Shell Scripting – Criando caixas de diálogo TUI com whiptail no Linux – 27

Postado Em 12/01/2015 [data do post] por Fábio dos Reis [autor do post] em Shell Scripting [categoria do post]

Tweet

Criando caixas de diálogo com whiptail no Linux

Neste artigo aprenderemos a criar caixas de diálogo via Shell Scripting usando a ferramenta **whiptail**.

O whiptail é um programa que nos permite exibir diversos tipos de caixas de diálogo com questões ou mensagens aos usuários a partir de um script do shell. O whiptail utiliza a biblioteca de programação newt, a qual é escrita em linguagem C.

Os tipos de caixas de diálogo implementadas atualmente são: yes/no, inputbox, messagebox, textbox, infobox, checklist, radiolist, gauge e passwordbox.

Usaremos um arquivo de script chamado **caixas.sh** para nossos testes. Crie-o com o editor vi (ou outro de sua preferência) e não se esqueça de dar permissão de execução após terminar de editá-lo e salvá-lo:

```
# chmod 755 caixas.sh
```

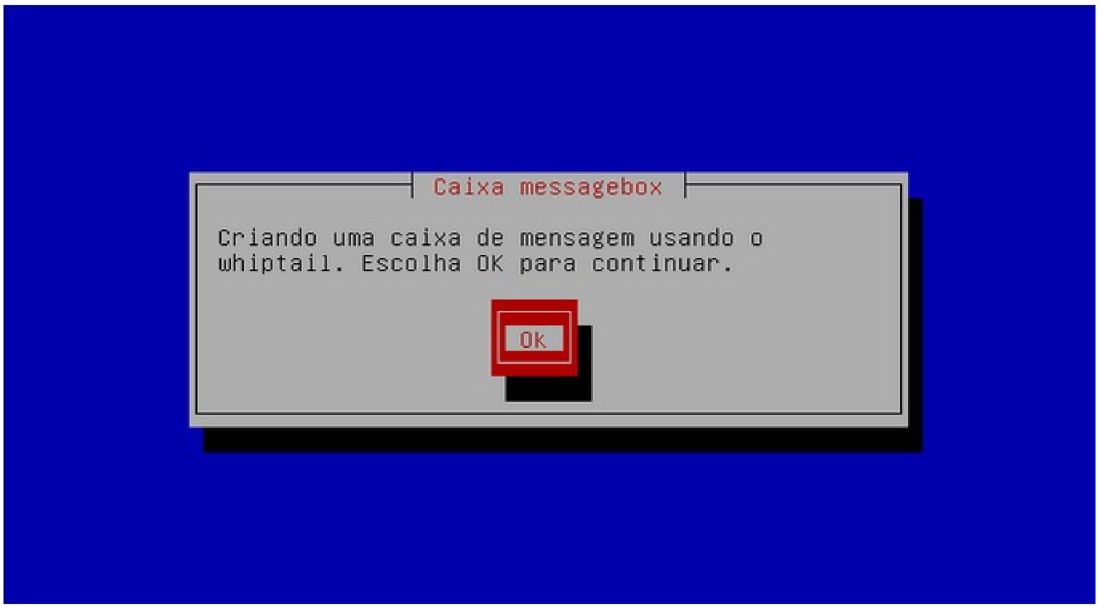
1. Vamos começar criando uma Caixa de Mensagem (Message Box). Podemos criar um message box que contenha uma mensagem qualquer e um botão de confirmação usando a sintaxe a seguir: whiptail –title "<título do message box>" –msgbox "<texto que será exibido>" <altura> <largura>

Exemplos:

```
#!/bin/bash
whiptail --title "Caixa de Mensagem de Teste" --msgbox "Criando uma caixa de
mensagem usando o whiptail. Escolha OK para continuar." --fb 10 50
```

A opção –**fb** permite o uso de botões completos (por padrão, o whiptail usa uma versão simplificada dos botões). Testando:

```
# ./caixas.sh
```

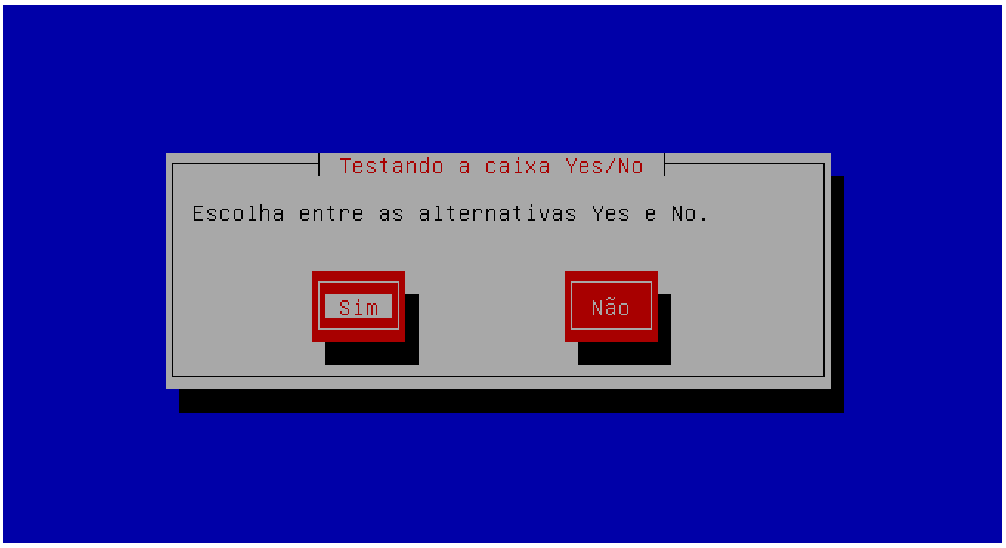


```
Caixa messagebox

Criando uma caixa de mensagem usando o
whiptail. Escolha OK para continuar.

Ok
```

2. Agora vamos criar uma caixa do tipo "Sim / Não" (Yes/No).
Sintaxe:
whiptail –title "<título da caixa de diálogo>" –yesno "<texto que será exibido>" <altura> <largura>

Exemplo:

```
#!/bin/bash
if whiptail --title "Testando a caixa Yes/No" --yesno "Escolha entre as
alternativas Yes e No." 10 50
then
   echo "Você escolheu Yes. O status de saída é $?."
else
   echo "Você escolheu No. O status de saída é  $?."
fi
```



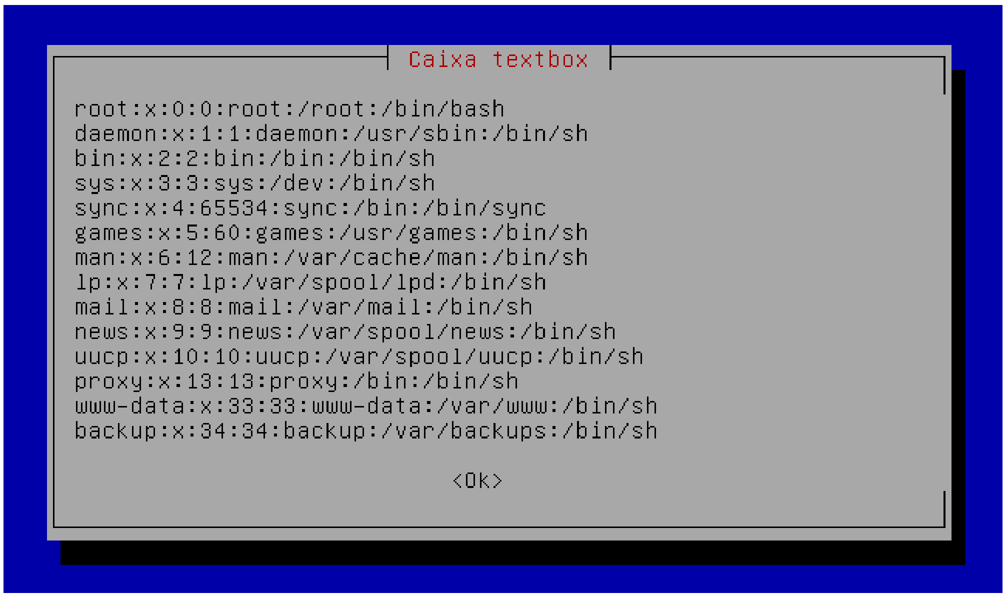
```
Testando a caixa Yes/No

Escolha entre as alternativas Yes e No.

Sim          Não
```

3. Trabalhando com o tamanho da caixa exibida:

```
#!/bin/bash
{
  for ((i=0 ; i<=70 ; i+=10)); do
     whiptail --title "Caixa de Teste" --msgbox "Testando o tamanho da caixa de 
texto. A largura atual é $i. Pressione OK para prosseguir." 20 $1
   done
}
```

4. Exibindo o conteúdo de um arquivo dentro de uma caixa do tipo textbox, com rolagem de tela (scroll):
Exemplo:

```
#!/bin/bash
whiptail --title "Caixa textbox" --textbox /etc/passwd 20 65 --scrolltext
```

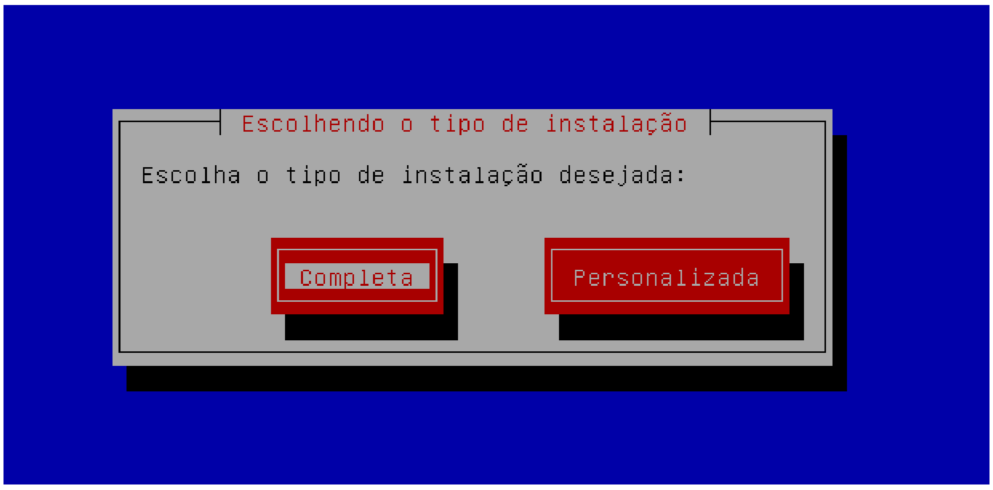


```
Caixa textbox

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh

                            <Ok>
```

5. Personalizando o texto dos botões da caixa Yes / No. Para isso usaremos as opções "-yes-button" e "–no-button":

Exemplo:

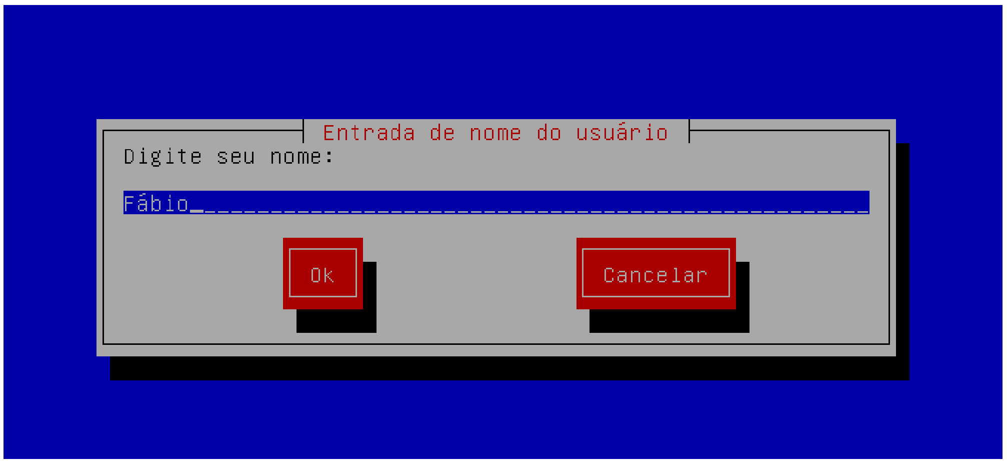
```
#!/bin/bash
if whiptail --title "Escolhendo o tipo de instalação" --yes-button "Completa" --
no-button "Personalizada"  --yesno "Escolha o tipo de instalação desejada" --fb 
10 50
then
   echo "Você escolheu a instalação Completa."
else
   echo "Você escolheu a instalação Personalizada."
fi
```


Escolhendo o tipo de instalação
Escolha o tipo de instalação desejada:
Completa
Personalizada

6. Trabalhando com entrada de texto do usuário: Input Box:
Sintaxe:
whiptail –title "<título do input box>" –inputbox "<texto que será exibido>" <altura> <largura> <texto padrão>

Exemplo:

```
#!/bin/bash
nome=$(whiptail --title "Entrada de nome do usuário" --inputbox "Digite seu
nome:" --fb 10 60 3>&1 1>&2 2>&3)
statussaida=$?
if [ $statussaida = 0 ]; then
   echo "O nome digitado foi: $nome"
else
   echo "Entrada cancelada pelo usuário."
fi
```

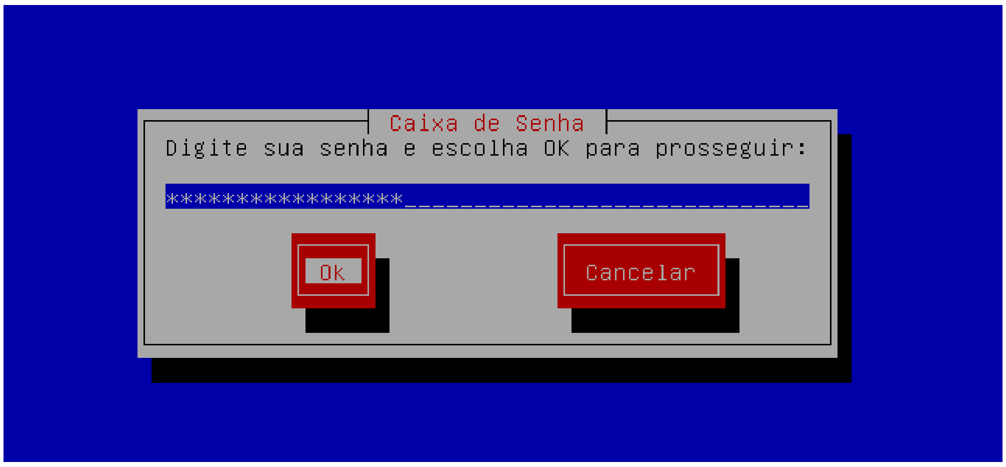


7. Criando uma caixa de senhas:
Sintaxe:
whiptail –title "<título da caixa de senha>" –passwordbox "<texto que será exibido>" altura> <largura>

Exemplo:

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

```
senha=$(whiptail --title "Caixa de Senha" --passwordbox "Digite sua senha e
escolha OK para continuar." --fb 10 50 3>&1 1>&2 2>&3)
status=$?
if [ $status = 0 ]; then
   echo "A senha digitada foi: $senha"
else
   echo "Entrada cancelada."
fi
```



```
Caixa de Senha
Digite sua senha e escolha OK para prosseguir:

*****************_______________________________

      <Ok>                 <Cancelar>
```

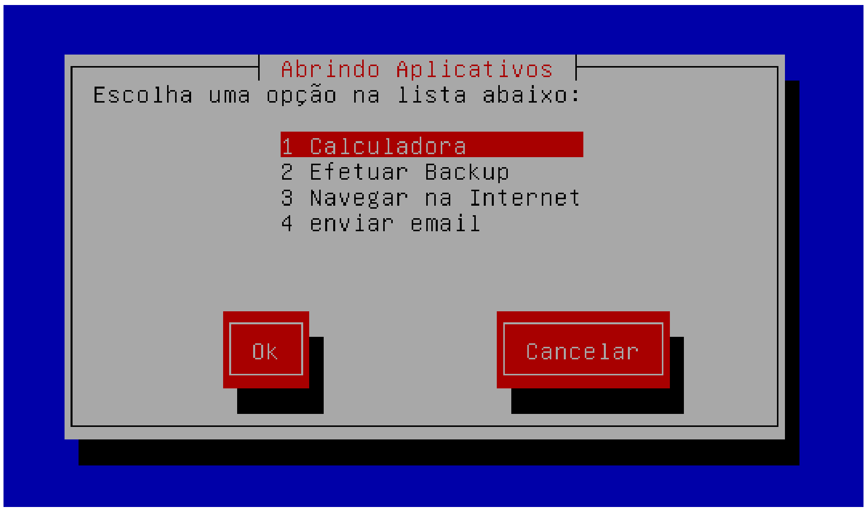
8. Criando uma caixa de menu. Com uma caixa de menu podemos pedir para o usuário selecionar um item dentre uma lista de itens.

Sintaxe:

whiptail –title "<título da caixa de menu>" –menu "<texto a ser exibido>" <altura> <largura> <altura da caixa de menu> [<tag> <item>] [<tag> <item>] [<tag> <item>]. . .

Exemplo:

```
#!/bin/bash
item=$(whiptail --title "Abrindo aplicativos" --menu "Escolha uma opção na lista
abaixo" --fb 15 50 4 \
"1" "Calculadora" \
"2" "Efetuar Backup" \
"3" "Navegar na Internet" \
"4" "Enviar email"  3>&1 1>&2 2>&3)

status=$?
if [ $status = 0 ]; then
   echo "Você escolheu a opção:" $item
else
   echo "Opção cancelada."
fi
```

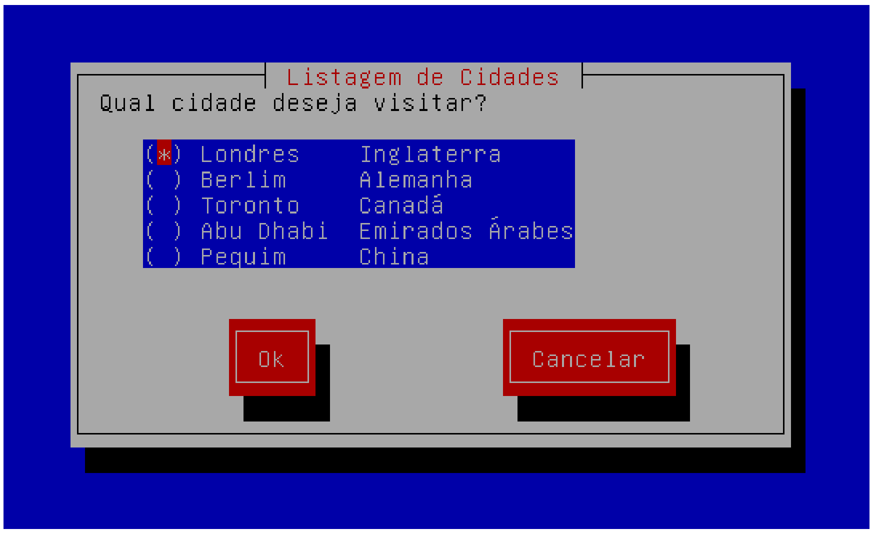


```
Abrindo Aplicativos
Escolha uma opção na lista abaixo:

                1 Calculadora
                2 Efetuar Backup
                3 Navegar na Internet
                4 enviar email

          <Ok>                  <Cancelar>
```

9. Criando uma radiolist para que um usuário possa escolher um único item entre uma lista de itens. Com uma radiolist é possível especificar um item selecionado por padrão:
Sintaxe:
whiptail –title "<título da lista>" –radiolist "<texto a ser exibido>" <altura> <largura> <altura da lista> [<tag> <item> <status>] [<tag> <item> <status>] . . .

Exemplo:

```
#!/bin/bash
cidade=$(whiptail --title "Listagem de Cidades" --radiolist \
"Qual cidade deseja visitar?" 15 50 5 \
"Londres" "Inglaterra" ON \
"Berlim" "Alemanha" OFF \
"Toronto" "Canadá" OFF \
"Abu Dhabi" "Emirados Árabes" OFF \
"Pequim" "China" OFF 3>&1 1>&2 2>&3)

status=$?
if [ $status = 0 ]
then
   echo "A cidade escolhida foi:" $cidade
else
   echo "Você não escolheu nenhuma cidade."
fi
```

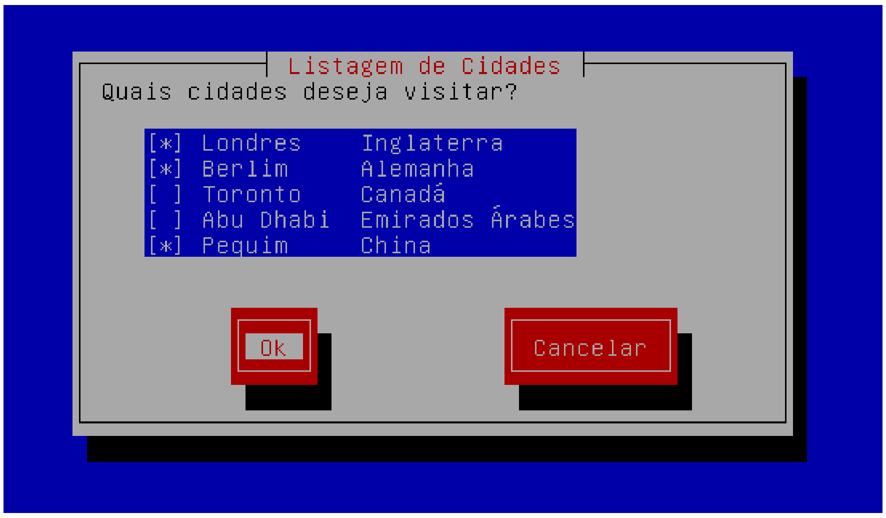


```
Listagem de Cidades
Qual cidade deseja visitar?

     (*) Londres     Inglaterra
     ( ) Berlim      Alemanha
     ( ) Toronto     Canadá
     ( ) Abu Dhabi   Emirados Árabes
     ( ) Pequim      China

          <Ok>                  <Cancelar>
```

10. Criando uma caixa de diálogo Checklist. Com uma checklist podemos mais de uma opção dentre uma lista de opções, diferentemente da radiolist, que só permite a escolha de uma opção.
Sintaxe:
whiptail –title "<título da checklist>" –checklist "<texto a ser exibido>" <altura> <largura> <altura da checklist> [<tag> <item> <status>] [<tag> <item> <status>] . . .

