

De volta ao shell – mas com janelas

# Papo de botequim 2.0

## Parte II

Janelas de seleção de arquivo com o Zenity.

por Julio Cezar Neves

– E aí, vamos pedir os chopes?  
– Cara, estou gostando tanto deste zenity que estou mais com sede de saber do que de chope.

– Chico, traz um sem colarinho e se vira para arrumar um com saber para o meu amigo.

– E você fez o exercício que te passei na primeira aula?

– Fiz, veja no **exemplo 1**. Vamos analisar o código deste exemplo.

Primeiro você criou uma função só para fazer as perguntas, muito bom! Depois você fez um loop para

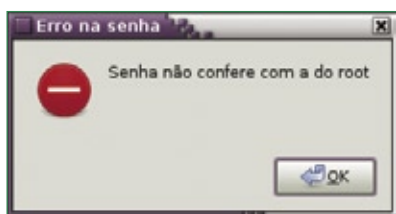
ler o nome e só sai dele quando a variável `$Nome` estiver preenchida (`[ "$Nome" ] && break`) ou aborta caso tenha sido clicado em OK na função (`Pergunta && exit 1`).

No primeiro bloco do loop de leitura da data de nascimento, caso tenha sido clicado em CANCELAR no calendário (`||`), você chama a função `Pergunta` e aborta se clicar em OK ou volta para o loop se clicar em CANCELAR (`Pergunta && exit 1 || continue`).

Ainda neste loop, você pegou a data de hoje no mesmo formato que usou para a data do calendário e testou se a diferença entre elas era maior que 10.000. Como ambas estão no formato AAAAMMDD, 10.000 significa que o último algarismo do ano atual é 1 a mais que o do nascimento. Gostei!

No loop de leitura do email, achei muito legal o uso do comando `test`, em que o `=~` significa estar usando expressões regulares (`[[ $Email =~ [[:alnum:]]+@[[:alnum:]] ]]`). O `[...]` representa o comando `test` propriamente dito, `=~` significa que você vai usar uma expressão regular e `[[:alnum:]]` é uma lista formada por letras maiúsculas, minúsculas, números (classe POSIX `[[:alnum:]]`), ponto (.) e hífen (-). É importante notar que expressões regulares só funcionam no comando `test` a partir da versão 3.1.17 do bash.

No loop de leitura dos telefones, achei bacana a linha do `grep` (`grep -q ' <<< ${Tel[i]} || continue`) por duas razões: o uso de *here strings* (`<<<`) que substitui com vantagens um `echo ${Tel[@]} | tr...` e o uso da opção `-q` (*quiet*) do `grep` que o torna mudo e



**Figura 1** Janela de erro.

### Tabela 1: Opção --error

Opção	Efeito
<code>--text=TEXTO</code>	Define o texto da caixa
<code>--no-wrap</code>	Não permite quebra automática do texto

### Exemplo 2: Diálogo de erro

```
1 zenity --error \
2   --title "Erro na senha" \
3   --text "Senha não confere com a do root"
```

**Exemplo 1: Exercício da primeira aula**

```

01 #!/bin/bash
02 # Programa didático em shell+zenity
03 # Cadastramento
04 function Pergunta
05 {
06     zenity --question \
07         --title "$Titulo" \
08         --text "$Texto"
09     return $?
10 }
11 while true
12 do
13     Nome=$(zenity --entry \
14         --title "Informações para o catálogo" \
15         --text "Informe o nome da pessoa a cadastrar")
16     [ "$Nome" ] && break
17     Titulo="Nome não informado"
18     Texto="Deseja abandonar?
19     - Clique em OK para sair do programa, ou
20     - Clique em CANCELAR para voltar"
21     Pergunta && exit 1
22 done
23 while true
24 do
25     Nasc=$(zenity --calendar \
26         --title "Cadastramento de aniversários" \
27         --text "Informe a data do aniversário de $Nome" \
28         --date-format "%Y%m%d") || {
29         Titulo="Data não informada"
30         Texto="Deseja abandonar?
31         - Clique em OK para sair do programa, ou
32         - Clique em CANCELAR para voltar"
33         Pergunta && exit 1 || continue
34     }
35     Hoje=$(date +%Y%m%d)
36     if [ $(($Hoje-$Nasc)) -lt 10000 ]
37     then
38         Titulo="Provável erro de informação"
39         Texto="Esta pessoa tem menos de um ano de idade.
40         - Clique OK para manter esta idade ou
41         - Clique CANCELAR para informar nova data"
42         Pergunta || continue
43     fi
44     break
45 done
46 while true
47 do
48     Email=$(zenity --entry
49         --title "Informações para o catálogo" \
50         --text "Informe o e-mail de $Nome")
51     [ "$Email" ] || break
52     [[ $Email =~ [[:alnum:]]+@[[:alnum:]]+ ] ] || {
53         zenity --warning \
54             --title "Erro no e-mail" \
55             --text "Caracteres estranhos no endereço de e-mail"
56         continue
57     }
58     break
59 done
60 while true
61 do
62     let i++
63     Tel[i]=$(zenity --entry \
64         --title "Informações para o catálogo" \

