro
# quando o usuario nao digita "s" ou "n" na opcao de matar
# mais de um processo.
#
#####
```

```

#-----#
# inicio do programa, condições de uso                                     #
#-----#

#-----#
# apresenta o programa e as opções do menu                               #
#-----#

if [ $# != 0 ]; then
    "
        ERRO:
        O Pacote 'plaspkg' não aceita
        Parametro inseridos diretamente
        da linha de comando
        Saindo Agora
    " && sleep 3
    clear
    exit
fi

echo
echo "
    ***** Bem vindo ao programa 'plaspkg versão 0.1'*****
    *
    *   Escrito por Reinaldo Marques de Lima (Plastico)
    *   Este programa é de código aberto e livre para ser
    *   alterado de acordo com as necessidades de cada
    *   usuario.
    *
    *               ***{[( Atenção )]}***
    *
    *   Se esta for a primeira vez que você está usando o
    *   programa, convem ler o conteúdo da Opção 6 ( Ajuda )
    *   e logo em seguida inserir os primeiros dado com a
    *   Opção 2 ( plasadd ).
    *
    *****"

echo

menu() {

    echo "    Opções: "
    echo
    echo "
        Digite:
        1 - Para 'plassee'
        2 - Para 'plasadd'
        3 - Para 'plasall'
        4 - Para 'plasrm'
        5 - Para 'plaskill'
        6 - Para Ajuda
        7 - Para Sair"

    echo
    echo "    Escolha uma opção: "
    echo
    read OPCAO

    if [[ $OPCAO -lt 1 || $OPCAO -gt 7 ]]; then
        echo
        echo "    Parametro invalido, saindo agora"
        echo
        sleep 2
        clear
        exit
    fi
}

```

```

funcao.OPCAO

}

#-----#
# segunda parte, inclui cada programa nas funções      #
#-----#

#-----#
# comando 'case' para direcionar a opção do menu      #
#-----#

funcao.OPCAO() {

    case $OPCAO in

        1) plassee; menu; ;;
        2) plasadd; menu; ;;
        3) plasall; ;;
        4) plasrm; menu; ;;
        5) plaskill; menu; ;;
        6) ajuda; menu; ;;
        7) sair; ;;

    esac

}

#-----#
# terceira parte, rotinas                                #
#-----#

#-----#
# plassee: opção 1                                       #
#-----#
# mostra na tela as ocorrencias contidas em 'plasall.txt' #
#-----#

plassee() {

    MOSTRA=`cat plasall.txt`

    echo
    echo "    O conteudo da Lista é:"
    echo
    for i in ${MOSTRA[*]}; do

        echo $i

    done
    echo
    echo "    Voltando ao Menu"
    echo
    sleep 5
    clear

unset MOSTRA

}

#-----#
# plasadd: opção 2                                       #
#-----#
# alimenta uma lista de aplicativo para serem iniciados  #
#-----#

```

```

plasadd() {
    echo
    echo "    Digite os nomes dos programas: "
    read PROGRAMAS
    ARQUIVO="plasall.txt"

    for i in ${PROGRAMAS}; do

        if ! grep -w ${i} ${ARQUIVO} > /dev/null; then

            echo ${i} >> $ARQUIVO
            echo "    '${i}' cadastrado em 'plasall.txt'."

        else

            echo "    Argumento '${i}' já existe na lista."

        fi

    done

    echo "    Voltando ao menu" && sleep 5 && clear
    echo

unset PROGRAMAS
unset ARQUIVO
}

#-----#
# plasall: opção 3                                     #
#-----#
# executa tudo que estiver listado em 'plasall.txt'      #
#-----#

plasall() {
LISTA=`cat plasall.txt`

    echo "    Iniciando a Lista de execução" && sleep 3
    echo
    for i in ${LISTA}; do

        if [ `ps -ef|grep $i | wc -l` -gt 1 ]; then

            echo "    Aplicativo $i ja está em execução."
            echo

        else

            echo "    Iniciando aplicativo $i"
            echo
            $i & >> /dev/null
            sleep 3

        fi

    done

    echo
    echo "    Fim da Lista de execução, por favor tecle ENTER para sair."
    echo
    exit

unset LISTA

```

```

}

#-----#
# plasrm: opção 4                                     #
#-----#
# retira um nome de aplicativo da lista em 'plasall.txt' #
#-----#

plasrm() {

    echo
    echo "    Digite o programa que será removido da lista:"
    read PROGRAMA
    ARQUIVO="plasall.txt"
    TESTE=`grep $PROGRAMA plasall.txt`

    if [ ! $TESTE ]; then

        echo
        echo "
            Argumento $PROGRAMA não existe na lista
            Voltando ao menu" && sleep 5 && clear

        echo

    else

        sed "/$PROGRAMA/d" $ARQUIVO > .tmp
        cat .tmp > $ARQUIVO
        rm -f .tmp

        echo
        echo "
            O argumento $PROGRAMA foi removido da lista
            Voltando ao menu" && sleep 5 && clear

        echo

    fi

unset ARQUIVO
unset PROGRAMA
unset TESTE

}

#-----#
# plaskill: opção 5                                     #
#-----#
# mata processos buscando o PID                         #
#-----#

plaskill() {

    KILL="/bin/kill"

    echo
    echo "    Digite o nome do processo que voce quer matar: "
    read APLICATIVO1

    echo
    echo "    Voce quer matar mais algum processo? (s/n) "
    read RESPOSTA

    if [ $RESPOSTA != "s" ] && [ $RESPOSTA != "n" ]; then

        echo
        echo "        ERRO: escolha somente (s) ou (n)"
        echo "        Saindo do Programa" && sleep 5 && exit

    fi

}

```

```

        echo

    fi

    if [ $RESPOSTA = s ]; then

        echo
        echo "    digite: "
        read APLICATIVO2
        PROCESSO=($APLICATIVO1 $APLICATIVO2)
    else