Exemplo:

```
#!/bin/bash
cidade=$(whiptail --title "Listagem de Cidades" --checklist --fb \
"Quais cidades deseja visitar?" 15 50 5 \
"Londres" "Inglaterra" ON \
"Berlim" "Alemanha" OFF \
"Toronto" "Canadá" OFF \
"Abu Dhabi" "Emirados Árabes" OFF \
"Pequim" "China" OFF 3>&1 1>&2 2>&3)

status=$?
if [ $status = 0 ]
then
   echo "As cidades escolhidas foram: " $cidade
else
   echo "Você não escolheu nenhuma cidade."
fi
```



```
Listagem de Cidades
Quais cidades deseja visitar?

[*] Londres     Inglaterra
[*] Berlim      Alemanha
[ ] Toronto     Canadá
[ ] Abu Dhabi   Emirados Árabes
[*] Pequim      China
```

11. Vamos trabalhar agora com caixas de diálogo sequenciais, onde o conteúdo de uma caixa de diálogo dependerá do que for selecionado na caixa anterior a ela:
Exemplo:

```
#!/bin/bash
pais=$(whiptail --title "Listagem de Países" --radiolist --fb \
"Qual país deseja visitar?" 15 50 5 \
"1" "Inglaterra" ON \
"2" "Alemanha" OFF 3>&1 1>&2 2>&3)

status=$?

if [ $status = 0 ]
then
 if [ $pais = 1 ]
 then
```

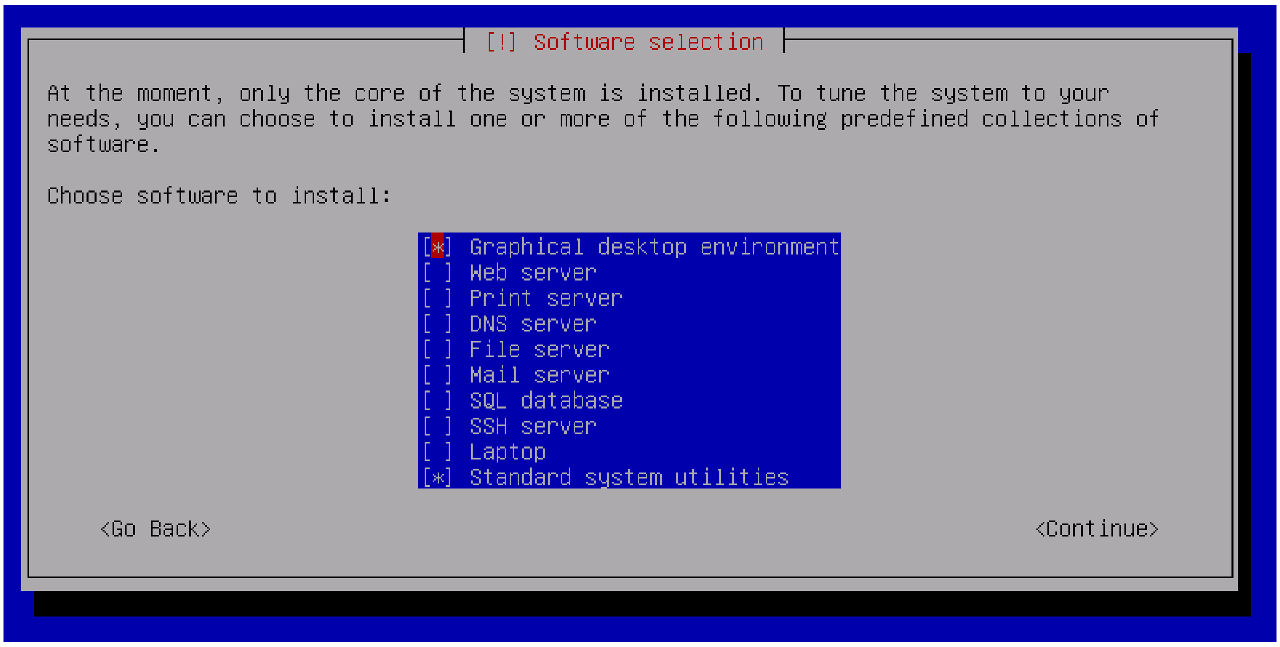
```
   cidade=$(whiptail --title "Cidades na inglaterra" --radiolist --fb \
   "Escolha a cidade na Inglaterra:" 15 50 5 \
   "Londres" "" ON \
   "Essex" "" OFF \
   "Liverpool" "" OFF 3>&1 1>&2 2>&3)
   status=$?
   if [ $status = 0 ]
    then
       echo "A cidade escolhida foi: " &cidade
   else
       echo "Opção cancelada pelo usuário"
   fi
else
   cidade=$(whiptail --title "Cidades na Alemanha" --radiolist --fb \
   "Escolha a cidade na Alemanha:" 15 50 5 \
   "Berlim" "" ON \
   "Frankfurt" "" OFF \
   "Dresden" "" OFF 3>&1 1>&2 2>&3)
   status=$?
   if [ $status = 0 ]
    then
       echo "A cidade escolhida foi: " &cidade
   else
       echo "Opção cancelada pelo usuário"
   fi
fi
else
 "Nenhum país selecionado. Entrada cancelada"
fi
```

https://www.youtube.com/watch?v=wFO3KgXPkIU

How to create dialog boxes in an interactive shell script

Last updated on September 28, 2020 by Dan Nanni

When you install new software in the terminal environment, you may often see informative dialog boxes popping up, accepting your input. The type of dialog boxes ranges from simple yes/no dialog to input box, password box, checklist, menu, and so on. The advantage of using such user-friendly dialog boxes is obvious as they can guide you to enter necessary information in an intuitive fashion.


```
[!] Software selection

At the moment, only the core of the system is installed. To tune the system to your
needs, you can choose to install one or more of the following predefined collections of
software.

Choose software to install:

                         [*] Graphical desktop environment
                         [ ] Web server
                         [ ] Print server
                         [ ] DNS server
                         [ ] File server
                         [ ] Mail server
                         [ ] SQL database
                         [ ] SSH server
                         [ ] Laptop
                         [*] Standard system utilities

    <Go Back>                                                          <Continue>
```

When you write an interactive shell script, you can actually use such dialog boxes to take user's input. Pre-installed on all modern Linux distributions, a program called `whiptail` can streamline the process of creating terminal-based dialogs and message boxes inside a shell script, similar to how Zenity or Xdialog codes a GUI for scripts.

In this tutorial, I describe **how to create user-friendly dialog boxes in a shell script by using `whiptail`**. I also show Bash code snippets of various dialog boxes supported by `whiptail`.

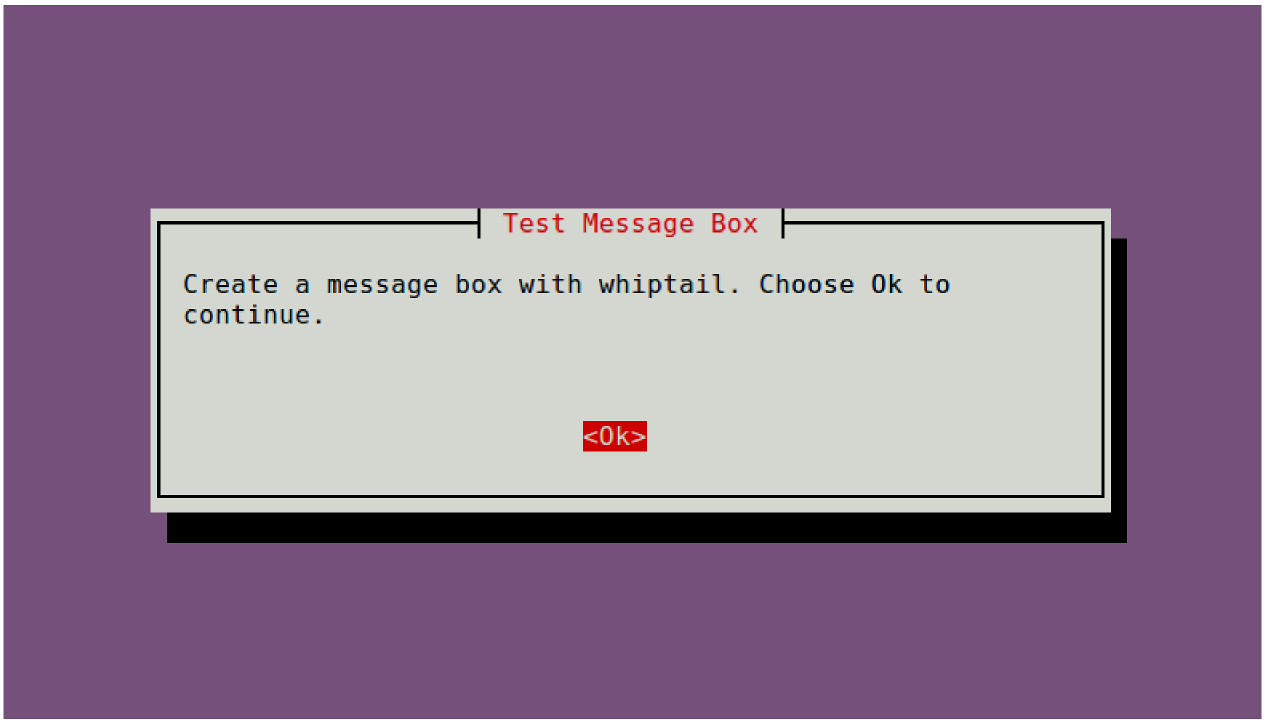
Create a Message Box

A message box shows any arbitrary text message with a confirmation button to continue.

```
whiptail --title "<message box title>" --msgbox "<text to show>" <height>
<width>
```

Example:

```
#!/bin/bash
whiptail --title "Test Message Box" --msgbox "Create a message box with
whiptail. Choose Ok to continue." 10 60
```



```
Test Message Box
Create a message box with whiptail. Choose Ok to
continue.
```

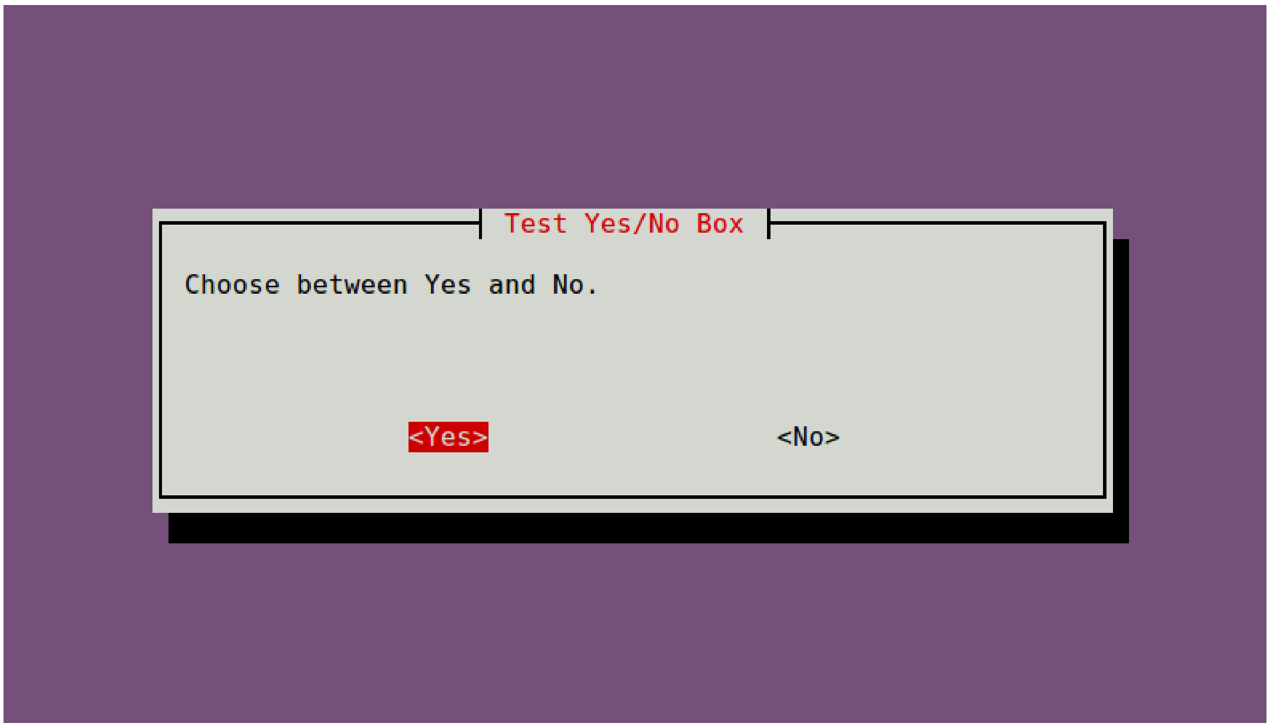
Create a Yes/No Box

One common user input is Yes or No. This is when a Yes/No dialog box can be used.

```
whiptail --title "<dialog box title>" --yesno "<text to show>" <height> <width>
```

Example:

```
#!/bin/bash
if (whiptail --title "Test Yes/No Box" --yesno "Choose between Yes and No." 10
60) then
    echo "You chose Yes. Exit status was $?."
else
    echo "You chose No. Exit status was $?."
fi
```

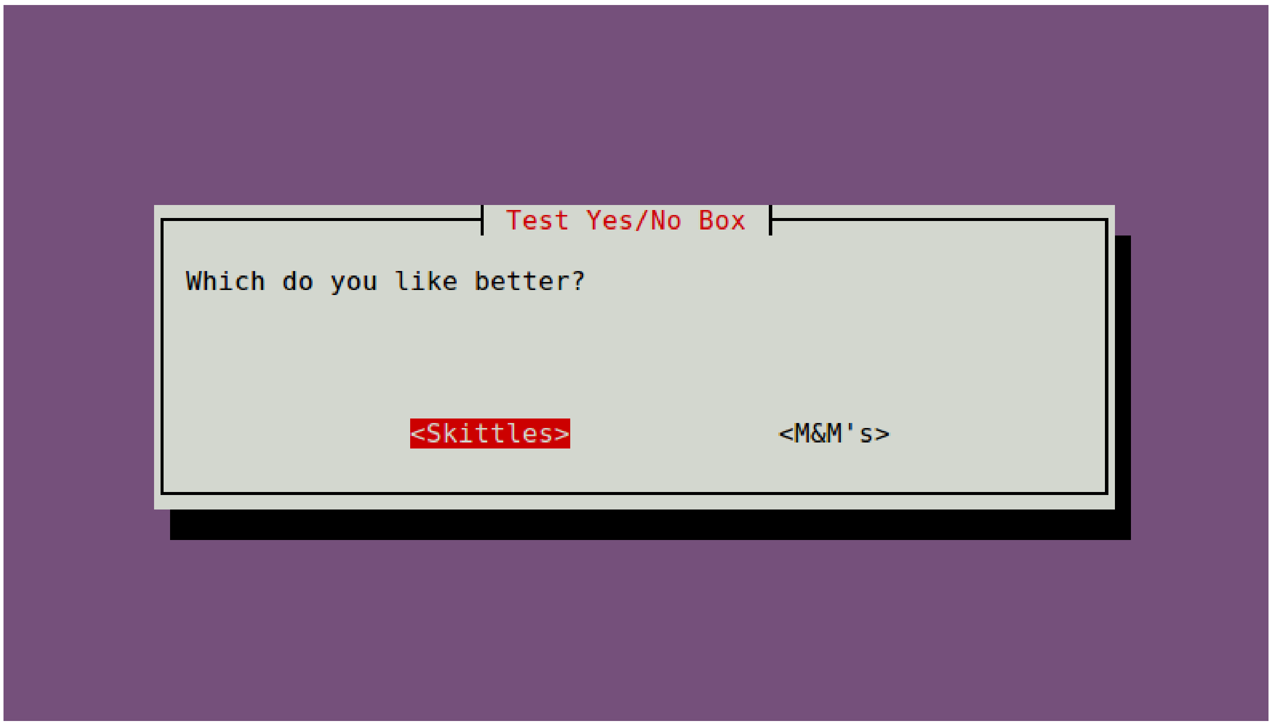


Choose between Yes and No.

Optionally, you can customize the text for Yes and No buttons with `--yes-button` and `--no-button` options.

Example:

```
#!/bin/bash
if (whiptail --title "Test Yes/No Box" --yes-button "Skittles" --no-button
"M&M's"  --yesno "Which do you like better?" 10 60) then
    echo "You chose Skittles Exit status was $?."
else
    echo "You chose M&M's. Exit status was $?."
fi
```



Which do you like better?

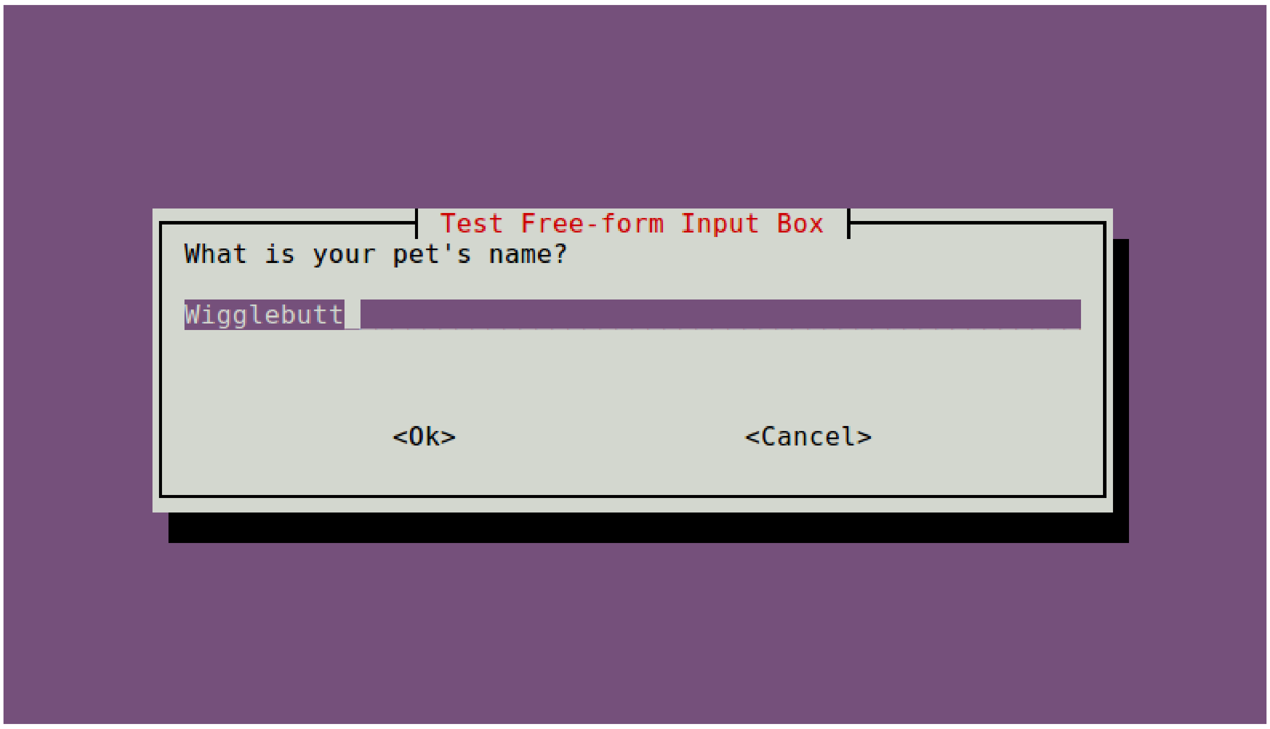
Create a Free-form Input Box

If you want to take any arbitrary text input from a user, you can use an input box.

```
whiptail --title "<input box title>" --inputbox "<text to show>" <height>
<width> <default-text>
```

Example:

```
#!/bin/bash
PET=$(whiptail --title "Test Free-form Input Box" --inputbox "What is your pet's
name?" 10 60 Wigglebutt 3>&1 1>&2 2>&3)

exitstatus=$?
if [ $exitstatus = 0 ]; then
    echo "Your pet name is:" $PET
else
    echo "You chose Cancel."
fi
```



```
Test Free-form Input Box
What is your pet's name?

Wigglebutt

<Ok>                <Cancel>
```

Create a Password Box

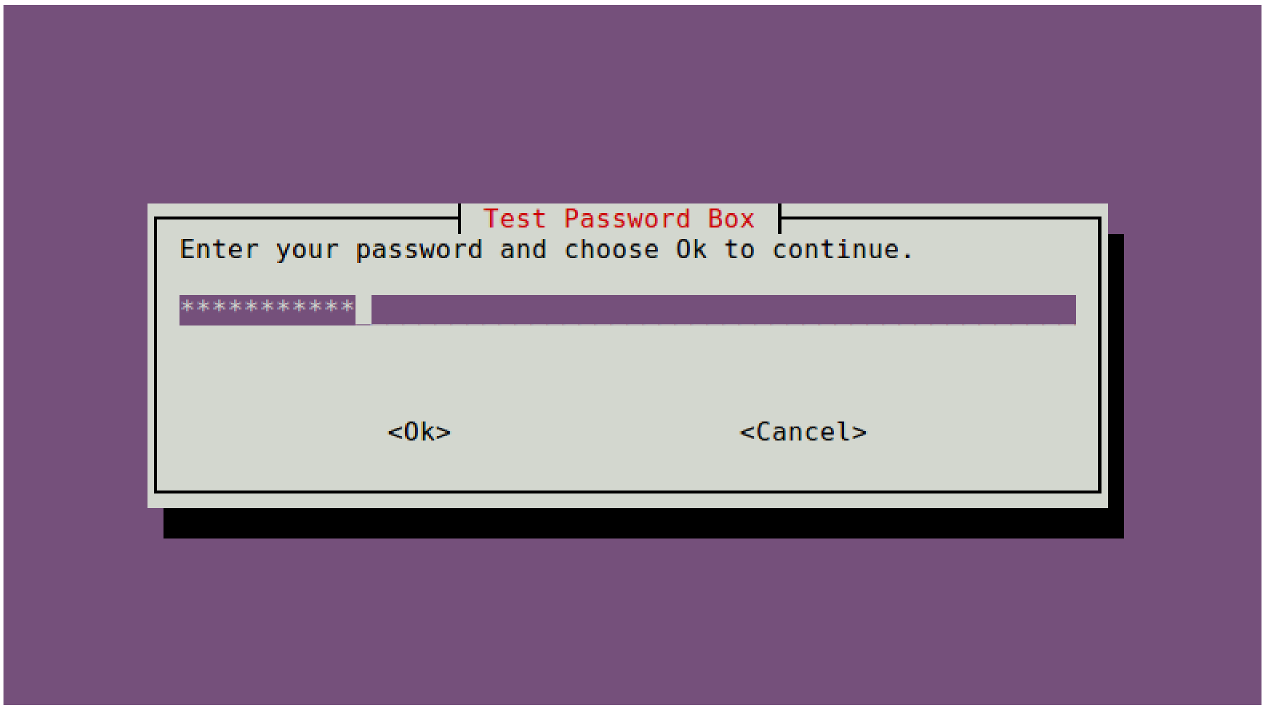
A password box is useful when you want to take a sensitive input from a user.

```
whiptail --title "<password box title>" --passwordbox "<text to show>" <height>
<width>
```

Example:

```
#!/bin/bash
PASSWORD=$(whiptail --title "Test Password Box" --passwordbox "Enter your
password and choose Ok to continue." 10 60 3>&1 1>&2 2>&3)

exitstatus=$?
if [ $exitstatus = 0 ]; then
```

```
    echo "Your password is:" $PASSWORD
else
    echo "You chose Cancel."
fi
```



Test Password Box

Enter your password and choose Ok to continue.