```

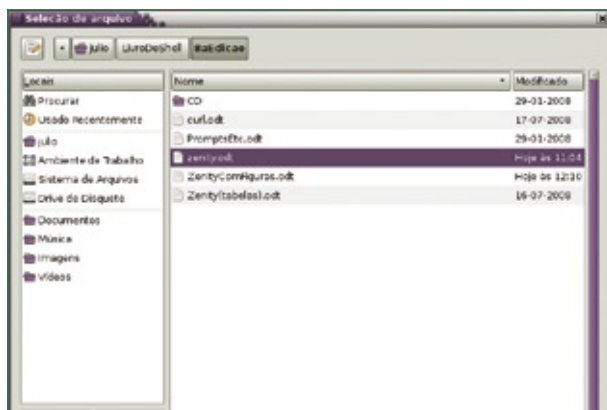


Figura 2 Janela de seleção de arquivo.

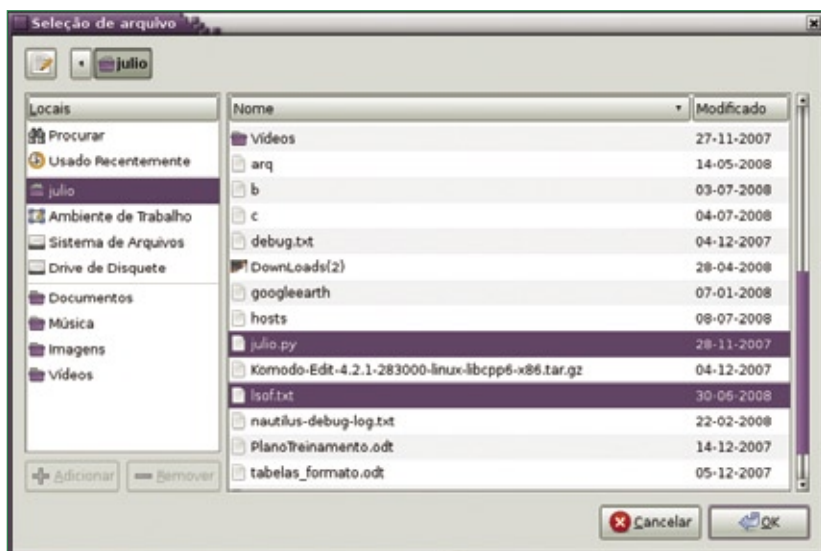


Figura 3 Janela para seleção de múltiplos arquivos.

### Exemplo 3: Confirmar antes de sobrescrever.

```
1 Arquivo=$(zenity --file-selection \
2   --title "Seleção de arquivo" \
3   --save \
4   --confirm-overwrite)
5 echo $Arquivo
6 /home/julio/UFs
```

### Exemplo 4: Abertura de múltiplos arquivos.

```
01 Arquivos=$(zenity --file-selection \
02   --title "Seleção de arquivo" \
03   --multiple \
04   --separator "^") || {
05   zenity --error \
06     --text "Nenhum arquivo foi selecionado"
07   exit 1
08 }
09 echo $Arquivos
10 /home/julio/julio.py^/home/julio/lsof.txt
```

não manda a saída para a tela.

Estou vindo que você aprendeu tudo que te contei sobre Bash no Papo de Botequim 1.0 que foi publicado a partir da primeira **Linux Magazine**. Mas como você disse que tinha sede de

saber quando chegamos aqui, vamos ver mais um pouco sobre o zenity.

A opção `--error` (tabela 1) exige uma janela acusando um erro, como na figura 1. Para ilustrar isso, suponha que a senha do exemplo anterior tenha sido informada errada. Você poderá fazer como no exemplo 2.

Assim como na opção `--question`, caso a caixa de erro quebrassem a mensagem definida na opção `--text` e isso não fosse desejado, você poderia incluir a opção `--no-wrap`. Procedendo desta forma, a caixa ficaria mais larga para caber a mensagem inteira. Não se esqueça que o mesmo poderia ser feito aplicando a opção `--height`, que pode ser usada em qualquer caixa de diálogo.

Use essa opção para conseguir o caminho absoluto de um arquivo selecionado. É rápida e simples. Veja o trecho de programa no exemplo 3, que cria a janela mostrada na figura 2.

Podemos também selecionar múltiplos arquivos. Para isso é necessário usarmos a opção `--multiple` e é aconselhável usarmos também a opção `--separator`. Veja o exemplo 4, que produz a janela mostrada na figura 3.