        PROCESSO=($APLICATIVO1)

    fi

    for i in ${PROCESSO[*]}; do

        if [ $i = X ]; then

            echo
            echo "    ts ts, fazendo caca!!!"
            echo "    Voltando ao menu" && sleep 5 && clear
            echo

            elif [ `ps -ef|grep $i | wc -l` -lt 2 ]; then

                echo
                echo "    Aplicativo $i não esta rodando ou nome esta errado"
                echo "    Voltando ao menu" && sleep 5 && clear
                echo

            else

                JOB=`ps -ef|grep $i | awk {'print $2'} | head -1`

                $KILL $JOB

                echo
                echo "    O processo $JOB referente ao aplicativo $i foi finalizado"
                echo

            fi

        done

    unset KILL
    unset JOB
    unset PROCESSO

}

#-----#
# ajuda: opção 6                                     #
#-----#
# explicação basica de como usar o programa          #
#-----#

ajuda() {

    echo "
        *****
        *                                           *
        *   Ajuda do Programa 'plaspkg'           *
        *                                           *
        * Este programa foi criado para automatiza  *
        * a execução de alguns programas, por exemplo *
    "

```



```

* ( amsn, firefox, xmms) ou qualquer outro      *
* programa de sua escolha.                        *
*                                                  *
* Escolha uma das opções do menu para uma        *
* determinada função:                            *
*                                                  *
* 1) plassee - mostra na tela o conteudo do      *
* arquivo 'plasall.txt' que é a lista de         *
* execução de programas.                         *
*                                                  *
* 2) plasadd - adiciona aplicativos a uma         *
* lista de execução, essa lista será usada       *
* pelo 'plasall' para executar esses            *
* aplicativos.                                    *
* Obs.: Se for a primeira vez que você          *
* estiver usando essa opção, pode aparecer      *
* uma mensagem de erro dizendo que o arquivo     *
* 'plasall.txt' não foi encontrado, mas isso     *
* acontece somente na primeira vez, e depois    *
* o proprio programa ja cria ele, se caso       *
* você quiser evitar isso, basta criar no seu   *
* home este arquivo;                             *
*                                                  *
* 3) plasall - executa todos os aplicativos       *
* que estiverem listados no arquivo plasall.txt *
* e depois sai do programa;                      *
*                                                  *
* 4) plasrm - simplesmente tira da lista os      *
* aplicativos que forem digitados ( faz o       *
* inverso da opção plasadd);                     *
*                                                  *
* 5) plaskill - busca processos pelo numero      *
* e da um kill no processo, serve para todos    *
* os processos, menos para o X ;) e sai do      *
* programa.                                       *
*                                                  *
*****"

echo
echo "      Voltando para o menu em 1 minuto!!! " && sleep 60 && clear
echo

}

# sair: opção 7
#-----
# esse não é dificil entender
#-----

sair() {

    echo
    echo "      Saindo do programa..."
    echo "      Até logo. " && sleep 2
    echo
    clear
    exit
}

menu

```

Plaspkg dialog

Mesmo pacote, mas com a interface amigável do dialog.

```
#!/bin/bash
```

```
#####
#
# plaspkg_dialog wrote by Reinaldo Marques de Lima (Plastico)
# criação iniciada em: 11/2005
# concluído em: em andamento
# última atualização: 18/11/2005
# descrição:
# pacote de scripts para automatizar a execução de programas
# roda aplicativos de uma lista, inclui e exclui aplicativos
# da lista, mostra o conteúdo da lista e mata processos que
# estejam travados.
#
# 05/11 - Início da implementação dos códigos do programa
# 'plaspkg' para dialog, Apresentação e opção 'plassee'
# 07/11 - Implementação da Ajuda do programa para dialog
# 08/11 - Implementação do código da opção plasadd
# 17/11 - Implementação dos códigos das opções plaskill e
# plasrm
# 18/11 - Implementação do código da opção plasall
#
#####
#
# A mesma versão do pacote 'plaspkg' em versão com dialog
#
#####
```

```
CREDITOS='plaspkg_dialog criado por Reinaldo Marques de Lima (Plastico)'
```

```
#-----#
# primeiro dialog, apresentação do programa
#-----#
```

```
APRESENTACAO='
```

```
    Bem vindo ao programa plaspkg versao 0.1
```

```
    Escrito por Reinaldo Marques de Lima (Plastico)
    Este programa eh de codigo aberto e livre para ser
    alterado de acordo com as necessidades de cada
    usuario.
```

```
----- Atencao -----
```

```
    Se esta for a primeira vez que voce estah usando o
    programa, convem ler o conteúdo da Opcao 6 ( Ajuda )
    e logo em seguida inserir os primeiros dados com a
    Opcao 2 ( plasadd ).
```

```
,
```

```
dialog --backtitle "$CREDITOS"      \
      --title 'Apresentacao'         \
      --msgbox "$APRESENTACAO"       \
      0 0 &&
```

```
[ $? -ne 0 ] && echo 'Esc ou CANCELAR apertado'
```

```
#-----#
# segundo dialog, menu do programa
#-----#
```

```
OPCAO=$( dialog --stdout \
```

```

--backtitle "$CREDITOS" \
--title 'Menu' \
--menu 'Escolha a opcao desejada e de OK' \
0 0 0 \
1 'Plassee, ve o conteudo da lista' \
2 'Plasadd, adiciona nomes a lista' \
3 'Plasall, executa o que estiver na lista' \
4 'Plasrm, remove ocorrencias da lista' \
5 'Plaskill, mata um processo qualquer' \
6 'Ajuda' )

[ $? -ne 0 ] && echo 'Esc ou CANCELAR apertado'

#-----#
# aqui entra o case que executa a opção escolhida #
#-----#

funcao.OPCAO() {

    case "$OPCAO" in

        1) plassee; ;;
        2) plasadd; ;;
        3) plasall; ;;
        4) plasrm; ;;
        5) plaskill; ;;
        6) ajuda; ;;

    esac

}

#-----#
# terceiro dialog, plassee: opção 1 #
#-----#
# mostra na tela as ocorrencias contidas em 'plasall.txt' #
#-----#

plassee() {

    dialog --backtitle "$CREDITOS" \
--title '( Plassee ) o conteudo do arquivo eh:' \
--textbox plasall.txt \
0 0

[ $? -ne 0 ] && echo 'Esc ou CANCELAR apertado'