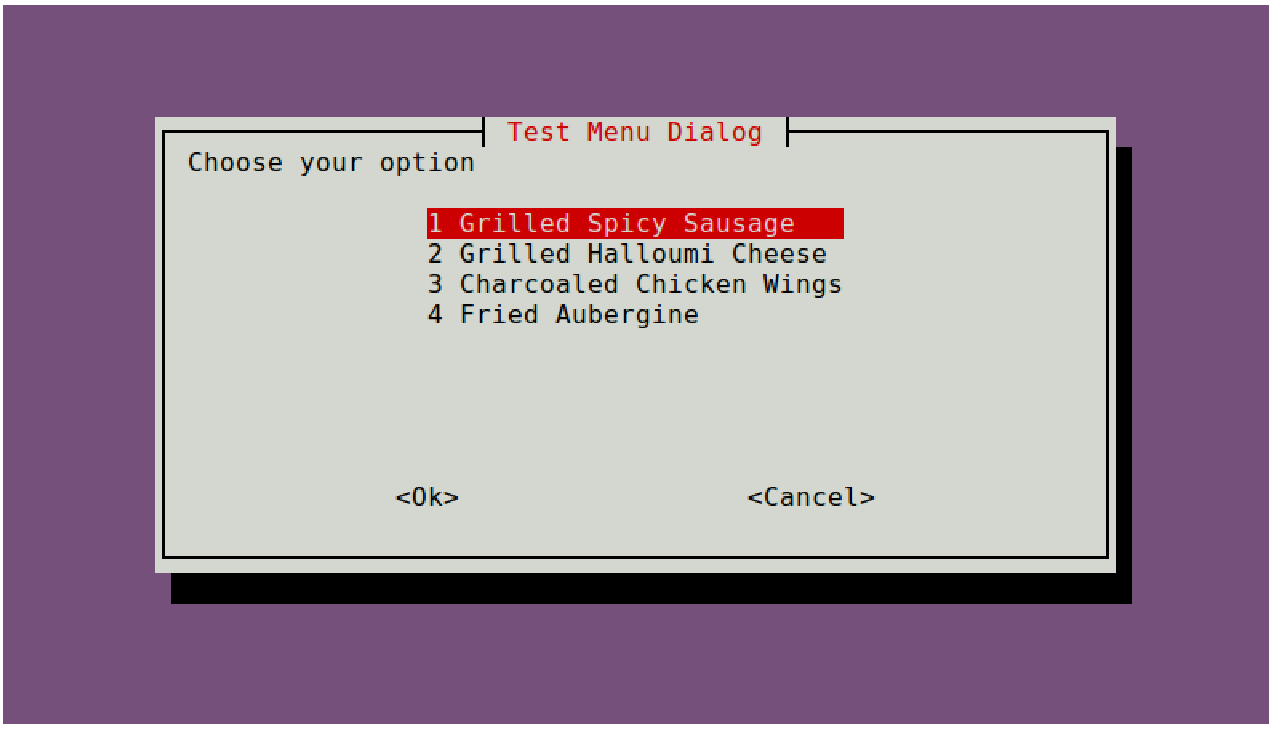
Create a Menu Box

When you want to ask a user to choose one among any arbitrary number of choices, you can use a menu box.

```
whiptail --title "<menu title>" --menu "<text to show>" <height> <width> <menu
height> [ <tag> <item> ] . . .
```

Example:

```
#!/bin/bash
OPTION=$(whiptail --title "Test Menu Dialog" --menu "Choose your option" 15 60 4
"1" "Grilled Spicy Sausage"
"2" "Grilled Halloumi Cheese"
"3" "Charcoaled Chicken Wings"
"4" "Fried Aubergine"  3>&1 1>&2 2>&3)

exitstatus=$?
if [ $exitstatus = 0 ]; then
    echo "Your chosen option:" $OPTION
else
    echo "You chose Cancel."
fi
```



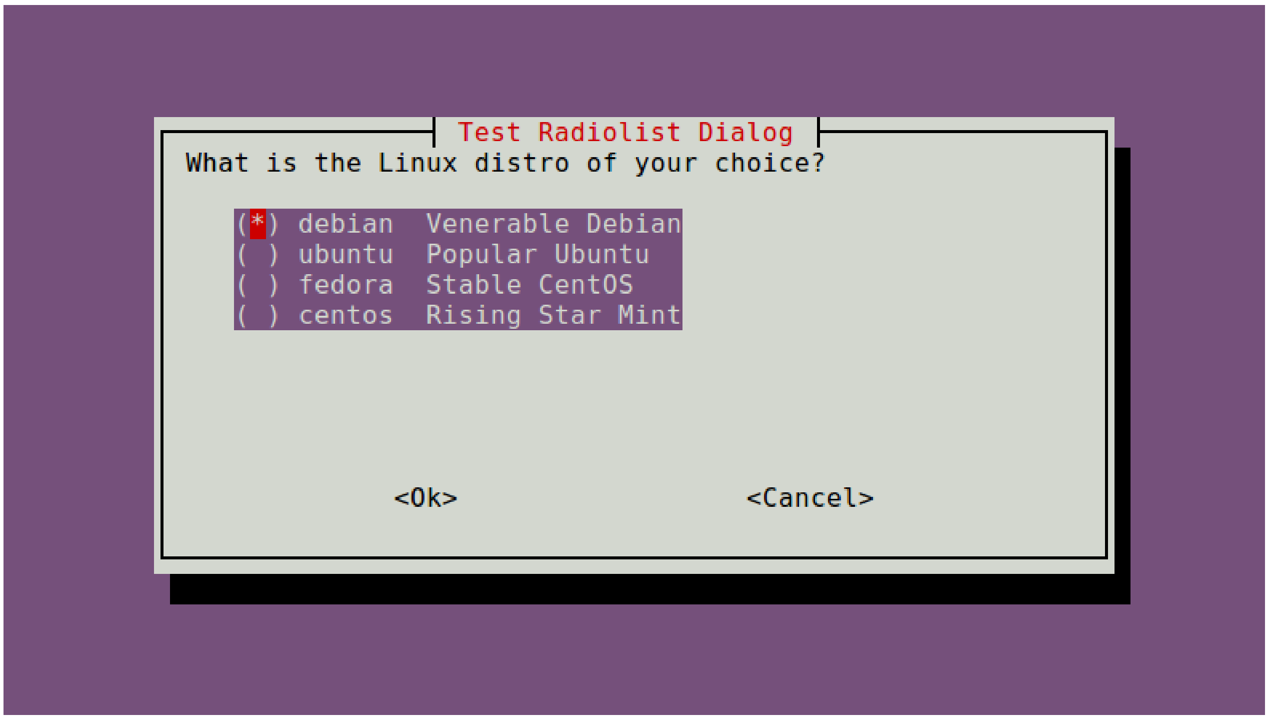
Create a Radiolist Dialog

A radiolist box is similar to a menu box in the sense that you can choose only option among a list of available options. Unlike a menu box, however, you can indicate which option is selected by default by specifying its status.

```
whiptail --title "<radiolist title>" --radiolist "<text to show>" <height>
<width> <list height> [ <tag> <item> <status> ] . . .
```

Example:

```
#!/bin/bash
DISTROS=$(whiptail --title "Test Checklist Dialog" --radiolist
"What is the Linux distro of your choice?" 15 60 4
"debian" "Venerable Debian" ON
"ubuntu" "Popular Ubuntu" OFF
"centos" "Stable CentOS" OFF
"mint" "Rising Star Mint" OFF 3>&1 1>&2 2>&3)

exitstatus=$?
if [ $exitstatus = 0 ]; then
    echo "The chosen distro is:" $DISTROS
else
    echo "You chose Cancel."
fi
```



```
┤ Test Radiolist Dialog ├
What is the Linux distro of your choice?

    (*) debian   Venerable Debian
    ( ) ubuntu   Popular Ubuntu
    ( ) fedora   Stable CentOS
    ( ) centos   Rising Star Mint


             <Ok>                  <Cancel>
```

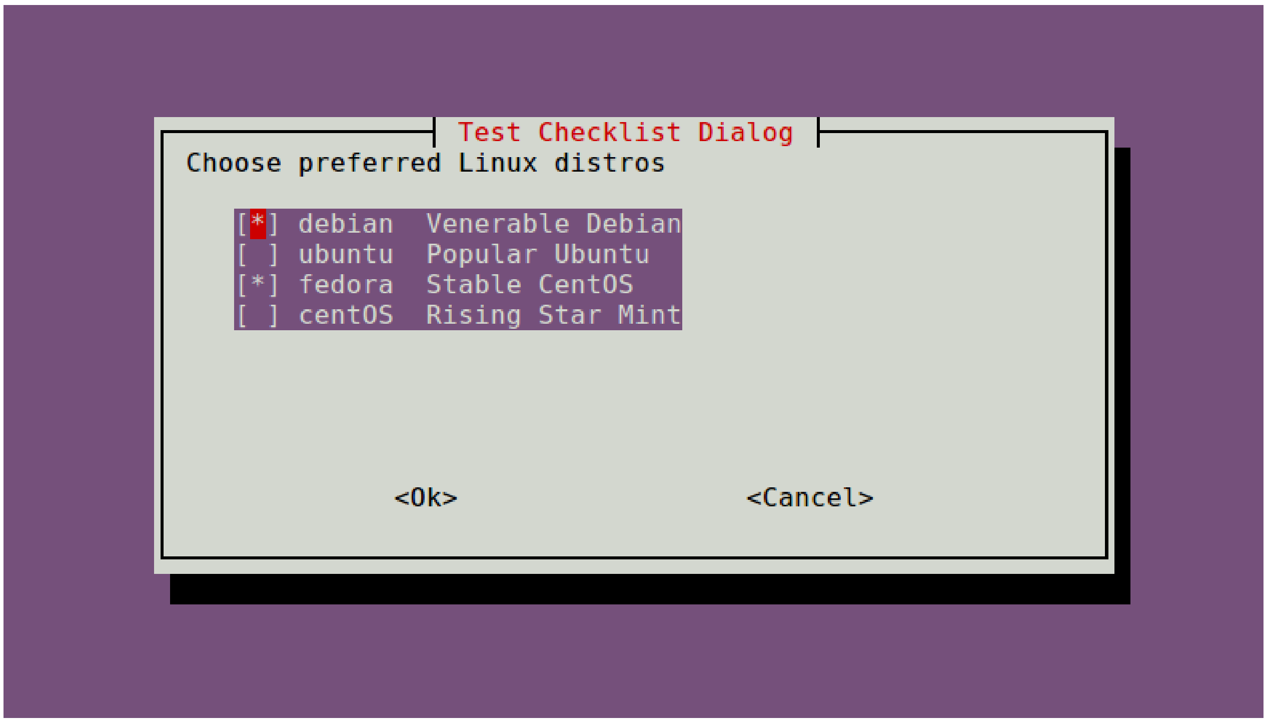
Create a Checklist Dialog

A checklist dialog is useful when you want to ask a user to choose more than one option among a list of options, which is in contrast to a radiolist box which allows only one selection.

```
whiptail --title "<checklist title>" --checklist "<text to show>" <height>
<width> <list height> [ <tag> <item> <status> ] . . .
```

Example:

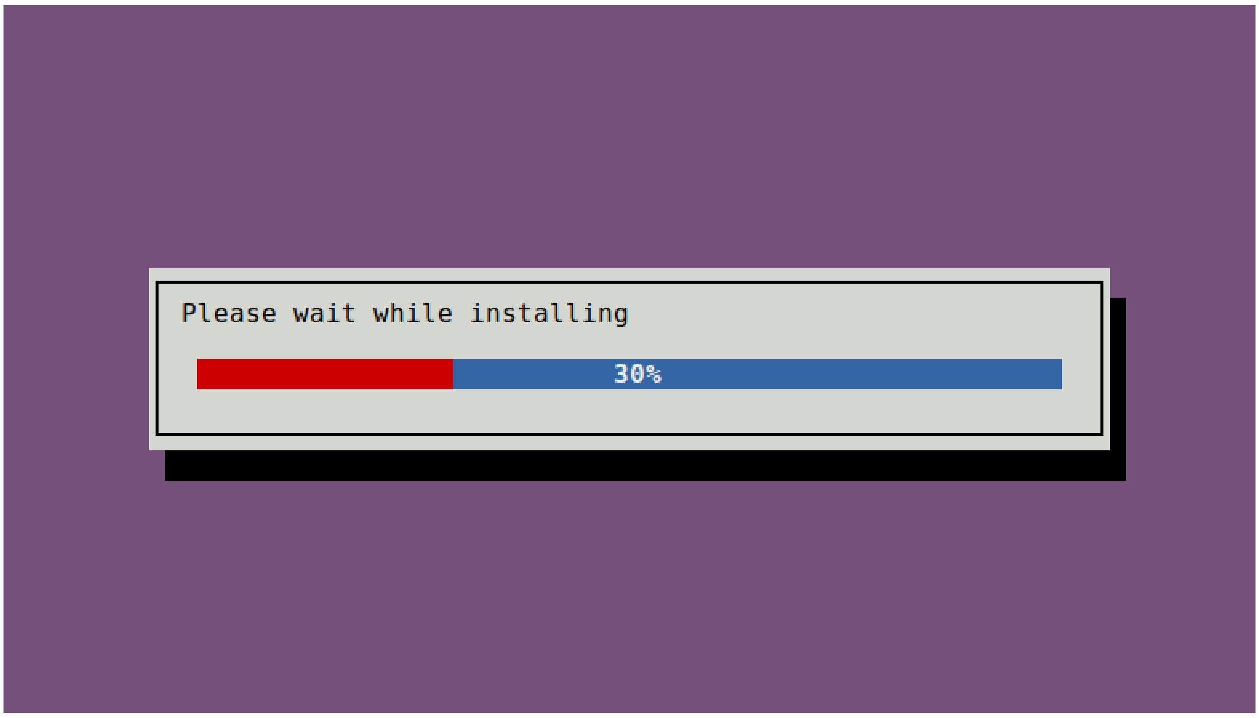
```
#!/bin/bash
DISTROS=$(whiptail --title "Test Checklist Dialog" --checklist
"Choose preferred Linux distros" 15 60 4
"debian" "Venerable Debian" ON
"ubuntu" "Popular Ubuntu" OFF
"centos" "Stable CentOS" ON
"mint" "Rising Star Mint" OFF 3>&1 1>&2 2>&3)

exitstatus=$?
if [ $exitstatus = 0 ]; then
    echo "Your favorite distros are:" $DISTROS
else
    echo "You chose Cancel."
fi
```


```
Test Checklist Dialog
Choose preferred Linux distros

[*] debian   Venerable Debian
[ ] ubuntu   Popular Ubuntu
[*] fedora   Stable CentOS
[ ] centOS   Rising Star Mint

<Ok>                 <Cancel>
```

Create a Progress Bar

Another user-friendly dialog box is a progress bar. `whiptail` reads from standard input a percentage number (`0` to `100`) and displays a meter inside a gauge box accordingly.

```
whiptail --gauge "<test to show>" <height> <width> <inital percent>

#!/bin/bash
{
    for ((i = 0 ; i <= 100 ; i+=20)); do
        sleep 1
        echo $i
    done
} | whiptail --gauge "Please wait while installing" 6 60 0
```



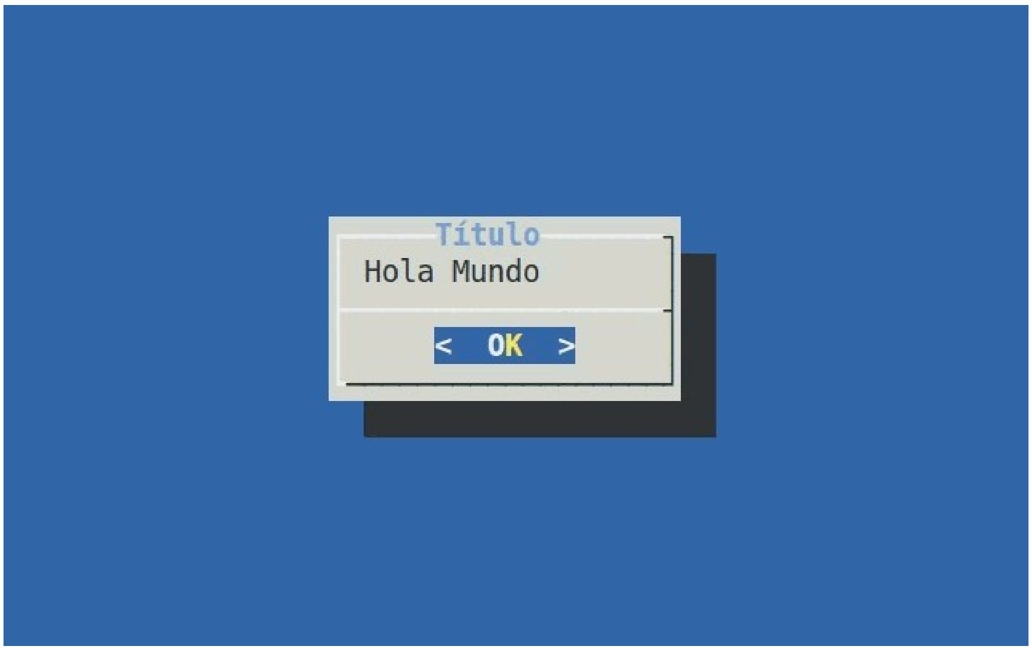
```
Please wait while installing

30%
```

By now, you must see how easy it is to create useful dialog boxes in an interactive shell script. Next time you need to write a shell script for someone, why don't you try `whiptail` and impress him or her? :-)

If you find this tutorial helpful, I recommend you check out the series of bash shell scripting tutorials provided by Xmodulo.

Dialog: How to create menus in your scripts

Written by Ignacio Alba Obaya on . 4 Comments



Dialog is a command of GNU / Linux that allows you to create dialog boxes in the terminal for you to use in your scripts programming, I'll show you how to use this command.

For the realization of this article I command basadoen the manual and on the following websites give my sincere thanks to the creators.

http://ovtoaster.com/scripts-en-linux-con-estilo/
http://linuxgazette.net/101/sunil.html

The colors in applications and systems

written texts in the command line: Blue
Output in the command line: Green
File names and file content: Brown

Installing dialog

To use this tool in linux first thing you have to do is install

Ubuntu

```
sudo apt-get install dialog
```

Debian

```
apt-get install dialog
```

Red hat

```
yum install dialog
```

Suse

```
zypper install dialog
```

Syntax

For boxes with only one option.

So much <width-of-box> <top-of-box> is represented by numerical values if we 0 They conform to content.

```
dialog <options-common> <box options> <width-of-box> <top-of-box>
```

Example:

```
dialog --title "Title"  --yesno "You like the Red color?" 0 0
```

For cases with multiple options

It is only for type boxes checklist, menu or radiolist. If we 0 in <number-of-options-visible> it automatically adjusts to the number of options.

```
dialog <options-common> <box options> <width-of-box> <top-of-box> <number-of-
options-visible>.
```

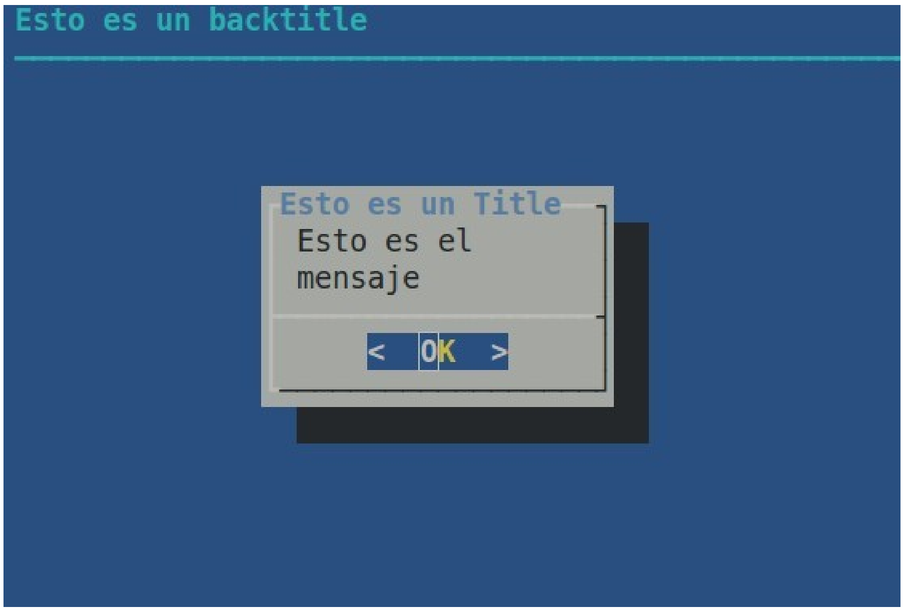
Example:

```
dialog --checklist "Choose the options you want:" 0 0 0  1 cheese on 2 "Mustard"
on  3 anchovies off
```

common options

They are options that are present in most types of dialog boxes "dialog" the most common are the following:

Titles



```
Esto es el
mensaje
```

–title

It is used to put the dialog title.