Repare que os arquivos selecionados vieram separados por um circunflexo (^), que foi a opção feita pelo `--separator` (exemplo 4). Podemos também usar esta opção para pegar um arquivo que será sobrescrito. Usando `--confirm-overwrite` juntamente com `--save`. Caso o arquivo escolhido já exista, será automaticamente aberta uma caixa de perguntas (`--question`). Se você clicar no botão OK, ambas as caixas abertas serão fechadas e será devolvido o caminho completo do nome do arquivo selecionado. Se a opção feita for o botão CANCEL, a caixa de perguntas será fechada para que se faça uma nova escolha de arquivos.

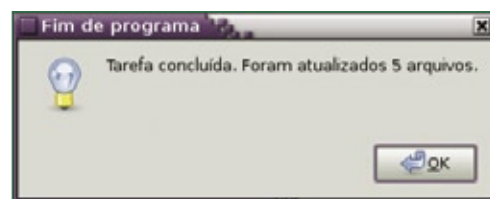
A opção `--directory` inibe a escolha de arquivos comuns, permitindo acesso somente aos diretórios.

**Tabela 2: Opção --file-selection**

Opção	Efeito
<code>--filename=NOME</code>	Define o nome do arquivo ou diretório inicial (default)
<code>--multiple</code>	Permite a seleção de diversos arquivos ou diretórios
<code>--directory</code>	Permite somente a seleção de diretórios
<code>--save</code>	Salva o arquivo selecionado
<code>--confirm-overwrite</code>	Usado com <code>--save</code> , pede confirmação caso o arquivo já exista
<code>--separator=SEPARADOR</code>	Usado com <code>--multiple</code> , especifica o separador quando retornar mais de um arquivo

**Tabela 3: Opção --info**

Opção	Efeito
<code>--text=TEXT0</code>	Define o texto da caixa
<code>--no-wrap</code>	Não permite quebra automática do texto

**Figura 5** Janela de informação.

Temos ainda a opção `--filename="/caminho/do/arquivo"`, que posiciona a caixa de seleção de arquivos no diretório `/caminho/do` e oferece `arquivo` como padrão no campo Nome. Se `/caminho/do` fizer parte da variável `$PATH`, basta especificar apenas o nome deste.

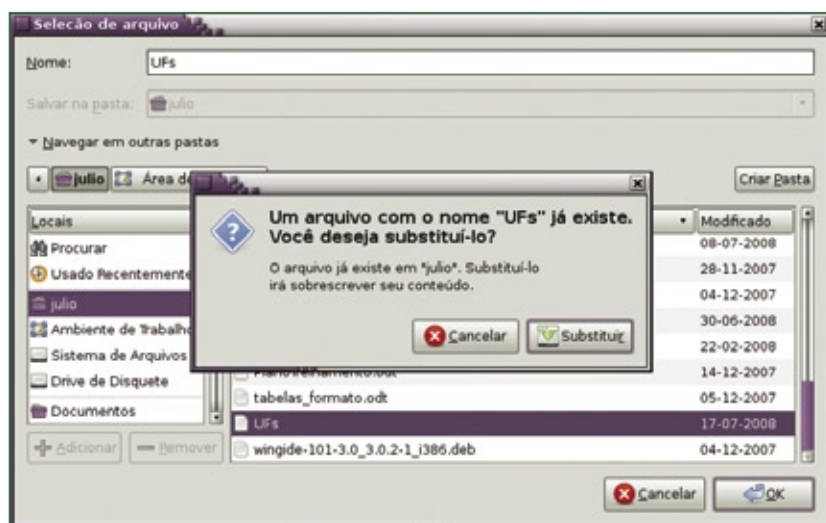
A opção `--info`, detalhada na **tabela 3**, exibe uma janela de informação, como na **figura 5**.

Use a opção `--info` para informar algo, como no **exemplo 5**.

Caso a caixa de informações quebras a mensagem definida pela opção `--text` e isso não fosse desejado, você poderia incluir a opção `--no-wrap`. Procedendo desta forma, a caixa ficaria mais larga para caber a mensagem inteira, sem quebrá-la em mais de uma linha. Não se esqueça que o mesmo poderia ser feito aplicando a opção `--height`, que pode ser usada em qualquer caixa de diálogo.

– Chico, traz dois chopes que já estou de goela seca de tanto falar. Acho que se cuspir, sai em pó!

Não pense que você vai se livrar do programinha para fazer em casa. Desta vez vou passar um bem curtinho: faça um programa que permita ao operador escolher um diretório (somente diretórios) para onde serão copiados determinados arquivos. Em seguida deixe-o escolher quais arquivos do diretório corrente (todos de uma vez) serão copiados para aquele destino. Caso já exista arquivo com o mesmo nome, pedir confirmação. ■

**Figura 4** Janela para confirmar a sobrescrita de arquivo.**Exemplo 5: Exibir uma janela de informação.**

```
1 zenity --info \
2   --title="Fim de programa" \
3   --text="Tarefa concluída. Foram atualizados 5 arquivos."
```

Agradecimentos especiais ao SERPRO, empresa exemplo do uso de Software Livre no Governo Federal.