}

#-----#
# quarto dialog, plasadd: opção 2 #
#-----#
# alimenta uma lista de aplicativo para serem iniciados #
#-----#

plasadd() {

PROGRAMAS=$( dialog --stdout \
--backtitle "$CREDITOS" \
--title '( Plasadd ) Insere programas a uma lista.' \
--inputbox 'Digite o nome do programa:' \
0 0 )

[ $? -ne 0 ] && echo 'Esc ou CANCELAR apertado'

ARQUIVO="plasall.txt"
TESTE=`cat plasall.txt | grep $PROGRAMAS`

```

```

if [ $PROGRAMAS = $TESTE ]; then

    dialog --backtitle "$CREDITOS" \
    --title '( Plasadd ) Insere programas a uma lista' \
    --msgbox 'Nome ja existe na lista' \
    0 0

[ $? -ne 0 ] && echo 'Esc ou CANCELAR apertado'

else

    for i in ${PROGRAMAS}; do

        echo ${i} >> $ARQUIVO

    done

    dialog --backtitle "$CREDITOS" \
    --title '( Plasadd ) Insere programas a uma lista' \
    --msgbox 'O programa foi cadastrado com sucesso' \
    0 0

[ $? -ne 0 ] && echo 'Esc ou CANCELAR apertado'

fi

}

#-----#
# quinto dialog, plasall: opção 3 #
#-----#
# executa tudo que estiver listado em 'plasall.txt' #
#-----#

plasall() {

LISTA=`cat plasall.txt`

    for i in ${LISTA}; do

        if [ `ps -ef|grep $i | wc -l` -gt 1 ]; then

            dialog --backtitle "$CREDITOS" \
            --title '( Plasall ) Inicia aplicativos' \
            --infobox 'Aplivativo ja esta em execucao.' \
            0 0

        else

            dialog --backtitle "$CREDITOS" \
            --title '( Plasall ) Inicia aplicativos' \
            --infobox 'Iniciando aplicativo, aguarde...' \
            0 0

            $i & >> /dev/null/
            sleep 3

        fi

    done

    dialog --backtitle "$CREDITOS" \
    --title '( Plasall ) Inicia aplicativos' \
    --msgbox 'Fim do processo' \
    de OK, Cancelar ou Esc para sair' \
    0 0

```

```

}

#-----#
# sexto dialog, plasrm: opção 4                                     #
#-----#
# retira um nome de aplicativo da lista em 'plasall.txt'          #
#-----#

plasrm() {

ARQUIVO="plasall.txt"

PROGRAMA=$( dialog --stdout \
--backtitle "$CREDITOS" \
--title '( Plasrm ) Remove ocorrencias da lista' \
--inputbox 'Digite o nome do programa que sera removido' \
0 0 )

[ $? -ne 0 ] && echo 'Esc ou CANCELAR apertado'

TESTE=`grep $PROGRAMA plasall.txt`

if [ ! $TESTE ]; then

    dialog --backtitle "$CREDITOS" \
    --title '( Plasrm ) Remove ocorrencias da lista' \
    --msgbox 'Argumento nao existe na lista.' \
    0 0

    [ $? -ne 0 ] && echo 'Esc ou CANCELAR apertado'

else

    sed "/$PROGRAMA/d" $ARQUIVO > .tmp
    cat .tmp > $ARQUIVO
    rm -f .tmp

    dialog --backtitle "$CREDITOS" \
    --title '( Plasrm ) Remove ocorrencias da lista' \
    --msgbox 'O argumento foi removido da lista.' \
    0 0

    [ $? -ne 0 ] && echo 'Esc ou CANCELAR apertado'

fi

}

#-----#
# setimo dialog, plaskill: opção 5                                   #
#-----#
# mata processos buscando o PID                                     #
#-----#

plaskill() {

KILL="/bin/kill"

APLICATIVO1=$( dialog --stdout \
--backtitle "$CREDITOS" \
--title '( Plaskill ) Mata processos travados' \
--inputbox 'Digite o nome do processo que voce quer matar:' \
0 0 )

[ $? -ne 0 ] && echo 'Esc ou CANCELAR apertado'

dialog --stdout \

```

```

--backtitle "$CREDITOS" \
--title '( Plaskill ) Atencao' \
--yesno 'Deseja Matar mais algum processo?' \
0 0

if [ $? -eq 0 ]; then

    APLICATIVO2=$( dialog --stdout \
    --backtitle "$CREDITOS" \
    --title '( Plaskill ) Segundo processo a ser terminado' \
    --inputbox 'Por favor digite outro nome' \
    0 0 )

    [ $? -ne 0 ] && echo 'Esc ou CANCELAR apertado'

    PROCESSO=($APLICATIVO1 $APLICATIVO2)
else

    PROCESSO=($APLICATIVO1)

fi

for i in ${PROCESSO[*]}; do

    if [ $i = X ]; then

        dialog --backtitle "$CREDITOS" \
        --title '( Plaskill ) Aviso' \
        --msgbox 'ts ts, fazendo caca!!!' \
        0 0

        [ $? -ne 0 ] && echo 'Esc ou CANCELAR apertado'

    elif [ `ps -ef|grep $i | wc -l` -lt 2 ]; then

        dialog --backtitle "$CREDITOS" \
        --title '( Plaskill ) Aviso' \
        --msgbox 'Aplicativo nao esta rodando ou nome esta errado' \
        0 0

        [ $? -ne 0 ] && echo 'Esc ou CANCELAR apertado'

    else

        JOB=`ps -ef|grep $i | awk {'print $2'} | head -1`
        $KILL $JOB
        dialog --backtitle "$CREDITOS" \
        --title '( Plaskill ) Aviso' \
        --msgbox 'Processo(s) finalizado(s) com sucesso' \
        0 0

        [ $? -ne 0 ] && echo 'Esc ou CANCELAR apertado'

    fi

done

}

#-----#
# oitavo dialog, ajuda: opção 6 #
#-----#
# ajuda do programa #
#-----#

ajuda() {

```

AJUDA='

Ajuda do Programa 'plaspkg'

Este programa foi criado para automatiza a execucao de alguns programas, por exemplo (amsn, firefox, xmms) ou qualquer outro programa de sua escolha.

Escolha uma das opcoes do menu para uma determinada funcao:

1) plassee - mostra na tela o conteudo do arquivo 'plasall.txt' que eh a lista de execucao de programas.

2) plasadd - adiciona aplicativos a uma lista de execucao, essa lista sera usada pelo 'plasall' para executar esses aplicativos.

Obs.: Se for a primeira vez que voce estiver usando essa opcao, pode aparecer uma mensagem de erro dizendo que o arquivo 'plasall.txt' nao foi encontrado, mas isso acontece somente na primeira vez, e depois o proprio programa ja cria ele, se caso voce quiser evitar isso,basta criar no seu home este arquivo;

3) plasall - executa todos os aplicativos que estiverem listados no arquivo plasall.txt e depois sai do programa;

4) plasrm - simplesmente tira da lista os aplicativos que forem digitados (faz o inverso da opcao plasadd);

5) plaskill - busca processos pelo numero e da um kill no processo, serve para todos os processos, menos para o X ;) e sai do programa.'