–backtitle

It is the title of the window background.

Change text buttons

–yes-label

Change the word YES in the dialog boxes for your wish.

–ok–label

Change the word OK in the dialog boxes for your wish.

–cancel-label

Change the word CANCEL of the dialog boxes for your wish.

–exit-label

Change the word EXIT of the dialog boxes for your wish.

–no-label

Change the word NO in the dialog boxes for your wish.

–nook

Remove the OK button of the dialog box you need to press enter to enter.

Command Output

These variables allow you to decide where will the choices made in the command will by default stderr if we can get him to screen.

–stderr

Command output to stderr if we choose this option can get the output to a file by adding at the end of the line of dialog:

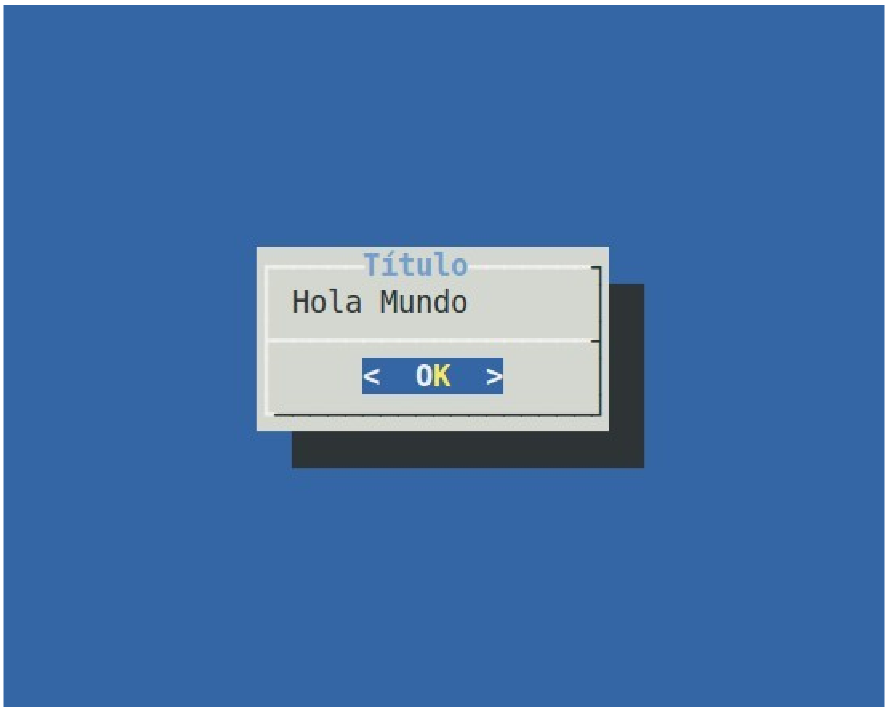
```
2><where to save the output file>
```

–stdout

With this option we will display the command output I use when I want to store in a variable command but do not want to be written to a temporary file, later in the examples we will see how.

Options box

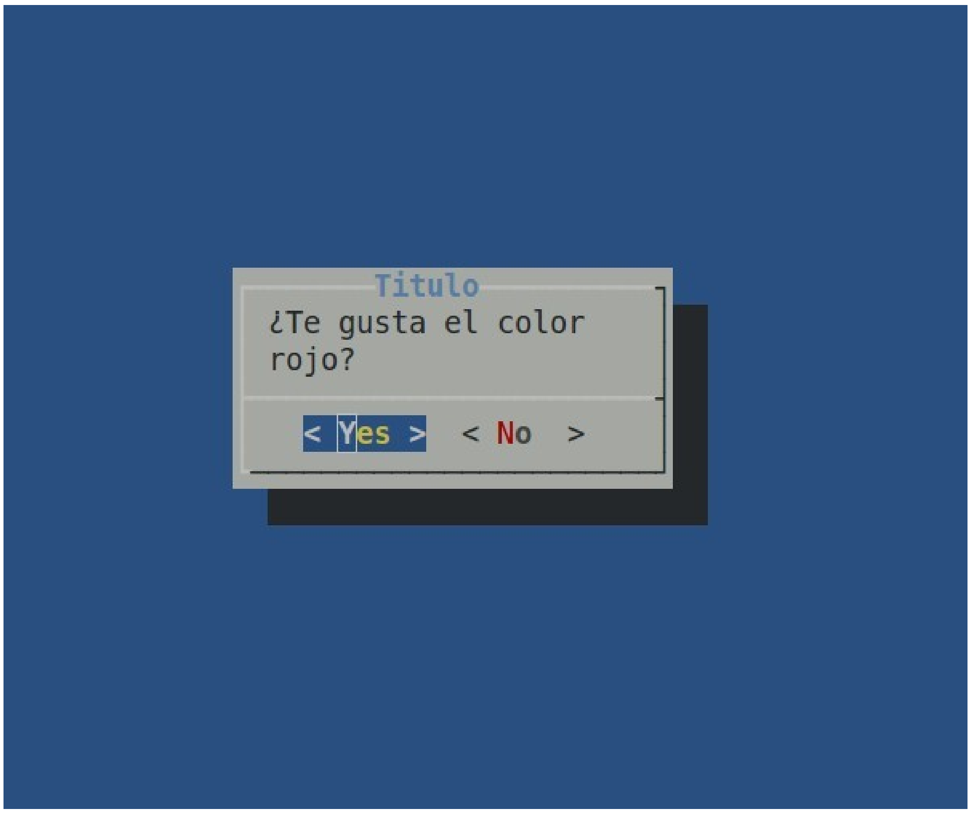
Message Box (–msgbox)



It serves to put a message on the screen until the user press enter.

```
dialog --title "Title" --msgbox "Hello World" 0 0
```

Menu YES / NO (–yesno)

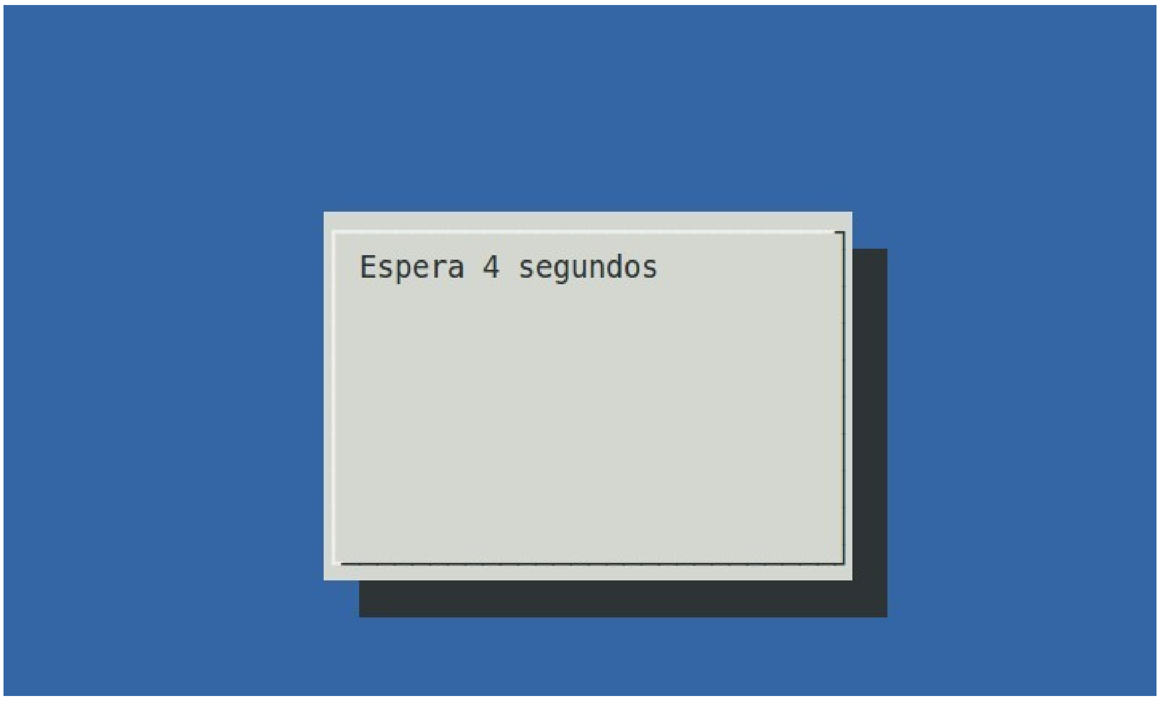


```
Titulo
¿Te gusta el color
rojo?
```

It serves to answer questions with answer YES / NO if the answer is YES returns 0 If the answer is NO returns 1

```
dialog --title "Title"  --yesno "You like the Red color?" 0 0
```

Infobox (–infobox)

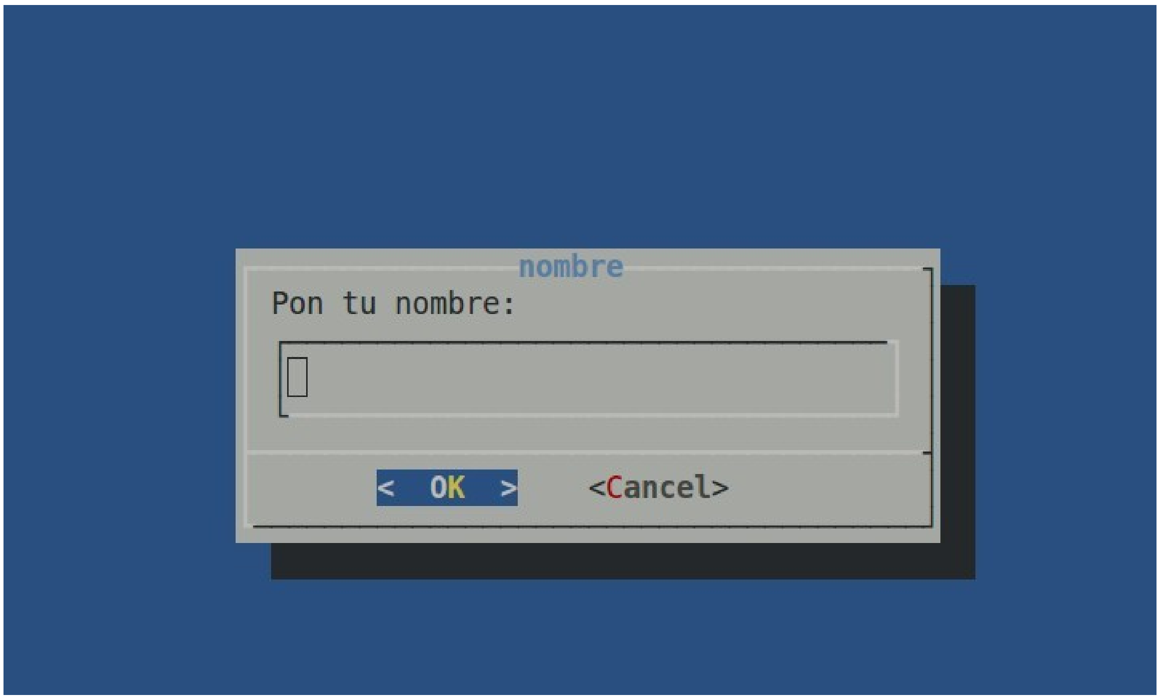


```
Espera 4 segundos
```

This box displays information we want is desirable to add a sleep (waiting x seconds to continue command) to view it.

```
dialog --infobox "Wait 4 seconds" 0 0 ; sleep 4
```

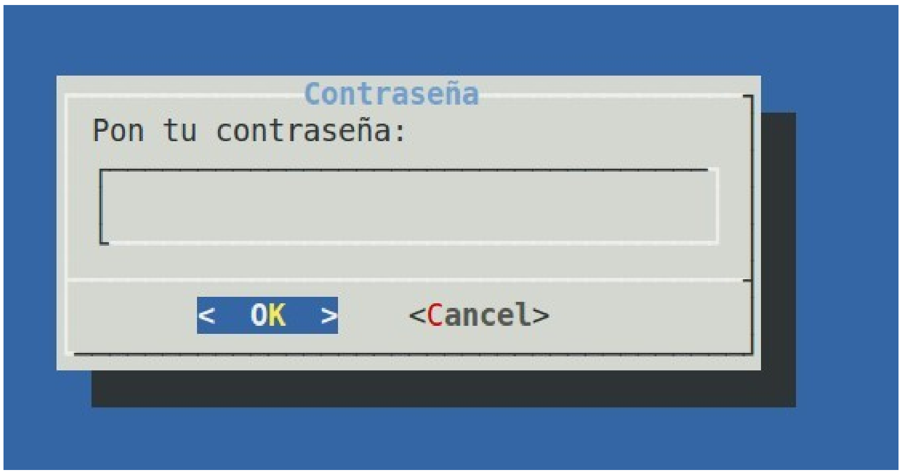
Headbox (–inputbox)



It is used to collect data.

```
dialog --title "name" --inputbox "Put your name:" 0 0
```

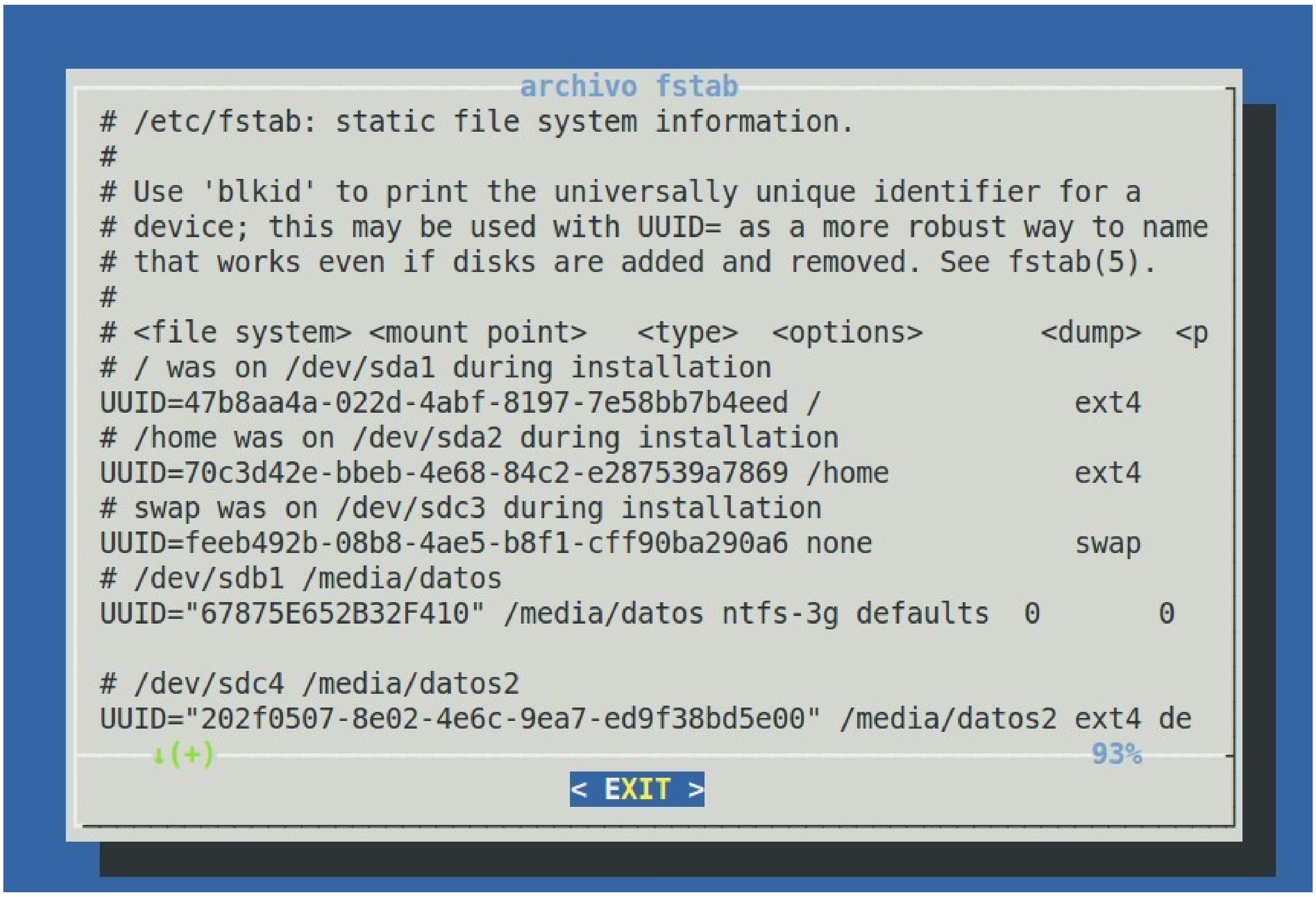
Password box (–PasswordBox)



It is equal to the input box but the characters typed are not displayed.

```
dialog --title "Password" --PasswordBox "Put your password:" 0 0
```

Box (–textbox)

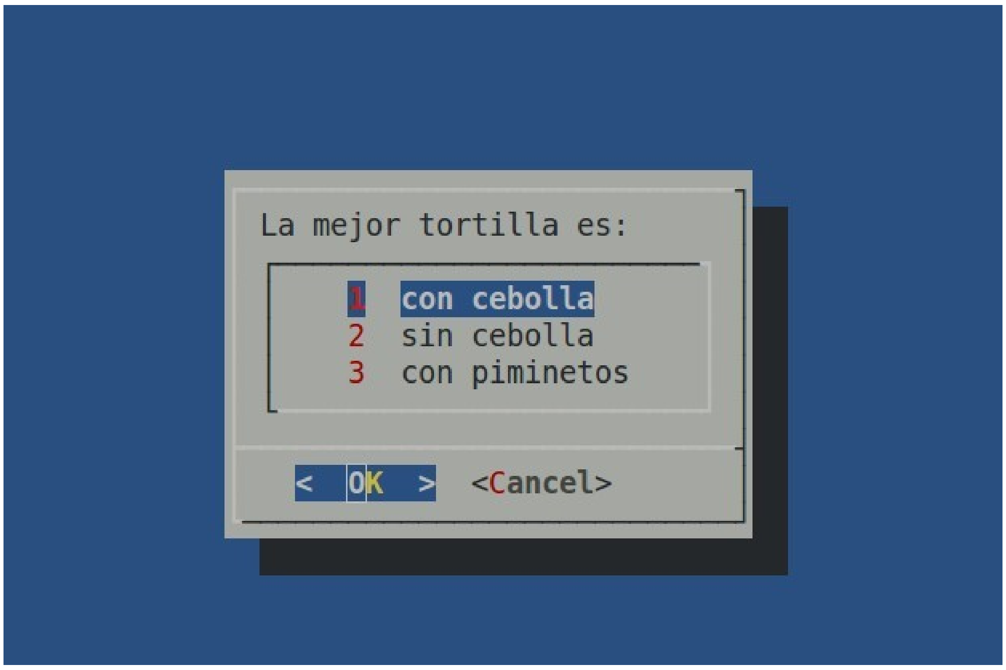


```
                      archivo fstab
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <p
# / was on /dev/sda1 during installation
UUID=47b8aa4a-022d-4abf-8197-7e58bb7b4eed /               ext4
# /home was on /dev/sda2 during installation
UUID=70c3d42e-bbeb-4e68-84c2-e287539a7869 /home           ext4
# swap was on /dev/sdc3 during installation
UUID=feeb492b-08b8-4ae5-b8f1-cff90ba290a6 none            swap
# /dev/sdb1 /media/datos
UUID="67875E652B32F410" /media/datos ntfs-3g defaults  0       0

# /dev/sdc4 /media/datos2
UUID="202f0507-8e02-4e6c-9ea7-ed9f38bd5e00" /media/datos2 ext4 de
    ↓(+)                                                      93%
                           < EXIT >
```

The box is a file viewer and shows us the file you indicate.

```
dialog --title "fstab file" --textbox /etc/fstab 0 0
```

Box menu (–menu)

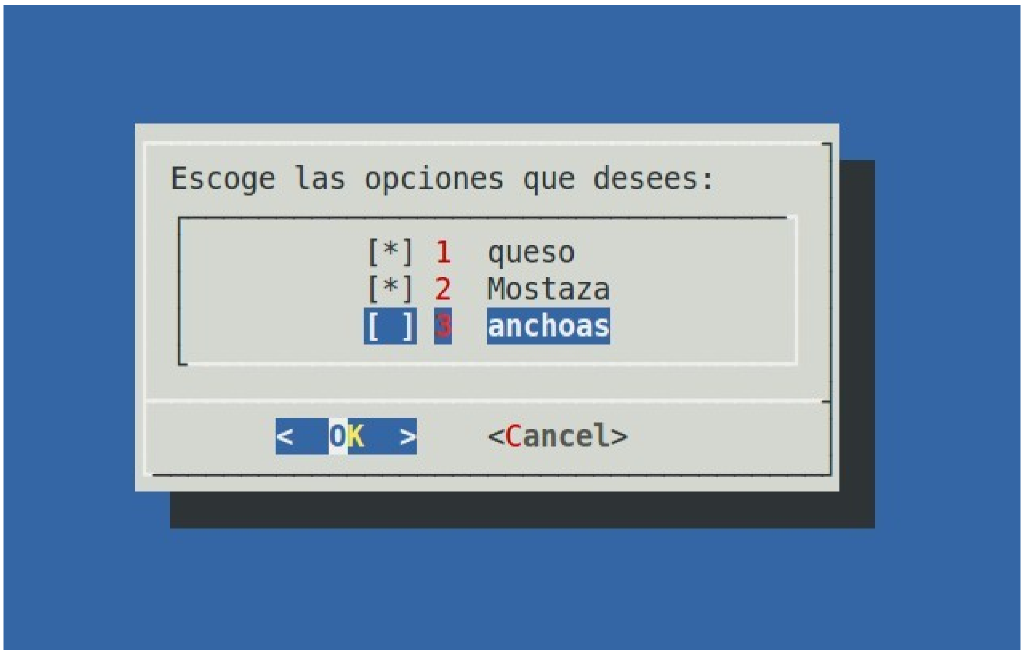


```
La mejor tortilla es:

      1   con cebolla
      2   sin cebolla
      3   con piminetos

  <  OK  >   <Cancel>
```

It allows us to choose an option among several numbered options

```
dialog --menu "The best tortilla is:" 0 0 0 1 "with onions" 2 "without onion" 3
"with piminetos"
```

Box checklist (–checklist)



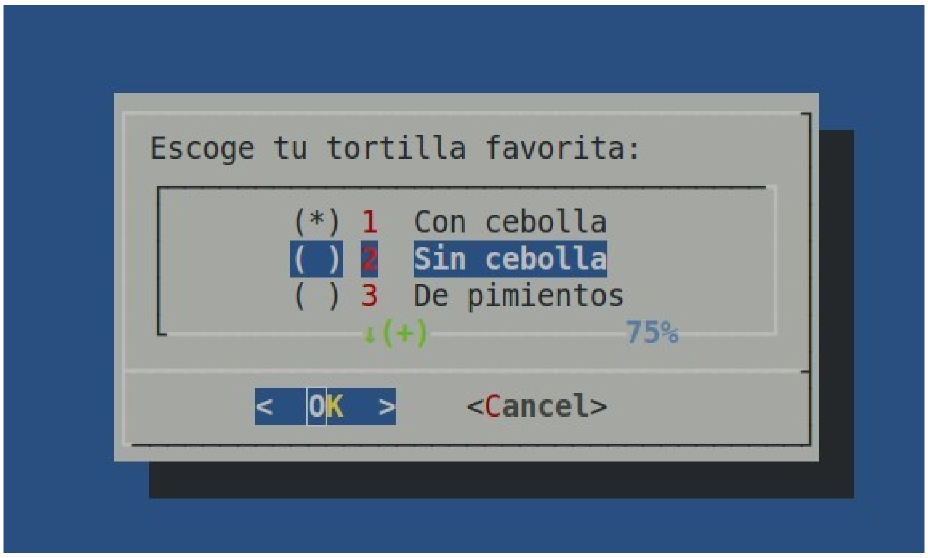
```
Escoge las opciones que desees:

        [*] 1  queso
        [*] 2  Mostaza
        [ ] 3  anchoas

     <  OK  >       <Cancel>
```

It is a menu with several options that we can choose several. The options put in **on** They will be lit and to put in **off** deleted.

```
dialog --checklist "Choose the options you want:" 0 0 0  1 cheese on 2 "Mustard"
on  3 anchovies off
```

A special mention within checklist for option –separate-output that returns us a line for each selection made very useful for running different commands for each selection made.