```
dialog --backtitle "$CREDITOS" \
--title 'Ajuda' \
--msgbox "$AJUDA" \
0 0
```

```
[ $? -ne 0 ] && echo 'Esc ou CANCELAR apertado'
```

```
}
```

funcao.OPCAO

echo 'Obrigado por usar o programa Plaspkg versão dialog'

Plaspkg zenity

Mais uma versão, mas agora em zenity.

#!/bin/bash

```
#####
#
# plaspkg_zenity wrote by Reinaldo Marques de Lima ( Plastico ) #
# criação iniciada em: 12/2005 #
# concluido em: em andamento #
# ultima atualização: 26/12/2005 #
# descrição: #
# pacote de scripts para automatizar a execução de programas #
# roda aplicativos de uma lista, inclui e exclui aplicativos #
# da lista,mostra o conteudo da lista e mata processos que #
# estejam travados. #
#
```

```
#####
#
# A mesma versão do pacote 'plaspkg' em versão com zenity
#
#####

#.....Variaveis.....#

TITLE="Plaspkg - Zenity"
PID=`ps -ef|grep $0 | awk {'print $2'} | head -1`
KILL="/bin/kill"

#.....Apresentação.....#

echo "
Bem vindo ao programa  plaspkg versao 0.1

Escrito por Reinaldo Marques de Lima (Plastico)
Este programa eh de codigo aberto e livre para ser
alterado de acordo com as nercessidades de cada
usuario.

-----  Atencao  -----

Se esta for a primeira vez que voce estah usando o
programa, convem ler o conteudo da Opcao 6 ( Ajuda )
e logo em seguida inserir os primeiros dado com a
Opcao 2 ( plasadd )." > apresenta.txt

zenity --title="$TITLE"
        --text-info
        --filename apresenta.txt
        --width=400
        --height=400

rm apresenta.txt

#.....Menu de Opções.....#

OPCAO=$( zenity --title="$TITLE"
        --text "Escolha a opcao desejada"
        --list
        --radiolist
        --column "Escolha"
        FALSE plassee FALSE plasadd FALSE plasall FALSE plasrm
        FALSE plaskill FALSE ajuda
        --column "Programas"
        --width=250
        --height=240 )

#.Se a opção não for nula o 'case' direciona a opção escolhida.#

[ -z $OPCAO ] && zenity --info --text "Opcao Nula, Saindo" && $KILL $PID

#.....Funções.....#

funcao.OPCAO() {

    case "$OPCAO" in

        plassee) plassee; ;;
        plasadd) plasadd; ;;
        plasall) plasall; ;;
        plasrm) plasrm; ;;
        plaskill) plaskill; ;;
        ajuda) ajuda; ;;


```



```

        esac
    }

    plassee() {

        zenity --title="$TITLE" \
        --text "O conteudo do arquivo eh:" \
        --text-info \
        --filename plasall.txt

    }

    plasadd() {

ARQUIVO="plasall.txt"
TESTE=`cat plasall.txt | grep $PROGRAMA`

        PROGRAMA=$( zenity --title="$TITLE" \
        --text "Digite o programa a ser inserido:" \
        --entry )

[ -z $PROGRAMA ] && zenity --info --text "Opcao Nula, Saindo" && $KILL $PID

if [ $PROGRAMA = $TESTE ]; then

        zenity --title="$TITLE" \
        --text "Programa jah existe na lista" \
        --error

else

        for i in ${PROGRAMA}; do

            echo ${i} >> $ARQUIVO

        done

        zenity --title="$TITLE" \
        --text "Programa cadastrado com sucesso" \
        --info

fi

    }

    plasall() {

LISTA=`cat plasall.txt`

        zenity --title="$TITLE" \
        --text "Iniciando programas, aguarde..." \
        --info

for i in ${LISTA}; do

        if [ `ps -ef|grep $i | wc -l` -gt 1 ]; then

            zenity --title="$TITLE" \
            --text "Programa ja esta rodando" \
            --error

        else

            $i & >> /dev/null
            sleep 3


```

```

        fi

done

        zenity --title="$TITLE" \
        --text "Processo finalizado, saindo agora" \
        --info && $KILL $PID

    }

    plasrm() {

        ARQUIVO="plasall.txt"
        TESTE=`cat plasall.txt | grep $PROGRAMA`

        PROGRAMA=$( zenity --title="$TITLE" \
            --text "Digite o programa a ser removido:" \
            --entry )

        [ -z $PROGRAMA ] && zenity --info --text "Opcao Nula, Saindo" && $KILL $PID

        if [ ! $TESTE ]; then

            zenity --title="$TITLE" \
            --text "Programa nao consta na lista." \
            --error

        else

            sed "/$PROGRAMA/d" $ARQUIVO > .tmp
            cat .tmp > $ARQUIVO
            rm -f .tmp

            zenity --title="$TITLE" \
            --text "Nome removido da lista" \
            --info

        fi

    }

    plaskill() {

        APLICATIVO1=$( zenity --title="$TITLE" \
            --text "Digite o processo que voce quer matar" \
            --entry )

        zenity --title="$TITLE" \
        --text "Quer matar mais algum processo?" \
        --question

        if [ $? -eq 0 ]; then

            APLICATIVO2=$( zenity --title="$TITLE" \
                --text "Digite outro nome" \
                --entry )

            PROCESSO=($APLICATIVO1 $APLICATIVO2)

        else

            PROCESSO=($APLICATIVO1)

        fi

        [ -z $PROCESSO ] && zenity --info --text "Opcao Nula, Saindo" && $KILL $PID
    }

```

```

for i in ${PROCESSO[*]}; do

    if [ $i = X ]; then

        zenity --title="$TITLE" \
        --text "ts ts, fazendo caca!!!" \
        --error && $KILL $PID

    elif [ `ps -ef|grep $i | wc -l` -lt 2 ]; then

        zenity --title="$TITLE" \
        --text "Programa nao esta rodando ou nome esta errado" \
        --error

    else

        JOB=`ps -ef|grep $i | awk {'print $2'} | head -1`
        $KILL $JOB
        zenity --title="$TITLE" \
        --text "Processo(s) finalizado(s) com sucesso" \
        --info

    fi

done

}

ajuda() {

echo "
#..... Ajuda do Programa 'plaspkg' .....#

Este programa foi criado para automatiza a execucao de
alguns programas, por exemplo ( amsn, firefox, xmms)
ou qualquer outro programa de sua escolha.

Escolha uma das opcoes do menu para uma determinada funcao:

1) plassee - mostra na tela o conteudo do arquivo
'plasall.txt' que eh a lista de execucao de programas.

2) plasadd - adiciona aplicativos a uma lista de execucao,
essa lista sera usada pelo 'plasall' para executar esses
aplicativos.
Obs.: Se for a primeira vez que voce estiver usando essa
opcao, pode aparecer uma mensagem de erro dizendo que o
arquivo 'plasall.txt' nao foi encontrado, mas isso acontece
somente na primeira vez, e depois o proprio programa ja
cria ele, se caso voce quiser evitar isso,basta criar no
seu home este arquivo;

3) plasall - executa todos os aplicativos que estiverem
listados no arquivo plasall.txt e depois sai do programa;

4) plasrm - simplesmente tira da lista os aplicativos
que forem digitados ( faz o inverso da opcao plasadd);

5) plaskill - busca processos pelo numero e da um kill
no processo, serve para todos os processos, menos para
o X ;) e sai do programa." > ajuda.txt

        zenity --title="$TITLE" \
        --text-info \
        --filename ajuda.txt \
        --width=500 \

```

```

--height=600

rm ajuda.txt

}

funcao.OPCAO

```

Plasconvert

Programa que converte arquivos .txt em .html, reconhecendo links e endereços de email. E converte .html ou arquivos no formato do navegador lynx em arquivos .txt. Este aplicativo não tem uma funcionabilidade tão boa quanto o txt2tags, ainda estou trabalhando nele, mas a ultima versão está razoavel, confira.

```
#!/bin/bash
```

```

##### [ Ficha Técnica: ] #####
#
# Plasconvert - versão 0.4
# Escrito por: Reinaldo Marques de Lima ( Plastico )
# Criado em: 01/2006
# Ultima Atualização: 13/01/2006
#
##### [ Descrição: ] #####
#
# Conversor em shell-script de arquivos texto para arquivos formato web
# e vice-versa.
#
##### [ Legenda dos comentarios: ] #####
#
#
# - Comentarios agrupados por parenteses "#( comentario )...#" , servem
# para especificar o agrupamento de comandos, por exemplo: variaveis,
# testes, funções, cases...etc
#
# - Comentarios agrupados por chaves "#[ comentario ]...#", servem para
# mostrar o que um trecho do código faz.
#
##### [ Evolução: ] #####
#
# Versão 0.1 - Simplesmente gerava o arquivo em formato html com as tags
# basicas ( html, head, title e body...)
#
# Versão 0.2 - O programa agora pega a três primeiras linhas do arquivo
# e usa como cabeçalho e subtítulo do documento.
#
# Versão 0.3 - Reconhece automaticamente emails e links ( http:// e www )
#
# Versão 0.4 - Usando a opção -t que converte para texto remove <tags> e
# também remove os links em formato [link] que são mostrados pelo lynx
# quando você salva um documento da web através dele.
#
# ( tenho muito trabalho a fazer )
#
#####
#( Variaveis do Programa ).....#

ARQUIVO=$2
LIMPO=$(echo $2 | sed 's/\..*//') #[ Remove a extensão do arquivo ].....#

#( Testes de parametro ).....#

```

```

#[ Se não for passado nenhum parametro ].....#

[ -z $1 ] && echo "
    $0: Parametro Inválido: -h para ajuda
    " && exit

#[ Se o parametro não for reconhecido pelo programa ].....#

if [[ -n $1 && $1 != -w && $1 != -h && $1 != -t ]]; then

    echo "
    $0: Parametro Inválido: -h para ajuda
    " && exit

else

#[ Se os parametros estiverem OK mas o $2 estiver zerado ].....#

    if [[ -z $2 && $1 = -w ]] || [[ -z $2 && $1 = -t ]]; then

        echo "
        $0: Falta de Parametros: -h para ajuda
        " && exit

    fi

fi

#( Funções ).....#

help() {

    echo "
    digite:'$0 -w [arquivo.txt]:Para converter para .html'

    ou

    digite:'$0 -t [arquivo.html]:Para converter para .txt'
    "

}

html() {

#( Variaveis da função ).....#

CABECALHO=$(head -1 $ARQUIVO)
SUBTITLE1=$(head -2 $ARQUIVO | tail -1)
SUBTITLE2=$(head -3 $ARQUIVO | tail -1)
PAGINA=$(sed '1,3d; s/\(.*\b\)/<A HREF="mailto:\1">\1</A>;
s/\(http:\/\/.*\b\)/<A HREF="\1">\1</A>;
s/\(www\..*\b\)/<A HREF="http:\/\/\1">\1</A>/' $ARQUIVO)

#[ Irão formatar o texto para gerar o html ].....#

TITLE="<TITLE>
$LIMPO
</TITLE>"
HEAD="<HEAD>
$TITLE
<H1>
<CENTER>
<B>
$CABECALHO
</B>
</CENTER>

```

```

</H1>
<H2>
<CENTER>
<I>
$SUBTITLE1
<BR>

$SUBTITLE2
</I>
</CENTER>
</H2>
</HEAD>"
BODY="<BODY bgcolor="white">
<FONT SIZE=3 FACE=verdana>
<PRE>
$PAGINA
</PRE>
</FONT>
</BODY>"
HTML="<HTML>
$HEAD
$BODY
</HTML>"