Box radiolist (–radiolist)

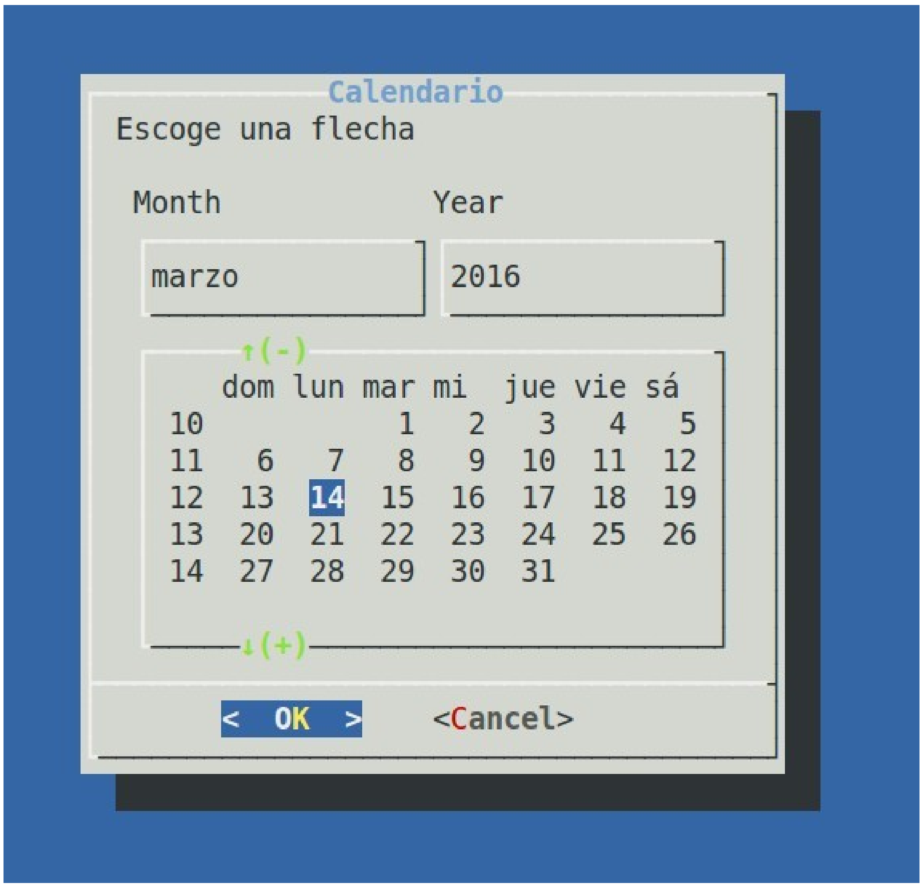


```
Escoge tu tortilla favorita:

        (*) 1  Con cebolla
        ( ) 2  Sin cebolla
        ( ) 3  De pimientos
          ↓(+)              75%

     <  OK  >       <Cancel>
```

It is a menu with several options that we can choose only one.

```
dialog --radiolist "Choose your favorite omelette:" 0 0 0 1 "with onion" off  2
"Without onion" on 3 "peppers" off 4 "asparagus" off
```

Calendar box (–calendar)



```
Calendario
Escoge una flecha

 Month              Year
 marzo              2016

        ↑(-)
    dom lun mar mi  jue vie sá
 10              1   2   3   4   5
 11   6   7   8   9  10  11  12
 12  13  14  15  16  17  18  19
 13  20  21  22  23  24  25  26
 14  27  28  29  30  31
        ↓(+)

      <  OK  >      <Cancel>
```

It is a calendar where we can choose a date.

```
dialog --title "Calendar" --calendar "Choose an arrow" 0 0
```

Other options box

In addition to the options box that we have seen there are other options that might be interesting to use to know them do not hesitate to come to the aid.

```
man dialog
```

How to use the data obtained with dialog

To use a script to data compiled by the dialog boxes there varaias options dialog I've found, but surely occur to you it will not hesitate to comunicarmelas more and add in the post.

Redirect output to a file dialog.

We can store the output of command to a file dialog, usually we will put in the / tmp and delete after use, then I put a script with an example explained.

```
#!/bin/bash
# By Ignacio Alba Obaya
# https://aplicacionesysistemas.com
# run the dialog box by ending 2>/tmp / nombre.tmp. $$
# store it in a file name entered.
# remember that 2> redirects the error output to a file.
dialog --title "name" --inputbox "Put your name:" 0 0 2>/tmp / nombre.tmp. $$
# delete the screen
clear
# show the stored name
cat /tmp/nombre.tmp.$$
# delete the file named
rm -f /tmp/nombre.tmp.$$
# We make a line break so that we are not amass the prompt
echo -e "\n"
```

Store the answer in a variable.

To store the result we will do this by creating a function and calling it from a variable.

```
#!/bin/bash
# By Ignacio Alba Obaya
# https://aplicacionesysistemas.com
# Store the result in a variable dialog
# We create the function fundialog
fundialog = ${fundialog=dialog}
# We create a variable with the output date dialog
# redireccinando with output dialog --stdout
# to standard output, Note that the function is between
# accents of the key [ if not it does not work.
fecha=`$fundialog --stdout --title "Calendar" --calendar "Choose a date" 0 0`
# Show captured date
echo $ date
```

How to manage outputs checklist

We also need to manage multiple output options priate have a checklist so that each option trigger different events. A hobble to do this is as follows:

```
#!/bin/bash
# By Ignacio Alba Obaya
# https://aplicacionesysistemas.com

# We created the varaible funcheck in which we store
# order dialog with the option --separate-output
funcheck =(dialog --separate-output --checklist "Select the groups they belong:"
0 0 0)

# We define the options on the screen
# appear lit the we have put on.
options =(1 "option 1" on
 2 "option 2" off
 3 "option 3" off
 4 "option 4" off
 5 "option 5" on
 6 "option 6" off
 7 "option 7" off)

# We create the function selections with options running funcheck
# and forwards the output to the terminal for the next run
# the commands
selections = $("${funcheck[@]}" "${options[@]}" 2>&1 >/dev/tty)

# clean the screen
clear
```

```
# add a for loop to run a command function
# the selections can change the echo by
# any commands or scripts
for selection in $ selections
do
 $ selection in case
 1)
 echo "You chose the option 1"
 ;;
 2)
 echo "You chose the option 2"
 ;;
 3)
 echo "You chose the option 3"
 ;;
 4)
 echo "You chose the option 4"
 ;;
 5)
 echo "You chose the option 5"
 ;;
 6)
 echo "You chose the option 6"
 ;;
 7)
 echo "You chose the option 7"
 ;;
 esac
done
```

Conclusions on dialog

In my opinion dialog is an elegant way to create menus for our scripts can easily create programs for terminal.

Darte las gracias por leer nuestro blog espero que te guste nuestro contenido, te pediria que no olvides dar +1 o compartir en las redes sociales, no ganamos nada con este blog y es para nosotros cada me gusta o mas 1 es una pequeña recompensa

Create menus and dialogs for shell scripts

Forms

If you need not only simple forms but also a complex input box, dialog provides you with the necessary tools. You can navigate in the forms using the tab and arrow keys. The syntax might seem confusing at first, in that there are size and position adjustments to make.

Listing 9 is a script that allows editing of data and shows how to return it as variables that can stored permanently in a file. Figure 5 shows how the position and size values are rendered.

Listing 9

Return Data as Variables

```
#! /bin/sh
a="Mr."
b="Tux"
c="Penguin"
dialog --title "Title" --backtitle "Customer Data" --ok-label "Save" \
  --stdout --form "Catalog" 10 60 3 "Salutation  " 1 1 "$a" 1 15 30 0 \
  "Family Name " 2 1 "$b" 2 15 30 0 "First Name  " 3 1 "$c" 3 15 30 0 >
output.txt
a=$(cat output.txt | head -1)
b=$(cat output.txt | head -2 | tail -1)
c=$(cat output.txt | head -3 | tail -1)
rm output.txt
dialog --title "Title" --backtitle "Background Title" --msgbox \
  "Saved values: \n $a \n $b \n $c " 0 0
```

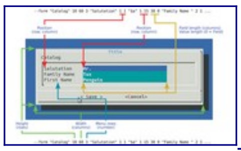


Figure 5: Position and size values for form elements.

To ensure that redirecting output to a file works, you can precede *--form* with the additional *--stdout* option. In *msgbox* , use the newline character (*\n*) to output each variable value on a separate line.

Forms with the *inputmenu* function need fewer size and position parameters, but you may not get the data of all the fields at the end. You can change only single fields with each dialog call. The output includes the action (*RENAMED*), field identifier and data. To be sure the function works, use the *--stdout* option, as you do for *--inputmenu* . Post-processing of data occurs with *cut* (Listing 10). Here, it's important that the field names do not have space characters. Databases can provide the corresponding field names so that it's easy to provide SQL statements with matching variables.

Listing 10

Post-Processing of Data

```
#! /bin/sh
newval=$(dialog --title "Title" --backtitle "Background Title" --stdout --
inputmenu "MENU HEADING" \
  17 60 15 "row-1 >" "Value 1" "row-2 >" "Value 2" "row-3 >" "")
column=$(echo $newval | cut -d \> -f 1 | cut -b 9-)
entry=$(echo $newval | cut -d \> -f 2 | cut -b 2-)
```

```
dialog --title "Title" --backtitle "Background Title" --msgbox "Stored Values: \
n $newval \n $column \n $entry " 0 0
dialog --title "Title" --backtitle "Background Title" --msgbox "Saved values: \n
$a \n $b \n $c " 0 0
```

The *--calendar* function provides an easy way to input date information. Calling it displays the current month and the day of the week in the left column. Use the cursor and up/down arrows to move to other data. Clicking the *OK* button returns the highlighted value.

To fill a variable, again use the *--stdout* option for output. Because dialog separates the data parts with a slash character, *tr* translates them into commas. Listing 11 shows how to construct a script that does just that.

Listing 11

Translate Data with tr

```
#! /bin/sh
a=$(dialog --title "Title" --backtitle "Background Title" --stdout --calendar
"TEXT" 0 0 | tr \/ .)
dialog --title "Title" --backtitle "Background Title" --msgbox "Selected date:
$a " 0 0
```

You enter time values with *--timebox* . To accept the entered value, again use *--stdout* . The window shows the time the call to the function was made. To enter different values, use the arrow key to proceed to the next entry field and provide another date.

You can enter any alphanumeric values using the input box (*--inputbox*). Register a value in a script by using the prefix *--stdout* option. You can provide an editable default value. The process requires only one line:

```
a=$(dialog --title "Title" \
  --backtitle "Background Title" \
  --stdout --inputbox "HEADLINE" \
  0 0 "DEFAULT")
```

You can edit small text files in a mini-editor (*--editbox*). Give the filename as an argument. If no file exists, create one with *touch* .

You can write enter or modify text and then save to another file or overwrite the original file. Specify a height and width for the widget so that the inner window doesn't displace the headline. Again, use *--stdout* so that the entries don't end up in the standard error output. This function again requires a simple line of shell code, as follows:

```
dialog --title "Title" --backtitle \
  "Background Title" --stdout \
  --ok-label "Save" --editbox text.txt \
  20 75 > new.txt
```

You can select directories and filenames from within the script with *--dselect* and *--fselect* , respectively.

These two parameters are usually used together. It's usually a good idea to prompt users unfamiliar with the context. You select a directory at the same level as the default with arrow keys and pressing the spacebar.

Choose the path on the left with *--fselect* ; press the spacebar twice and choose the file on the right. Navigate between fields with the tab key and inside fields with the arrow keys.

The bottom field allows for manual entries. With the cursor in it, move to the next highest level by deleting everything up to the right-most slash (Figure 6). You can see how to select a directory and file in Listing 12.

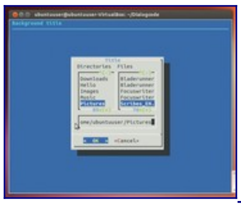


Figure 6: The --dselect and --fselect widgets make choosing directories and files easy.

Listing 12

Select a Directory and File

```
#!/bin/sh
a=$(echo $HOME)
while true
do
  a=$(dialog --title "Title" --backtitle "Background Title" --stdout --fselect
$a 0 0)
  dialog --title "Title" --backtitle "Background Title" --defaultno --yesno "$a
Apply"  0 0
  if [ $? -eq 0 ]; then
    break
  fi
done
```

You can provide multiple selections using *--checklist* . This option returns the designator of each entry enclosed in quotes. Field separators are space characters. You give each entry a status of *on* for marked or *off* for unmarked. You can specify space-separated values for height, width, and list height.

The list height should correspond to the actual number of items. If all the items don't fit, the box is scrollable.

Listing 13 shows the code together with *tr* , which removes the quotes. You can build a single-selection box in the same way by using the *--radiolist* dialog type and setting a single entry to *on* , getting the tag value without the quotes. Use the space bar to choose the value:

```
a=$(dialog --title "Title" \
  --backtitle "Background Title" --stdout \
  --radiolist "SELECTION HEADING" \
  10 40 3 TAG-1 "INFO-1" on \
  TAG-2 "INFO-2" off TAG-3 "INFO-3" off)
```

Listing 13

Provide Multiple Selections

```
#!/bin/sh
a=$(dialog --title "Title" --backtitle "Background Title" --stdout \
           --checklist "SELECTION HEADING" 10 40 3 TAG-1 "INFO-1" on \
           TAG-2 "INFO-2" off TAG-3 "INFO-3" on)
a=$(echo $a | tr -d \")
```

```
sleep 5
```

This changes the item selection to a single one. The script creates the data query dynamically. The tag consists of the record value or a unique value that the script uses to read in the record value.

The example in Listing 14 shows an SQL query with the psql client for the PostgreSQL database. The aim is to pass the customer number *orgnr* to other parts of the script. The script writes the instructions for the menu with *--radiolist* into a variable and executes the code with *eval* . For further development, the *--form* or *--inputmenu* dialog type can be used depending on programming preference.

Listing 14

Sample SQL Query

```
#!/bin/sh
# Search dialog for the database query
sube=$(dialog --title "Customer Search" --backtitle "Customer Management" \
  --stdout --inputbox "Enter customer name" 0 0 "")
# Check for sufficient data
a=$(psql -t -c "select orgnr from customers where name = '$sube';")
if [ -z "$a" ]; then
  dialog --title "Customer Search" --backtitle "Customer Management" \
         --msgbox "No matching customers found! " 0 0
  exit  # or break when in loop!
fi
# Get data for single-selection and select structure
# Counters for first "on" status
v=0
teil1=$(echo "a=\`dialog --title \"Customer Search\" --backtitle \"Customer
Management\" \
  --stdout --radiolist \"found: \" 10 40 3 ")
# Determine customer numbers
for i in $(psql -t -c "select orgnr from customers where name = '$sube' \
  order by lastname, firstname, birthday ;"); do
  if [ $v -eq 0 ]; then
    status="on"
    v=1
  else
    status="off"
  fi
  info=$(psql -t -c "select (lastname || , , || firstname || , , || birthday ) \
  from customers where orgnr = $i;";)
  part2=`echo $part2 $i \"$info\" $status`
done
selection_mask=$(echo "$part1 $part2\`")
echo $selection_mask
eval $selection_mask
dialog --title "Customer Search" --backtitle "Customer Management" \
  --msgbox "Record number $a selected " 0 0
```

You can add menus to your program with the *--menu* dialog type. It provides a tag that you later evaluate. As with almost all other value output, use *--stdout* before the dialog type. Listing 15 shows an example.

Listing 15

Add Menus

```
#!/bin/sh
```

```
while true; do
  a=`dialog --title "TITLE" --backtitle "BACKGROUND TITLE" \
    --stdout --menu "MENU HEADING" 0 0 0 1 "FIRST" 2 "SECOND" 9 "END"`
  if [ $a -eq 1 ]; then
    dialog --title "Title" --backtitle "Background Title" \
    --msgbox "First entry selected" 0 0
  elif [ $a -eq 2 ]; then
    dialog --title "Title" --backtitle "Background Title" \
    --msgbox "Second entry selected" 0 0
  elif [ $a -eq 9 ]; then
    dialog --title "Title" --backtitle "Background Title" \
    --no-label "End of program" --yes-label "Continue" \
      --yesno "End of program?" 0 0
    if [ $? -eq 1 ]; then
      break # or exit
    fi
  fi
done
dialog --title "Title" --backtitle "Background Title" \
  --msgbox "Saved values: \n $a \n $b \n $c " 0 0
```

Conclusion

If you want to enhance your shell scripts with a user-friendly interface, you'll find everything you need with the dialog tool.

If you require simple queries only, you can benefit from prefabricated building blocks. Otherwise, you can always use something other than shell code.

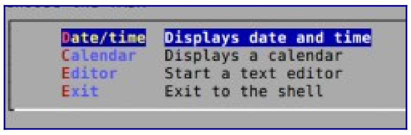
A menu box

Jump to navigation Jump to search

← A password box • **Home** • A progress bar (gauge box) →

- A **menu box** display a list of choices to the user in the form of a menu.
- Each menu is made of a *tag* string and an *item* string. In this example, a tag (e.g., Calendar) is on left side and an item (e.g., "Displays a calendar") is on right side:

```
Date/time "Displays date and time" \
Calendar "Displays a calendar" \
Editor "Start a text editor" \
Exit "Exit to the shell"
```


Date/time Displays date and time
Calendar Displays a calendar
Editor Start a text editor
Exit Exit to the shell
A sample menu output

- The *tag* gives the entry a name to distinguish it from the other entries in the menu. Use the *tag* to make decision using if statement or case..esac statement.
- The item is nothing but a short description of the option that the entry represents.
- All choices (menus) are displayed in the order given.
- On exit the *tag* of the chosen menu entry will be printed on dialog's output. This can be redirected to the file using the following syntax:

```
> /tmp/menu.output
```

- If the "--help-button" option is given, the corresponding help text will be printed if the user selects the help button.

Example

- Create a shell script called utilitymenu.sh:

```
#!/bin/bash
# utilitymenu.sh - A sample shell script to display menus on screen
# Store menu options selected by the user
INPUT=/tmp/menu.sh.$$

# Storage file for displaying cal and date command output
OUTPUT=/tmp/output.sh.$$
```

```
# get text editor or fall back to vi_editor
vi_editor=${EDITOR-vi}

# trap and delete temp files
trap "rm $OUTPUT; rm $INPUT; exit" SIGHUP SIGINT SIGTERM

#
# Purpose - display output using msgbox
#  $1 -> set msgbox height
#  $2 -> set msgbox width
#  $3 -> set msgbox title
#
function display_output(){
        local h=${1-10}                 # box height default 10
        local w=${2-41}                 # box width default 41
        local t=${3-Output}     # box title
        dialog --backtitle "Linux Shell Script Tutorial" --title "${t}" --clear
--msgbox "$(<$OUTPUT)" ${h} ${w}
}
#
# Purpose - display current system date & time
#
function show_date(){
        echo "Today is $(date) @ $(hostname -f)." >$OUTPUT
    display_output 6 60 "Date and Time"
}
#
# Purpose - display a calendar
#
function show_calendar(){
        cal >$OUTPUT
        display_output 13 25 "Calendar"
}
#
# set infinite loop
#
while true
do

### display main menu ###
dialog --clear  --help-button --backtitle "Linux Shell Script Tutorial" \
--title "[ M A I N - M E N U ]" \
--menu "You can use the UP/DOWN arrow keys, the first \n\
letter of the choice as a hot key, or the \n\
number keys 1-9 to choose an option.\n\
Choose the TASK" 15 50 4 \
Date/time "Displays date and time" \
Calendar "Displays a calendar" \
Editor "Start a text editor" \
Exit "Exit to the shell" 2>"${INPUT}"

menuitem=$(<"${INPUT}")


# make decsion
case $menuitem in
        Date/time) show_date;;
        Calendar) show_calendar;;
        Editor) $vi_editor;;
        Exit) echo "Bye"; break;;
esac

done
```

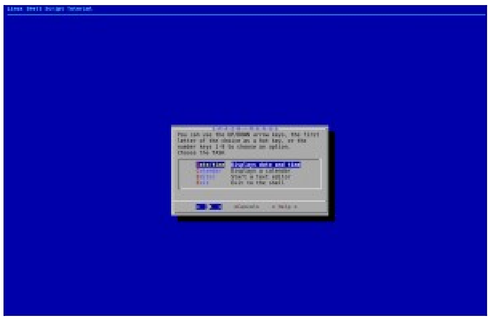
```
# if temp files found, delete em
[ -f $OUTPUT ] && rm $OUTPUT
[ -f $INPUT ] && rm $INPUT
```

Save and close the file. Run it as follows:

```
chmod +x utilitymenu.sh
./utilitymenu.sh
```

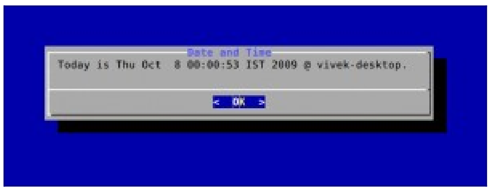
Sample outputs:

- utilitymenu.sh shell script output (dialog command with menus)

- 

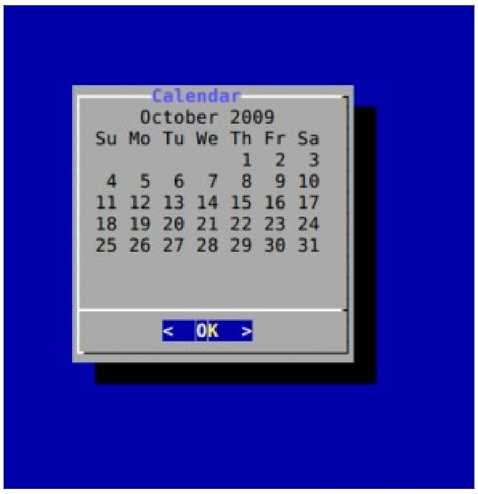
```
Linux Shell Script Tutorial

You can use the UP/DOWN arrow keys, the first
letter of the choice as a hot key, or the
number keys 1-9 to choose an option.
Choose the TASK

    Date/time   Displays date and time
    Calendar    Displays a calendar
    Editor      Start a text editor
    Exit        Exit to the shell

    <  OK  >    <Cancel>    < Help >
```

  A menu based dialog box

- 

```
                     Date and Time
Today is Thu Oct  8 00:00:53 IST 2009 @ vivek-desktop.

                        <  OK  >
```

  Messagebox displaying date & time

- 
```
Calendar
     October 2009
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

Messagebox displaying a calendar

The file selection box

Jump to navigation Jump to search

← A progress bar (gauge box) • **Home** • The form dialog for input →

- The **fselect (file-selection) dialog** box shows a text data entry window in which operator can type or select a filename / directory name from the directory or filepath.
- An operator can use tab or arrow keys to move between the windows. Press the space-bar to select the filename into box.

Syntax

```
dialog --title "text" --fselect /path/to/dir height width
FILE=$(dialog --stdout --title "Please choose a file" --fselect $HOME/ 14 48)
echo "${FILE} file chosen."
```

Where,

- *filepath* - can be a filepath in which case the file and directory windows will display the contents of the path and the text-entry window will contain the preselected filename.
- *height* - set dialog box height.
- *width* - set dialog box width.

Example

- Create a script called deletefile.sh:

```
#!/bin/bash
# deletefile.sh - Remove the file using dialog box
# --------------------------------------------------
# purpose - remove file
#  $1 - filename
function delete_file(){
        local f="$1"
        local m="$0: file $f failed to delete."
        if [ -f $f ]
        then
                /bin/rm $FILE && m="$0: $f file deleted."
        else
                m="$0: $f is not a file."
        fi
        dialog --title "Remove file" --clear --msgbox "$m" 10 50
}
```

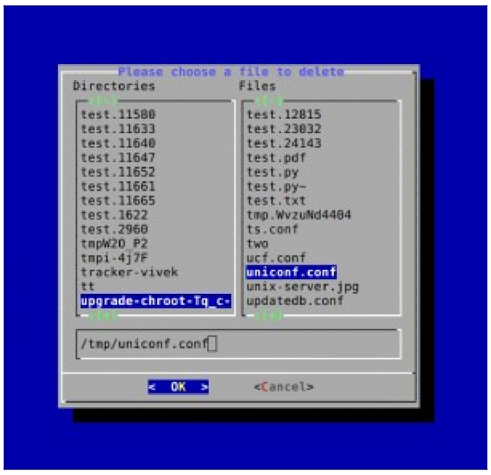
```
# select filename using dialog
# store it to $FILE
FILE=$(dialog --title "Delete a file" --stdout --title "Please choose a file to
delete" --fselect /tmp/ 14 48)

# delete file
[ ! -z $FILE ] && delete_file "$FILE"
```

Save and close the file. Run it as follows:

```
chmod +x deletefile.sh
./deletefile.sh
```

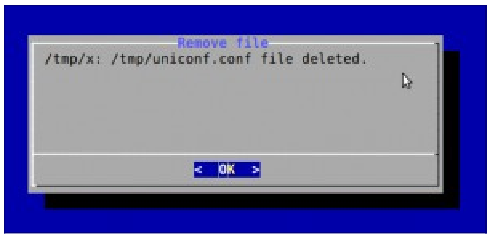
Sample outputs:

- deletefile.sh: shell script output (dialog command with file selection)

- 

```
Please choose a file to delete
Directories                 Files
test.11580                  test.12815
test.11633                  test.23032
test.11640                  test.24143
test.11647                  test.pdf
test.11652                  test.py
test.11661                  test.py~
test.11665                  test.txt
test.1622                   tmp.WvzuNd4404
test.2960                   ts.conf
tmpW2O_P2                   two
tmpi-4j7F                   ucf.conf
tracker-vivek               uniconf.conf
tt                          unix-server.jpg
upgrade-chroot-Tq_c-        updatedb.conf

/tmp/uniconf.conf
```

  The file selection box in action

- 

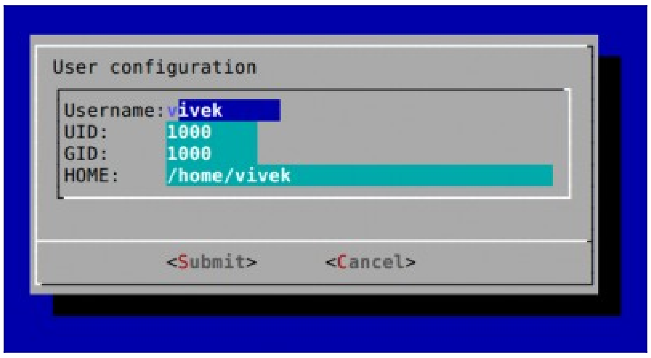
```
Remove file
/tmp/x: /tmp/uniconf.conf file deleted.
```

  Messagebox confirming file deletion operation

External links

- dialog

The form dialog for input



```
User configuration

Username:vivek
UID:      1000
GID:      1000
HOME:     /home/vivek
```

A sample data entry form dialog box

- The **form dialog** displays data entry form which consisting of labels and fields.
- You can set the field length.
- An operator can use up/down arrows to move between fields and tab to move between windows.

The syntax is as follows:

```
dialog --form text height width formheight [ label y x item y x flen ilen ]
```

- Where,
  - The field length flen and input-length ilen tell how long the field can be.
  - If flen is zero, the corresponding field cannot be altered. and the contents of the field determine the displayed-length.
  - If flen is negative, the corresponding field cannot be altered, and the negated value of flen is used as the displayed-length.
  - If ilen is zero, it is set to flen.

Example

- Create a shell script called useradd1.sh:

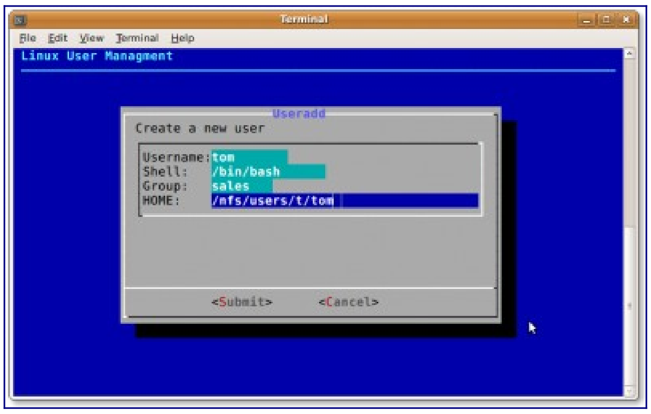
```
#!/bin/bash
# useradd1.sh - A simple shell script to display the form dialog on screen
# set field names i.e. shell variables
shell=""
groups=""
user=""
home=""

# open fd
exec 3>&1

# Store data to $VALUES variable
VALUES=$(dialog --ok-label "Submit" \
          --backtitle "Linux User Managment" \
          --title "Useradd" \
          --form "Create a new user" \
15 50 0 \
        "Username:" 1 1 "$user"         1 10 10 0 \
        "Shell:"    2 1 "$shell"        2 10 15 0 \
        "Group:"    3 1 "$groups"       3 10 8 0 \
        "HOME:"     4 1 "$home"         4 10 40 0 \
2>&1 1>&3)

# close fd
exec 3>&-

# display values just entered
echo "$VALUES"
```

Save and close the file. Run it as follows:

```
chmod +x useradd1.sh
./useradd1.sh
```

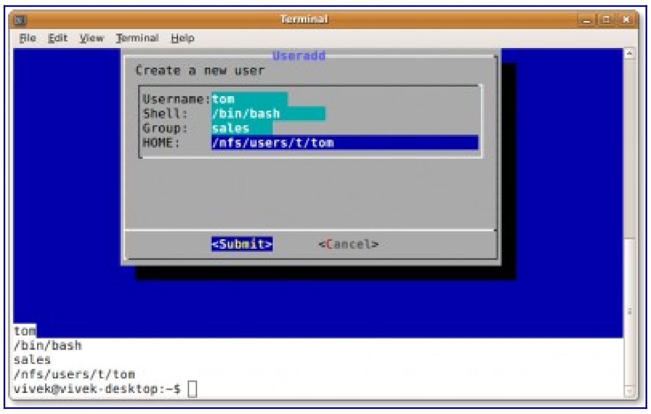
Sample outputs:

- useradd1.sh: shell script output (the dialog command with data entry form)

- 

```
Linux User Managment

Useradd
Create a new user
Username:tom
Shell:   /bin/bash
Group:   sales
HOME:    /nfs/users/t/tom

      <Submit>     <Cancel>
```

The form dialog box in action

- 

```
Create a new user

Username:tom
Shell:   /bin/bash
Group:   sales
HOME:    /nfs/users/t/tom
```

```
tom
/bin/bash
sales
/nfs/users/t/tom
vivek@vivek-desktop:~$
```

Displaying output stored in $VALUES

https://bash.cyberciti.biz/guide/The_form_dialog_for_input

How To Create GUI Dialog Boxes In Bash Scripts With Whiptail In Linux

Written by Karthick *1545* Views
0 comment
6

A while ago, we briefly discussed about **Zenity**, a simple program that allows you to create graphical (GTK+) dialog boxes in command-line and shell scripts. In this article, we are going to discuss yet another GUI utility called **Whiptail** that can be used to create GUI dialog boxes in Bash scripts in Linux.

Not every script that you write needs a frontend graphical interface. But sometimes it would be better if you create a graphical interface instead of relying on interacting with the command line. In my case, if there is a long list of responses needed in the script I would choose to go with a graphical interface.

Whiptail is a friendly GUI utility that uses a **newt** programming library. Whiptail offers different dialog boxes for different purposes. Depending upon your use case you can use these dialog boxes to make your script more interactive.

Install Whiptail in Linux

Whiptail comes pre-installed with many distributions but if your distribution has no whiptail installed follow the below instructions to install it.

To check if whiptail is already installed run the following command.

```
$ which whiptail
```

To install Whiptail on Debian/Ubuntu and its derivative distributions, run the following command:

```
$ sudo apt install whiptail -y
```

Fedora/RHEL/CnetOS/AlmaLinux/Rocky Linux:

```
$ sudo dnf install newt
```

Arch Linux, EndeavourOS, Manjaro Linux:

```
$ sudo pacman -S whiptail
```

Alpine Linux:

```
$ apk add newt
```

Help Option

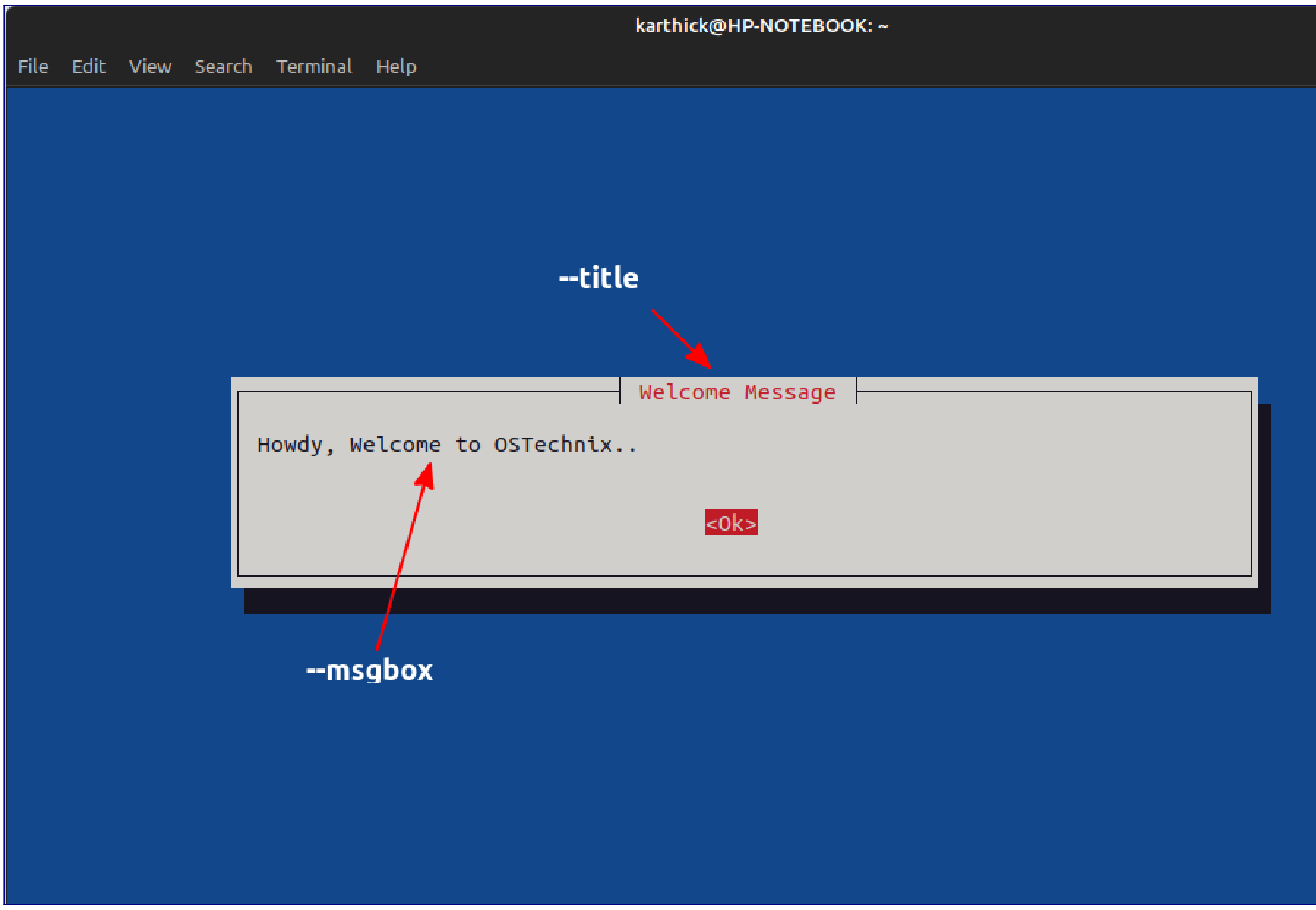
You can use `--help` flag which will display the list of supported dialog boxes and other options you can use. In total, there are 10 dialog box supported with various functionality and we will look at them about all in the upcoming sections.

```
$ whiptail -help
Box options:
    --msgbox <text> <height> <width>
    --yesno  <text> <height> <width>
    --infobox <text> <height> <width>
    --inputbox <text> <height> <width> [init]
    --passwordbox <text> <height> <width> [init]
    --textbox <file> <height> <width>
    --menu <text> <height> <width> <listheight> [tag item] ...
    --checklist <text> <height> <width> <listheight> [tag item status]...
    --radiolist <text> <height> <width> <listheight> [tag item status]...
    --gauge <text> <height> <width> <percent>
```

1. Message Box

Message box will display messages to the user and wait till the user presses **<ok>** or **<ESC>** key. When you press **<ok>** it will throw a **return code 0** and if you press **<ESC>** it will throw **return code 255**.

```
$ whiptail --title "Welcome Message" --msgbox "Howdy, Welcome to OSTechnix.." 8
78
```



```
Welcome Message

Howdy, Welcome to OSTechnix..
```

Let's decode the above command.

--title	This will add title to the window
--msgbox	This will print the message you provide within the quotes.
8 78	This set Height(8) and Width(78) of the window.

You can open a new terminal and check for the whiptail process. It will be in a sleep state. Meaning - it is waiting for you to press **<ok>** or **<ESC>**.

```
$ ps -ef | grep -i whiptail
karthick   20023    9251  0 22:41 pts/0    00:00:00 whiptail --title Welcome
Message --msgbox Howdy, Welcome to OSTechnix.. 8 78
karthick   20071   19379  0 22:41 pts/1    00:00:00 grep --color=auto -i
whiptail

$ ps -q 20023 -o state --no-headers
S
```

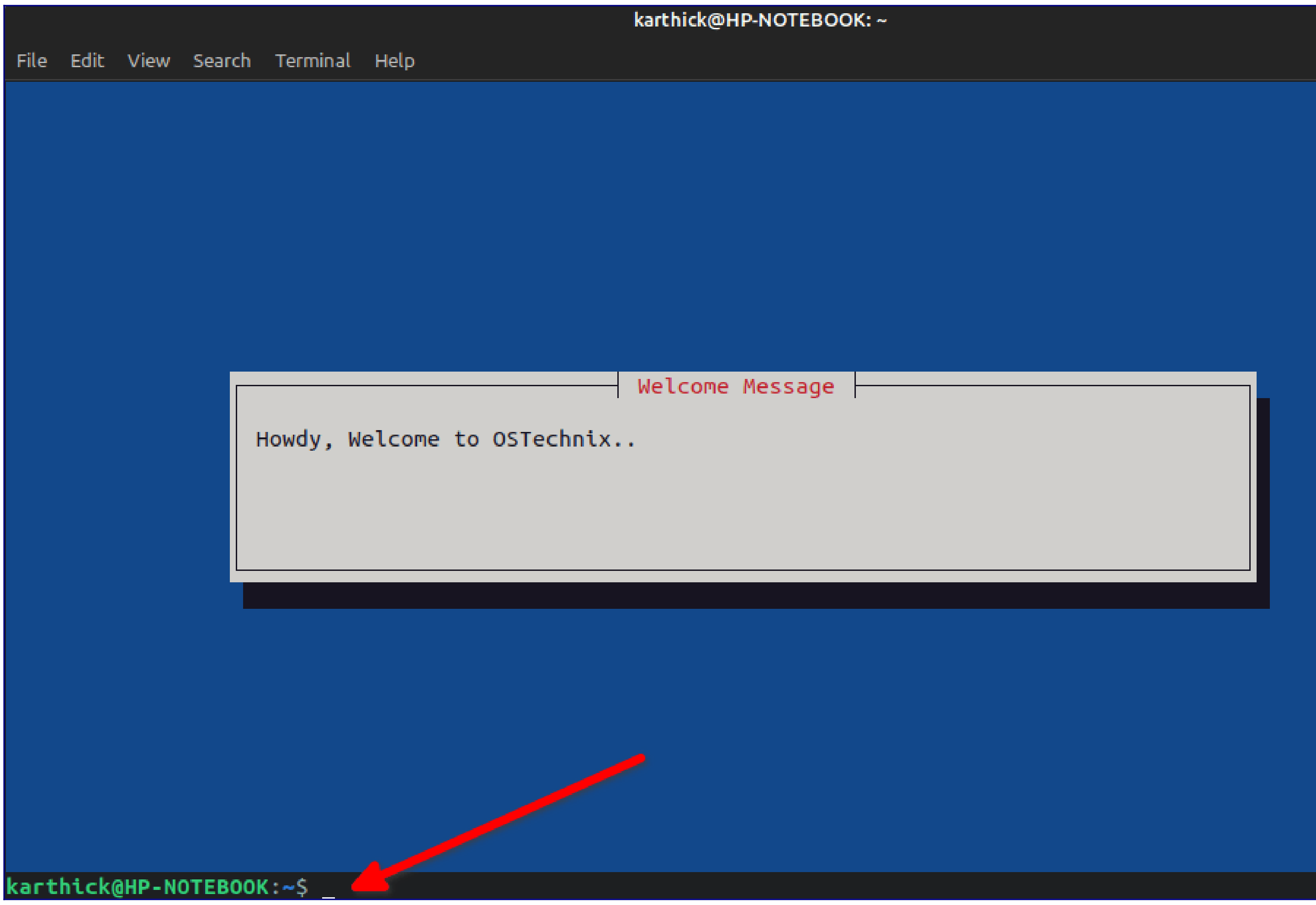
State (**S**) -> interruptible sleep (waiting for an event to complete).

2. Info Box

Info is similar to the message box but the difference is unlike the message box info box will not wait for the user input. Use `--infobox` flag and pass a string as an argument which will be displayed in the info box.

In some shells, info box will run but will not display any result. You have to change the terminal emulation and run it as I did in the below snippet.