#[ O printf irá gerar o html mantendo as formatações do arquivo ].....#

    printf "$HTML" > $LIMPO.html
    echo "
    Texto $ARQUIVO convertido com sucesso!!!
    "

}

texto() {

#( Variaveis da função ).....#

#[ Substitui links de imagens e links http por nada e remove tags ].....#

    TEXTO=$(sed 's/\[.*\]//g; s/<[^>]*>//g;
    References/q' $ARQUIVO |grep -v References)

#[ O printf irá gerar o txt mantendo as formatações do arquivo ].....#

    printf "$TEXTO" > $LIMPO.txt
    echo "
    Texto $ARQUIVO convertido com sucesso!!!
    "

}

#( Case para executar a opção desejada ).....#

case $1 in

    -w) html; ;;
    -t) texto; ;;
    -h) help; ;;

esac

```

Plasinfo

Programa que busca informações em sites da web (condições e pevisão do tempo, loteria, cotação do dolar ...) e mostra na tela do terminal.
 Este script pode sofrer problemas devido a constantes mudanças nos sites utilizados nele.

```
#!/bin/bash

#### [ Ficha : ] #####
#
# plasinfo - versão 0.2.1
# Escrito por: Reinaldo Marques de Lima ( Plastico )
# criado em: 16/01/2006
# ultima atualização: 26/01/2006
#
#### [ Descrição : ] #####
#
# Programa que informa as ultimas noticias, condição do tempo, e do
# aeroporto de Congonhas (sp).
#
#### [ Evolução : ] #####
#
# Versão 0.1
# - Busca informações sobre as ultimas noticias, condição do tempo e do
# aeroporto de Congonhas.
#
# Versão 0.2
# - Adicionado, resultado da mega-sena e a cotação do Dolar e do Euro e
# previsão do tempo para 4 dias.
#
# Versão 0.2.1
# - Devido a constantes alterações em página da web tive que refazer a
# função da opção 'loteria', a função 'estado' (opção -news) ficou
# inutilizada por conta de uma destas alterações na página do estado de
# de São Paulo
#
#####

#( Testes ).....#

[ -z $1 ] && echo "$0 : Erro : use $0 -help ( para ajuda)" && exit

#( Funções ).....#

estado() {

URL=http://www.estadao.com.br/agestado/

ESTADAO=$(lynx -dump $URL |sed 's/\[.*\]//g;
/mais not.*/q' |grep -v mais |grep -v PUBLICIDADE |grep -v IFRAME |tail -32)

    echo "$ESTADAO
"

unset URL
}

folhatempo() {

URL=http://www1.folha.uol.com.br/folha/tempo/br-sao_paulo.shtml

FOLHA=$(lynx -dump $URL |sed 's/\[.*\]//g;
/PREVIS/q' |tail -15 | head -10 | uniq)

    echo "
$FOLHA
"

unset URL
```

```

}

aeroporto() {

URL=http://www.apolo11.com/tempo_historico.php?id=SBSP

AEROPORTOS=$(lynx -dump $URL | sed 's/\[.*\]//g; /Apolo11\.com/q' | tail -32 | head -30 | sed
'5,25!d' | column -t | sed 's/\([0-9]\{2\}\) \([0-9]\{2\}:[0-9]\{2\}\) \(.*\) \(.*\)
\([0-9]\{2\}.C\) \([0-9]\{2\}%\) \([0-9]\{4\}\) \(.*\) \(.*)/| \1 | \2 | \3
\4 | \5 | \6 | \7 | /g'`

    echo "
Histórico Meteorológico
SP - São Paulo
Aeroporto de Congonhas

+-----+-----+-----+-----+-----+-----+
| Dia | Hora | Tempo | Graus | U. Rel. | Pressão |
+-----+-----+-----+-----+-----+-----+
$AEROPORTOS
+-----+-----+-----+-----+-----+-----+
"

unset URL
}

loteria() {

URL=http://www.estadao.com.br/ext/loterias/

LOTERIA=$(lynx -dump $URL | sed 's/\[.*\]//g;
/Copyright/q' | tail -27 | sed 6q)

    echo "
$LOTERIA
"

unset URL
}

moeda() {

URL=http://www.estadao.com.br/economia/financas/cotacoes/resumo.htm

MOEDA=$(lynx -dump $URL | sed 's/\[.*\]//g; /CDB/q' | tail -8 | head -6)

    echo "
$MOEDA
"

unset URL
}

previsao() {

URL=http://www4.climatempo.com.br/site/espelho.php?estados=SP&pg=capitais&pc=estadao

PREVISAO=$(lynx -dump $URL | sed 's/\[.*\]//g; /References/q' | tail -58 | head -45)

    echo "$PREVISAO"

unset URL
}

help() {

    echo "

```


Programa que informa noticias na tela do terminal.

digite:

```
-news      : Para noticias da página do Estado de São Paulo.
-tempo     : Para informação do tempo agora.
-previsao  : Informa previsão dos proximos 4 dias.
-aero      : Para noticia da situação do aeroporto de congonhas.
-loteria   : Para saber o resultado da mega-sena.
-moeda     : Informa cotação do Dolar e do Euro.
```

}

#{ Case).....#

case \$1 in

```
-news) estado; ;;
-tempo) folhatempo; ;;
-aero) aeroporto; ;;
-loteria) loteria; ;;
-moeda) moeda; ;;
-help) help; ;;
-previsao) previsao; ;;
```

esac

#{ Fim).....#

Sysinfo

Programinha simples que passa informações do sistema.

#!/bin/bash

```
#####
#
# Sysinfo - versão 0.1
# Escrito por: Reinaldo Marques de Lima ( Plastico )
# ultima atualização: 02/01/2006
#
#####
#
# Descrição:
# - mostra de forma clara e detalhada a situação do sistema, informa
# nome do sistema, da maquina, versão do kernel, situação da memória, HD
# e tempo ativo do sistema .
#
#####
```

#{ Variaveis).....#