```
$ TERM=ansi whiptail --title "Welcome Message" --infobox "Howdy, Welcome to
OSTechnix.." 8 78
```



```
Welcome Message

Howdy, Welcome to OSTechnix..

karthick@HP-NOTEBOOK:~$ _
```

3. Yes/No Box

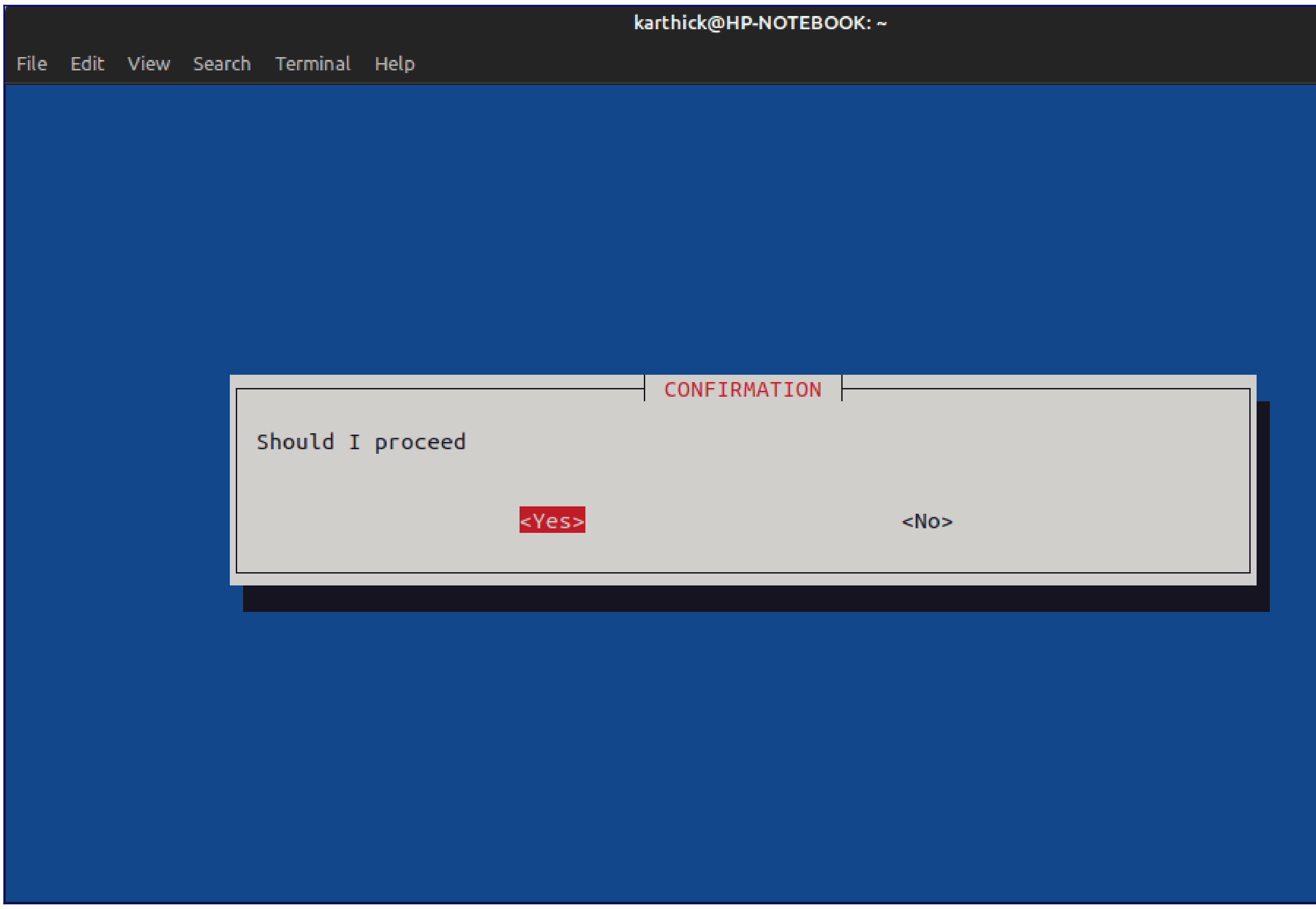
Yes/No Box will display a dialog box with **YES** or **NO** option where if you choose **<Yes>** it will throw **return code 0** and when you press **<No>** it will throw **return code 1**.

Use `--yesno` flag to prompt for the choice. Run the following snippet which combines the yes/no box and the message box. At first, it will display the Yes/No option, and depending upon your choice it will throw the return code.

Create a shell script, copy the below snippet, and run it.

```
#!/usr/bin/env bash

whiptail --title "CONFIRMATION" --yesno "Should I proceed" 8 78
if [[ $? -eq 0 ]]; then
  whiptail --title "MESSAGE" --msgbox "Process completed successfully." 8 78
elif [[ $? -eq 1 ]]; then
  whiptail --title "MESSAGE" --msgbox "Cancelling Process since user pressed
<NO>." 8 78
elif [[ $? -eq 255 ]]; then
```

```
  whiptail --title "MESSAGE" --msgbox "User pressed ESC. Exiting the script" 8
78
fi
```



```
CONFIRMATION

Should I proceed

<Yes>                    <No>
```

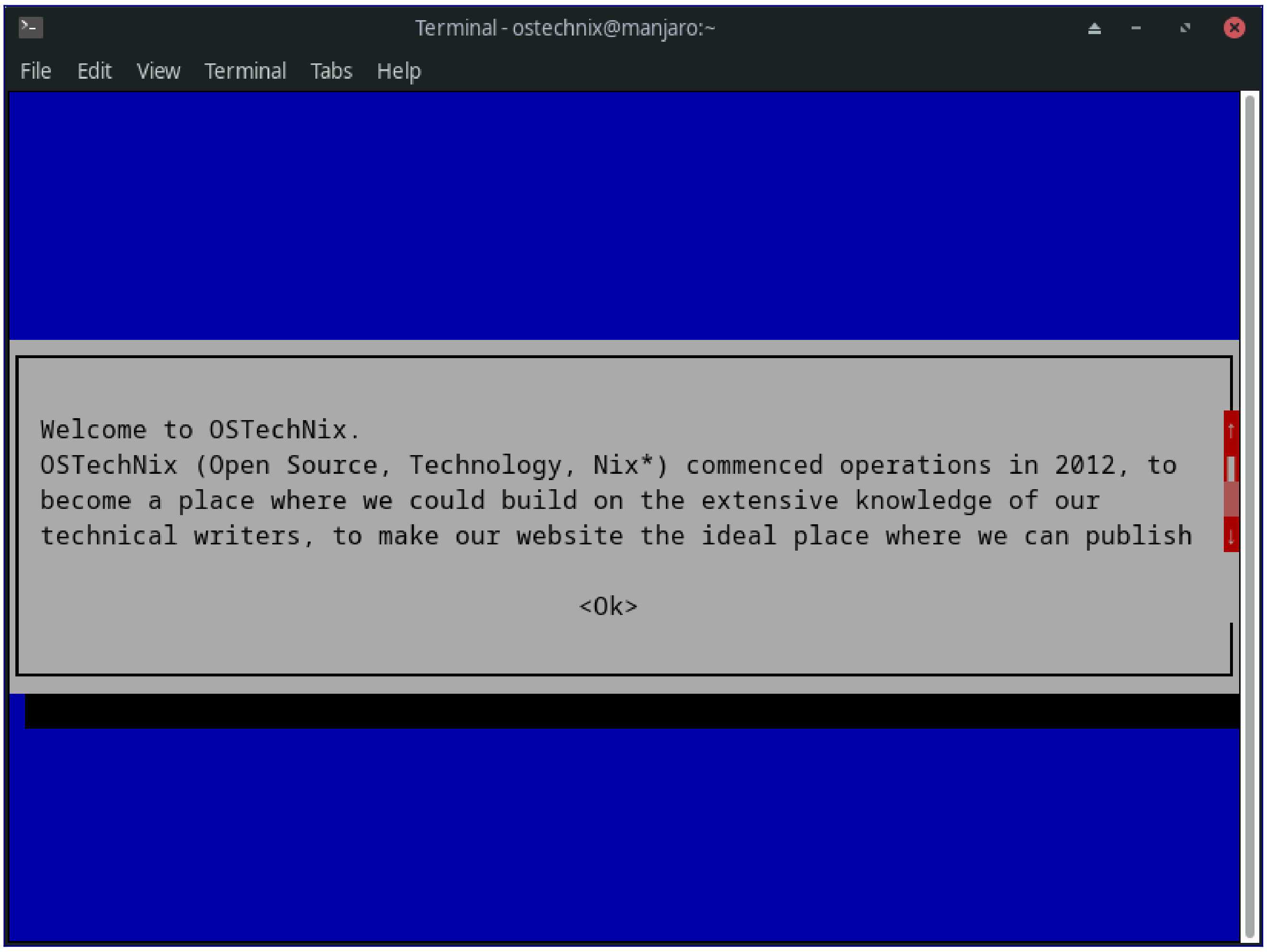
If you have no idea about bash conditional statements, take a look at our brief article on the same using the link below.

- **Bash Scripting – Conditional Statements**

4. Text Box

Text box will read and print the file. In the below snippet I am reading the `ostechnix.txt` file. The flag –scrolltext allows you to use the mouse wheel to scroll vertically when you have long pages of text that do not fit into the current window.

```
$ whiptail --textbox --scrolltext ostechnix.txt 10 80
```


Terminal - ostechnix@manjaro:~
File Edit View Terminal Tabs Help

```
Welcome to OSTechNix.
OSTechNix (Open Source, Technology, Nix*) commenced operations in 2012, to
become a place where we could build on the extensive knowledge of our
technical writers, to make our website the ideal place where we can publish

                                   <Ok>
```

5. Redirections

The dialog boxes that you are going to see in the upcoming section require output to be stored in a variable and later use for processing. The return value from the widget is sent to stderr instead of stdout. So you have to swap stdout and stderr so that the result is stored in the variable.

You have to use the following expression to swap stdout and stderr.

```
3>&1 1>&2 2>&3
```

Let’s try to understand the above expression. You know FD1 is the standard output and FD2 is the standard error.

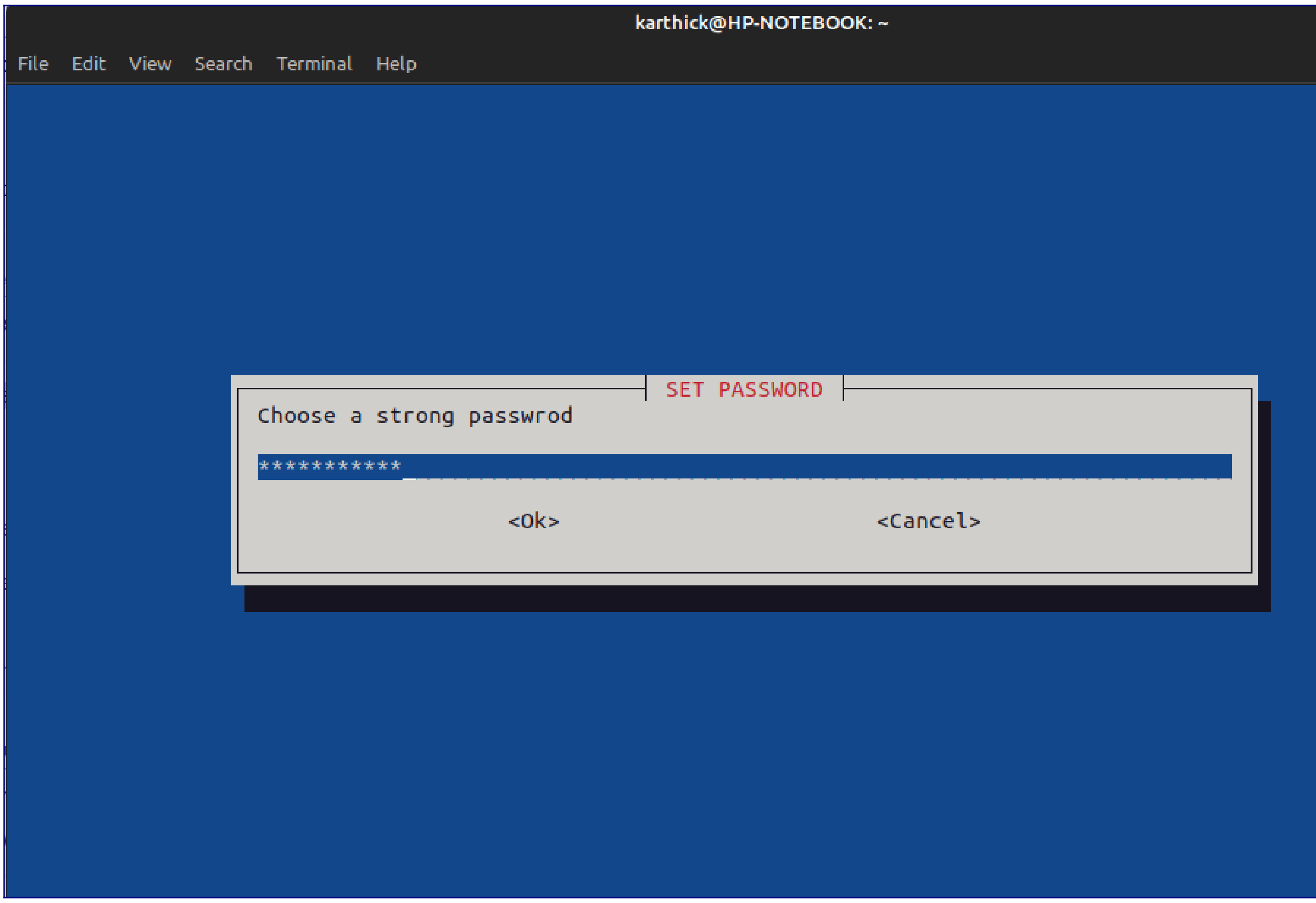
- **3>&1** - Anything that is redirected to file descriptor 3 is redirected to file descriptor 1.
- **1>&2** - Anything that is sent to file descriptor 1(Stdout) is redirected to file descriptor 2.
- **2>&3** - Anything that is sent to file descriptor 2(stderr) is redirected to file descriptor 3.

This way we are swapping stdout and stderr so the variable can store the return value from dialog boxes.

6. Password Box

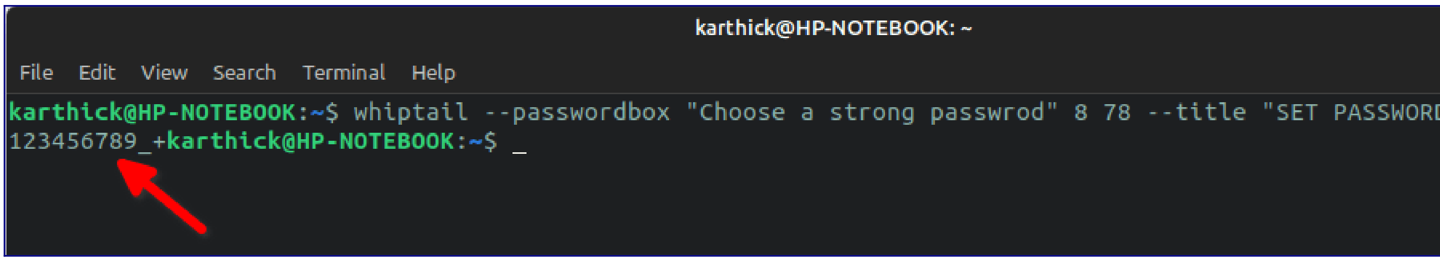
Using the password dialog box you can type passwords that will not be displayed as plain text as you type. Use `--passwordbox` to prompt to enter the password.

```
$ whiptail --title "SET PASSWORD" --passwordbox "Choose a strong password"
```



```
SET PASSWORD
Choose a strong passwrod
***********
<Ok>                <Cancel>
```

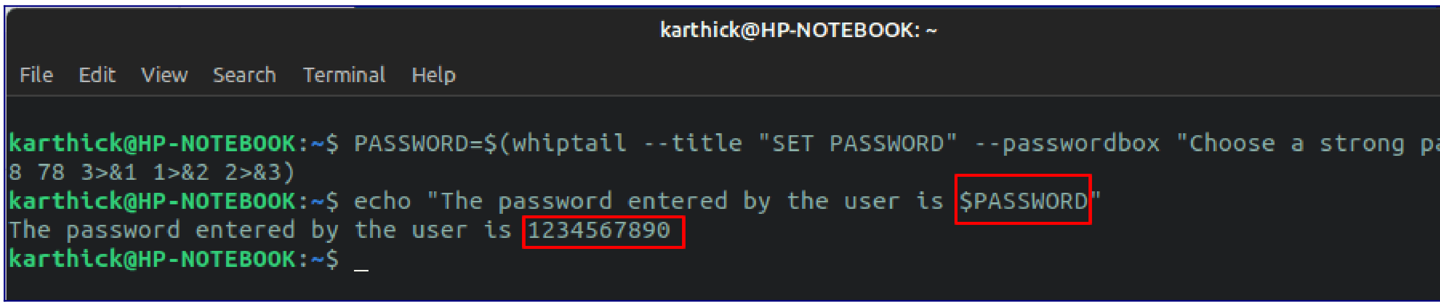
When you press <**ok**>, it will throw the return code 0 and will return the password you typed to the console (stderr) if you are running from the terminal.



```
karthick@HP-NOTEBOOK:~$ whiptail --passwordbox "Choose a strong passwrod" 8 78 --title "SET PASSWORD
123456789_+karthick@HP-NOTEBOOK:~$ _
```

You need to capture the password into a variable then later use it in the script. As discussed in the Redirection section you have to redirect the result.

```
$ PASSWORD=$(whiptail --title "SET PASSWORD" --passwordbox "Choose a strong
password" 8 78 3>&1 1>&2 2>&3)
$ echo "The password entered by the user is $PASSWORD"
```

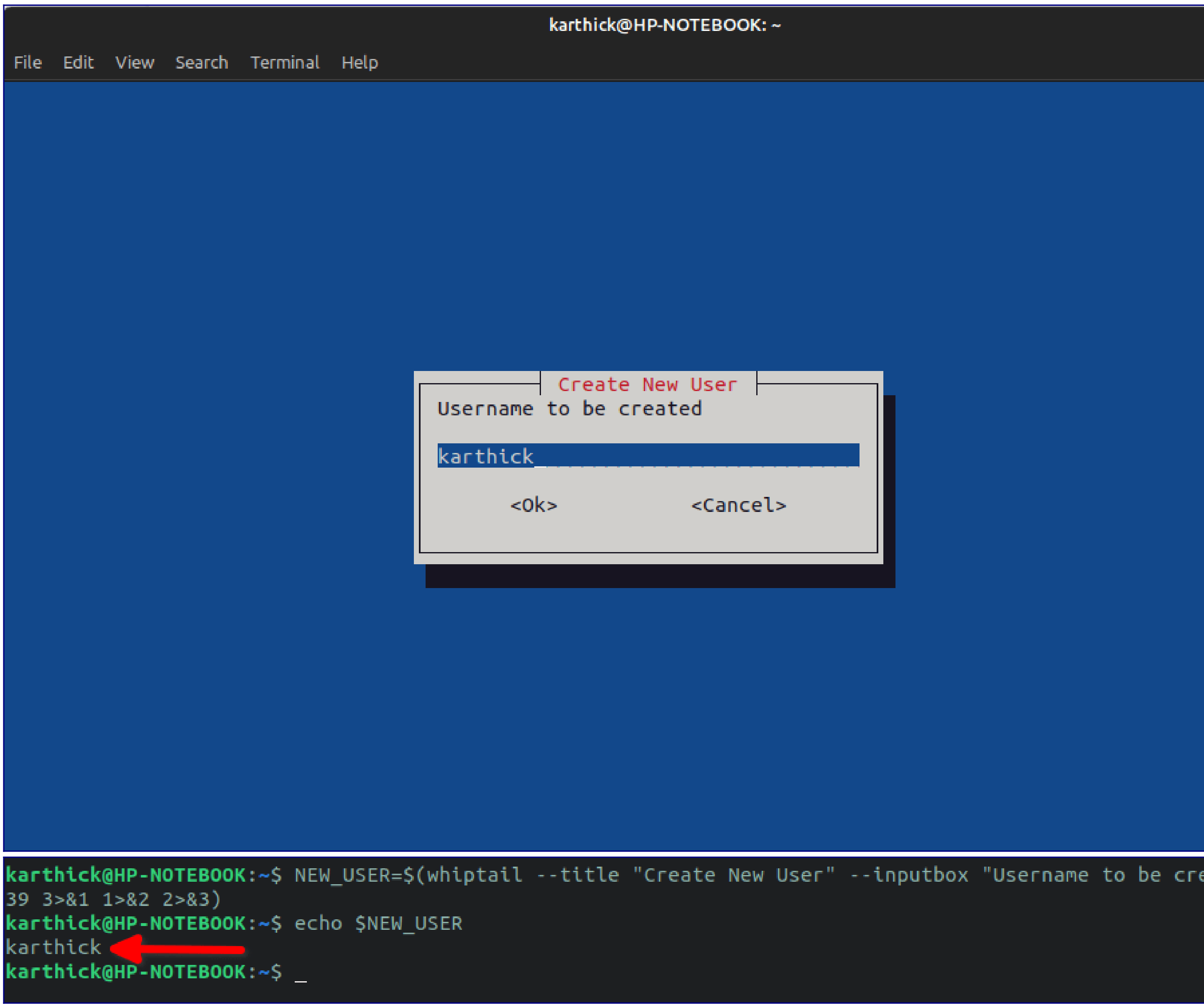


```
karthick@HP-NOTEBOOK:~$ PASSWORD=$(whiptail --title "SET PASSWORD" --passwordbox "Choose a strong pa
8 78 3>&1 1>&2 2>&3)
karthick@HP-NOTEBOOK:~$ echo "The password entered by the user is $PASSWORD"
The password entered by the user is 1234567890
karthick@HP-NOTEBOOK:~$ _
```

7. Input Box

Input dialog box will prompt the user to provide the input. As with the password dialog box, the input given by you will be printed to the terminal if you are running it from the terminal. You have to use the redirections and store the value to a variable then later use for processing according to your program logic.