```
SISTEMA=$(uname)
DISTRO=$(sed 's/\\.*//' /etc/issue)
MAQUINA=$(uname -n)
KERNEL=$(uname -r)
HORA=$(date +%T)
DATA=$(date +%d/%m/%y)
UP=$(uptime | awk '{print $3}')
MEMT=$(free -m | grep Mem | awk '{print $2}')
MEMF=$(free -m | grep Mem | awk '{print $4}')
HDMB=$(df -m | grep hda1 | awk '{print $2}')
HDPC=$(df -m | grep hda1 | awk '{print $5}')
```

```

#( Mostrando tudo ).....#

echo "
+-----+
| Olá, eis aqui a situação do sistema agora: |
+-----+

Sistema: $SISTEMA ;
Distribuição: $DISTRO ;
Nome da Maquina: $MAQUINA ;
Versão do Kernel: $KERNEL ;
O sistema está up a $UP dias ;
A memória total do sistema é de $MEMT MB ;
Com $MEMF MB livres ;
O tamanho do HD é de $HDMB, com $HDPC usados

$HORA                                $DATA

+-----+
"

#( Fim ).....#

```

Sysinfo gmessage

Mesma versão, com interface gmessage.

```
#!/bin/bash
```

```

#####
#                                                                    #
# Sysinfo - versão 0.1                                              #
# Escrito por: Reinaldo Marques de Lima ( Plastico )              #
# ultima atualização: 09/0s/2006                                    #
#                                                                    #
#####
#                                                                    #
# Descrição:                                                        #
# - mostra de forma clara e detalhada a situação do sistema, informa #
# nome do sistema, da maquina, versão do kernel, situação da memória, HD #
# e tempo ativo do sistema .                                         #
#                                                                    #
#####

```

```

#( Variaveis ).....#

```

```

SISTEMA=$(uname)
DISTRO=$(sed 's/\.\.*//' /etc/issue)
MAQUINA=$(uname -n)
KERNEL=$(uname -r)
HORA=$(date +%T)
DATA=$(date +%d/%m/%y)
UP=$(uptime | awk '{print $3}')
MEMT=$(free -m | grep Mem | awk '{print $2}')
MEMF=$(free -m | grep Mem | awk '{print $4}')
HDMB=$(df -m | grep hda1 | awk '{print $2}')
HDPC=$(df -m | grep hda1 | awk '{print $5}')

```

```

#( Mostrando tudo ).....#

```

```

gmessage -geometry 450x450 -buttons "OK" " "
+-----+

```

```
| Olá, eis aqui a situação do sistema agora: |
+-----+
```

```
Sistema: $SISTEMA ;
Distribuição: $DISTRO ;
Nome da Maquina: $MAQUINA ;
Versão do Kernel: $KERNEL ;
O sistema está up a $UP dias ;
A memória total do sistema é de $MEMT MB ;
Com $MEMF MB livres ;
O tamanho do HD é de $HDMB MB,
com $HDPC usados.
```

\$HORA

\$DATA

```
+-----"
```

```
##( Fim ).....#
```

Go

Script que se conecta a um servidor remoto via ssh abrindo em um novo terminal.

```
#!/bin/bash
```

```
##### [ Ficha: ] #####
#
# Script "go" - versão única
# formulado por: Reinaldo Marques de Lima
# em: 18/01/2006
#
##### [ Descrição: ] #####
#
# Conecta-se a um servidor remoto via ssh
#
##### [ HOW-TO: ] #####
#
# - Antes de mais nada verifique se o usuário tem as
# permissões necessárias para se conectar remotamente a
# um servidor via ssh.
#
# - Para que o sistema reconheça o comando, copie o "go"
# para um dos diretórios do seu $PATH. ( echo $PATH )
#
# - Na linha de comando use: ~$ go 'nome do servidor'.
#
# - Não esqueça de trocar 'usuario' na linha abaixo pelo
# login que vai ser usado para se conectar ao servidor
# remoto.
#
#####
```

```
[ -z $1 ] && echo "$0: erro: use $0 [ servidor ]" && exit
```

```
xterm -T $1 -e ssh -vv usuario@$1
```

Scripts index

Script que cria uma página em html para visualizar todo o conteúdo do diretório "scripts" em forma de link.

```
#!/bin/bash

#####[ Ficha: ]#####
#
# Nome: Scripts Index
# Escrito por: Reinaldo Marques de Lima ( Plastico )
# Criado em: 22/02/2005
# Ultima atualização: 22/02/2005
#
#####[ Descrição: ]#####
#
# Cria uma página html com link para todos os script da máquina
#
#####

#( diretório dos scripts, altere aqui ).....#

cd /home/plastico/scripts/

#( looping simples para listar os arquivos ).....#

LISTA=$(ls *.bsh)
for i in ${LISTA[*]}; do

    echo "<A HREF=/home/plastico/scripts/$i>$i</A>" >> scripts.tmp

done

cd

#( formata o texto e escreve a página html ).....#

CORPO=$(sed 's/\(<.*>\)/home/plastico/scripts/\(<.*>\)/\1\2\3/g'
/home/plastico/scripts/scripts.tmp)

echo "
<HTML>
<HEAD>
<TITLE>
Lista de arquivos bsh
</TITLE>
<BODY>
<PRE>
$CORPO
</PRE>
</BODY>
</HTML>" > scripts.html

rm /home/plastico/scripts/scripts.tmp

#( Fim, facinho né? 8^ ) .....#
```

GoWalk

Script em interface zenity que mostra na tela um aviso a cada 50 minutos, insentivando a não ficar muito tempo na frente do computador.

```
#!/bin/bash

#####
#
# Go Walk - versão 0.1 ( zenity )
#
```

```

# Escrito por: Reinaldo Marques de Lima ( Plastico )
# Ultima atualização: 02/01/2006
#
#####
#
# Descrição:
# Baseado na preocupação do fisioterapeutas em manter um habito saudavel
# para usuários de computador, onde dizer que não se deve passar mais de
# 50 minutos consecutivos em frente ao computador por questoes esteticas
# desenvolvi este script para me policiar e tentar manter este regra
# básica que é bem facil de ser seguida e mantem uma qualidade de vida
# um pouco melhor para os usuários de informática como eu por exemplo.
#
#####

#( variaveis ).....#

OK=0

#( looping ).....#

while [ $OK = 0 ]; do

    zenity --title="Go Walk" \
    --text "Ja se passaram 50 minutos, vai dar uma volta, vai!" \
    --question

    if [ $? = 0 ]; then

        sleep 3000

    else

        OK=$(( OK+$? ))

    fi

done

```

GoWalk Xmessage

Mesma versão, mas para outras distribuições.

```
#!/bin/bash
```

```

#####
#
# Go Walk - versão 0.1 ( zenity )
# Escrito por: Reinaldo Marques de Lima ( Plastico )
# Ultima atualização: 02/01/2006
#
#####
#
# Descrição:
# Baseado na preocupação do fisioterapeutas em manter um habito saudavel
# para usuários de computador, onde dizer que não se deve passar mais de
# 50 minutos consecutivos em frente ao computador por questoes esteticas
# desenvolvi este script para me policiar e tentar manter este regra
# básica que é bem facil de ser seguida e mantem uma qualidade de vida
# um pouco melhor para os usuários de informática como eu por exemplo.
#
#####

```

```

#( variaveis ).....#
OK=0
#( looping ).....#
while [ $OK = 0 ]; do

    xmessage "Ja se passaram 50 minutos, vai dar uma volta, vai!" \
    -nearmouse \
    -buttons "OK:0,Cancel:1"

    if [ $? = 0 ]; then

        sleep 3000

    else

        OK=$(( OK+$? ))

    fi

done

```

Meu IP

Script que busca o endereço de IP e a Subnet-mask.

```

#!/bin/bash

#####
#
# Meu IP - versão 0.2
# escrito por: Reinaldo Marques de Lima ( Plastico )
# ultima atualização: 02/01/2006
#
#####
#
# Descrição:
# - Filtra o comando 'ifconfig -a' e passa, de acordo com o parametro
# escolhido, o endereço de IP ou a Subnet-Mask do usuário.
#
#####

#( Mensagens de erro - Para parametros invalidos ou sem parametros ).....#

[ -z $1 ] && echo "$0: Parametro Inválido: -h para ajuda"

if [[ -n $1 && $1 != -i && $1 != -s && $1 != -h ]]; then

    echo "$0: Parametro Inválido: -h para ajuda"

fi

#( Funções ).....#

ip() {

IP=`ifconfig -a | grep inet | head -1 | awk '{print $2}' | cut -d: -f2`

    echo
    echo "Seu IP é $IP"
    echo

```

```

}

mask() {

MASK=`ifconfig -a | grep Mask | head -1 | cut -d: -f4`

    echo
    echo "Sua subnet mask é $MASK"
    echo

}

help() {

    echo "
Use '$0 [opção]'

opções de parametro:

    -i    para saber o endereço de IP
    -s    para saber a mascara de sub-rede"
echo

}

#( Case para seleccionar a função ).....#

case $1 in

    -i) ip; ;;
    -s) mask; ;;
    -h) help; ;;

esac

#( Fim ).....#

```

Meu IP gmessage

Passa informações de forma visual pelo gmessage.

```
#!/bin/bash
```

```

#####
#                                                                 #
# Meu IP - Versão gmessage 0.1                                     #
# wrote by: Reinaldo Marques de Lima                             #
# ultima atualização: 26/12/2005                                   #
#                                                                 #
#####
#                                                                 #
# Descrição:                                                       #
# - Script que filtra a saída do comando ifconfig mostrando apenas o #
# número do IP e da Subnet-mask do usuário, usando o 'gmessage' do Gnome #
#                                                                 #
#####

#( Filtros ).....#

IP=`ifconfig -a | grep inet | head -1 | awk '{print $2}' | cut -d: -f2`
MASK=`ifconfig -a | grep Mask | head -1 | cut -d: -f4`

```

```

#( Mensagem do Gmessage ).....#

gmessage "Meu ip eh
$IP
e minha subnet mask eh
$MASK
"

#( Fim, facil né? ).....#

```

Considerações Finais

Agradecimentos

Em primeiro lugar quero agradecer a Deus todo poderoso por todas as graças e por sempre estar presente na minha vida.

Em segundo lugar quero agradecer a minha mulher Janaina (Jana) e a minha filhinha Julia (estrelinha) simplesmente por existirem, meus amores, não sei o que seria da minha vida sem vocês.

A todos da família Marques, agregados, parentes distantes etc, e aos da família Veiga.

Quero também agradecer aos meus amigos mais antigos da época do colegial que são os que eu guardo até hoje como meus irmãos. Os amigos da faculdade que agüentaram sempre juntos as Provas e tudo mais, os companheiros de trabalho do UOL que muito me ajudaram a crescer profissionalmente, aos membro das listas de discussão sobre shell-script e sed que sempre tiraram minhas duvidas, Um MUITO OBRIGADO a todos.

Bibliografias e Links

Referências para criação desta apostila.

(Livros)

- Bash - Guia de Consulta rápida / Joel Saade.
- Programação SHELL LINUX / Julio Cezar Neves. - 5 ed.
- Expressões Regulares - Guia de Consulta rápida / Aurélio Marinho Jargas.
- Comandos do Linux - Guia de Consulta rápida / Roberto G. A. Veiga.
- Servidor Web usando Apache / Melanie Hoag

(Páginas web)

Página do Aurélio:

<http://aurelio.net>

Página do Thobias:

<http://thobias.org>

Página do Julio:

www.julioneves.com

Página da Oracle sobre shell script:

http://www.oracle.com/technology/pub/articles/saternos_scripting.html

Página do MySQL sobre expressões regulares:

<http://dev.mysql.com/doc/refman/4.1/pt/regexp.html>

Página sobre kdialog:

<http://developer.kde.org/documentation/tutorials/kdialog/x85.html>

Guia avançado de bash scripting:

<http://www.tldp.org/LDP/abs/html/index.html>

Guia Foca/GNU Linux:

<http://focalinux.cipsga.org.br/>

Bash Programming HOWTO:

<http://www.tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html>

Exemplos de shell:

<http://planeta.yi.org/unix/exemplos.txt>

E mais um mooooooonte de páginas aleatórias de tópicos específicos em inglês e português que tiveram o google como ponto de partida.