```
NEW_USER=$(whiptail --title "Create New User" --inputbox "Username to be
created" 8 40 3>&1 1>&2 2>&3)
```



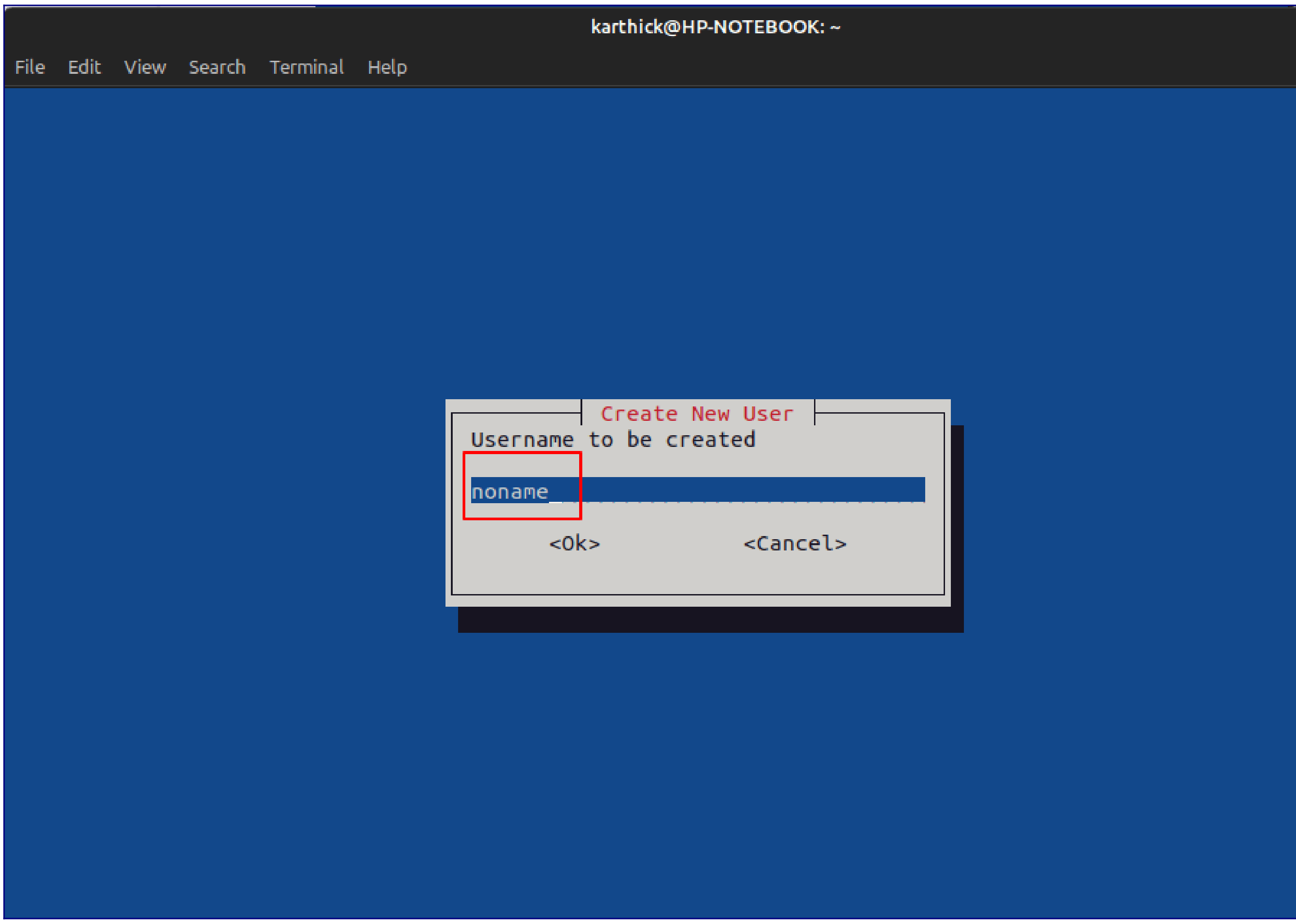
```
karthick@HP-NOTEBOOK:~$ NEW_USER=$(whiptail --title "Create New User" --inputbox "Username to be cre
39 3>&1 1>&2 2>&3)
karthick@HP-NOTEBOOK:~$ echo $NEW_USER
karthick
karthick@HP-NOTEBOOK:~$ _
```

You can also set default input text. All you have to do is add the text after height and width. Below is the syntax where instead of `[init]` you will place the default text.

```
--inputbox <text> <height> <width> [init]
```

Example:

```
whiptail --title "Create New User" --inputbox "Username to be created" 8 40
noname
```



```
Create New User
Username to be created

noname_

<Ok>          <Cancel>
```

Now let’s combine the input box, password box, Yes/No Box and Text Box and write a simple user creation program to see how these dialog boxes can be coupled together.

8. Checklist Dialog Box

Checklist allows you to create a list of options that a user can select from.

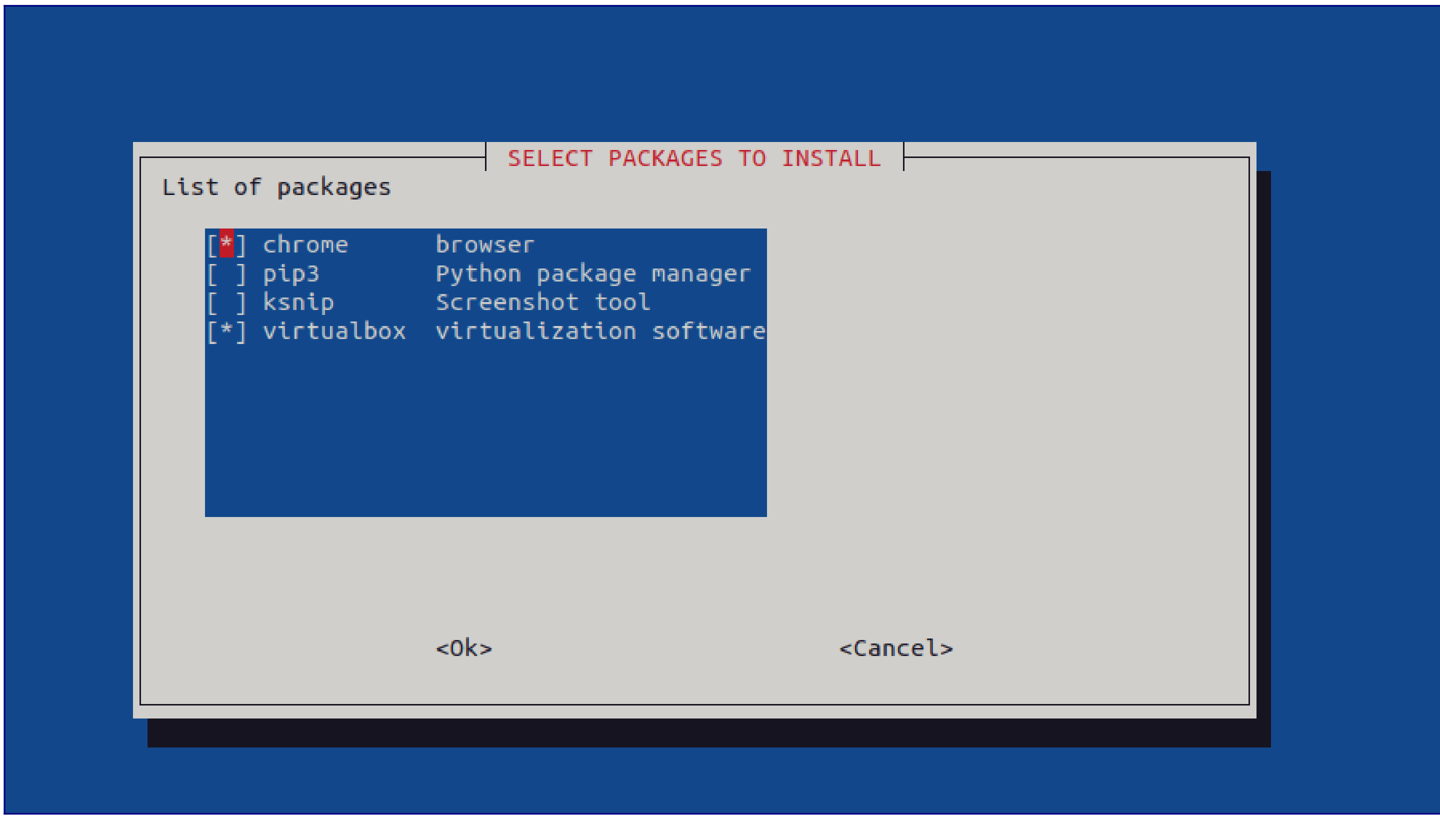
```
--checklist <text> <height> <width> <listheight> [tag item status]...
```

Above is the syntax for creating a checklist dialog box. You have to use the `--checklist` flag followed by setting up the height and width of the dialog box.

The option `<listheight>` specifies how many lists are you going to create. Each list will be tagged with `<status>` which is set to ON or OFF. On points to list selected and Off points to no selection of list.

```
$ whiptail --title "SELECT PACKAGES TO INSTALL" --checklist \
"List of packages" 20 100 10 \
"chrome" "browser" OFF \
"pip3" "Python package manager" OFF \
"ksnip" "Screenshot tool" OFF \
"virtualbox" "virtualization software" ON
```

To select a list press space bar and use up and down arrows to move between the list. Once done press enter.



```
SELECT PACKAGES TO INSTALL
List of packages

[*] chrome      browser
[ ] pip3        Python package manager
[ ] ksnip       Screenshot tool
[*] virtualbox  virtualization software

<Ok>                     <Cancel>
```

You can store the output to an array and later use it. The tag ("Chrome, pip3, ksnip, virtualbox") names will be printed as output to stderr based on the selection.

```
SELECTED=($(whiptail --title "SELECT PACKAGES TO INSTALL" --checklist \
"List of packages" 20 100 10 \
"chrome" "browser" OFF \
"pip3" "Python package manager" OFF \
"ksnip" "Screenshot tool" OFF \
"virtualbox" "virtualization software" ON 3>&1 1>&2 2>&3))

$ echo ${SELECTED[@]} # Array of values
```

Sample output:

```
"pip3" "ksnip" "virtualbox"
```

If you have no idea about bash arrays, we have a detailed article about bash array. I suggest you take a look at the link below.

- **Bash Scripting - Indexed Array Explained With Examples**

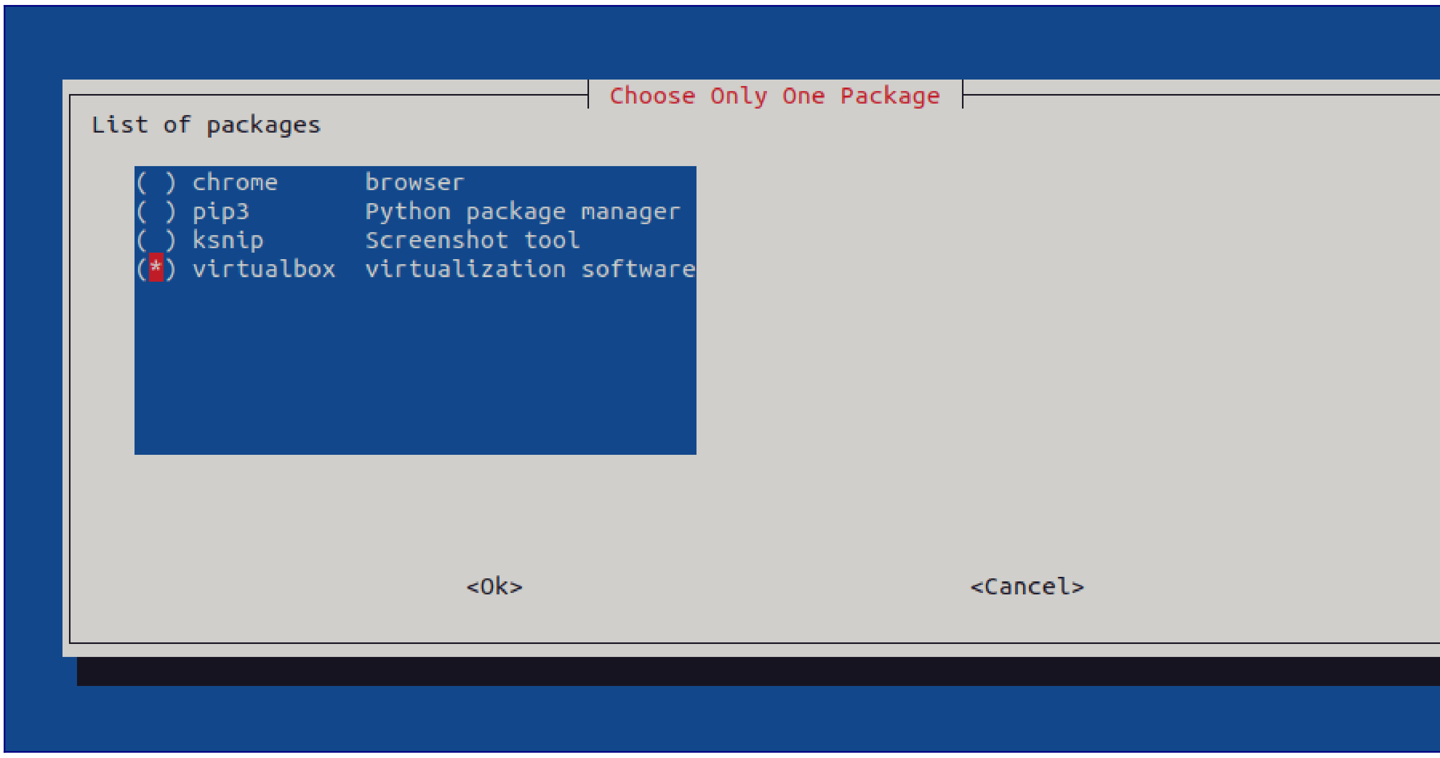
9. Radio List Dialog Box

Radio list dialog box is similar to the checklist dialog box but the only difference is you can only choose one option from the list. Syntactically both the radio list and checklist are the same.

```
--radiolist <text> <height> <width> <listheight> [tag item status]...
```

Example:

```
SELECTED=$(whiptail --title "Choose Only One Package" --radiolist \
"List of packages" 20 100 10 \
"chrome" "browser" OFF \
"pip3" "Python package manager" OFF \
"ksnip" "Screenshot tool" OFF \
"virtualbox" "virtualization software" OFF 3>&1 1>&2 2>&3)
```



```
Choose Only One Package
List of packages

( ) chrome       browser
( ) pip3         Python package manager
( ) ksnip        Screenshot tool
(*) virtualbox   virtualization software
```

```
$ echo $SELECTED
virtualbox
```

10. Menu Dialog Box

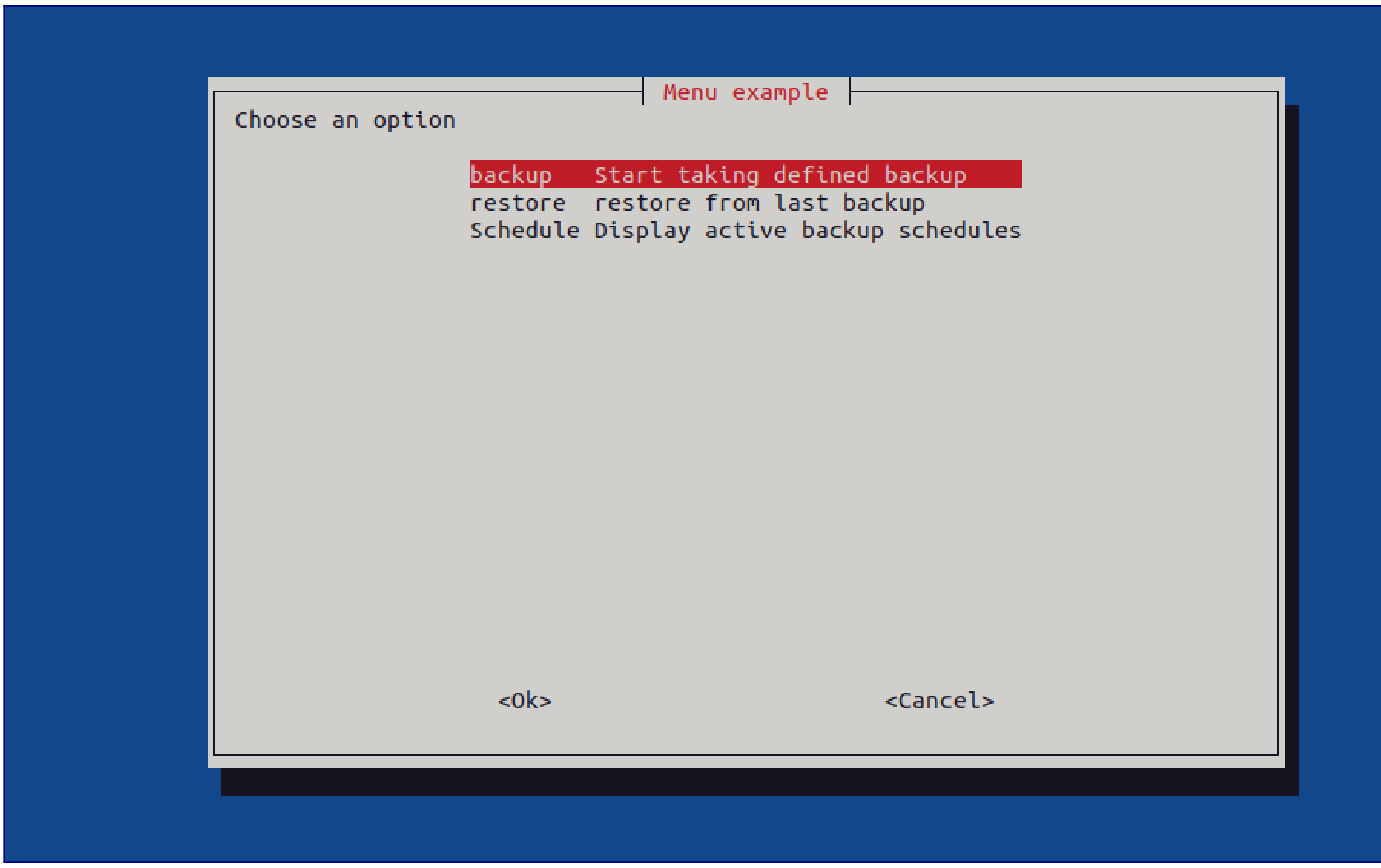
Menu dialog box is similar to the radio button dialog box. The only difference I feel is, in the radio button dialog box you have to press <space-bar> to select an item from the list then press enter. But in the menu dialog box, all you have to do is just press enter which will return the tag name to stderr.

The syntax is similar to the checklist and radio button but the only difference is there is no need for the "status" option to set ON/OFF in the menu dialog box.

```
--menu <text> <height> <width> <listheight> [tag item]
```

Example:

```
TO_RUN=$(whiptail --title "Menu example" --menu "Choose an option" 25 78 5 \
"backup" "Start taking defined backup" \
"restore" "restore from last backup" \
"Schedule" "Display active backup schedules" 3>&1 1>&2 2>&3)
```



```
                          Menu example
 Choose an option

               backup   Start taking defined backup
               restore  restore from last backup
               Schedule Display active backup schedules

                  <Ok>                        <Cancel>
```

```
$ echo $TO_RUN
backup
```

11. Progress Bar

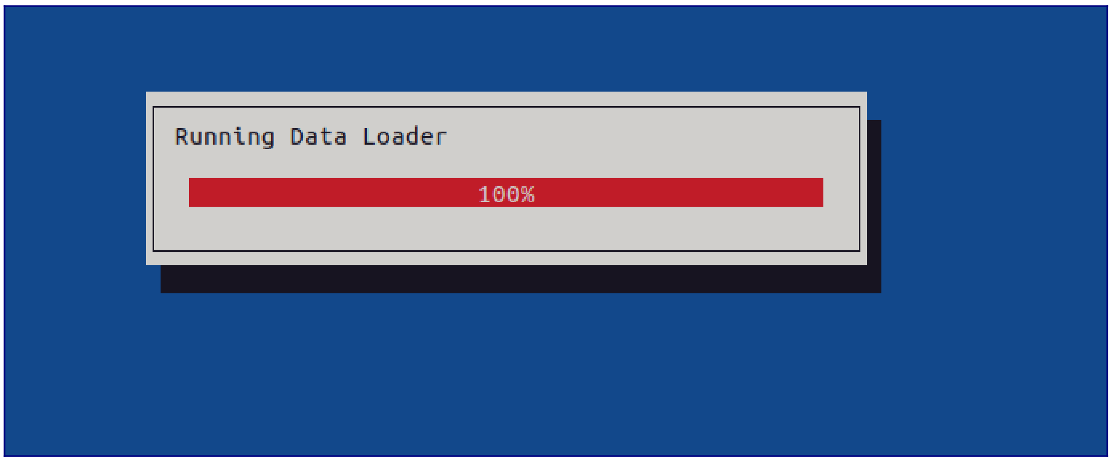
To create a progress bar you have to use the following syntax. First, you will pass a text which will be printed when the progress bar is running and set the height and width of the window followed by the percentage of the progress.

```
--gauge <text> <height> <width> <percent>
```

The progress percentage will be controlled by our logic. Take a look at the below snippet. I am redirecting the while loop to the progress bar and incrementing the COUNTER variable in 10 counts which will be used as the percentage of progress.

```
#!/usr/bin/env bash

COUNTER=0
while [[ ${COUNTER} -le 100 ]]; do
  sleep 1
  COUNTER=$(($COUNTER+10))
  echo ${COUNTER}
done | whiptail --gauge "Running Data Loader" 6 50 ${COUNTER}
```

The progress bar will increment in the count of 10’s.



```
Running Data Loader
100%
```

Conclusion

We have reached the end of this article. Here, we have briefly seen how to use Whiptail to create various dialog boxes in bash scripts. If you have already used whiptail and have any trick under the slave, share it with us through the comment box.

Resource:

- **Whiptail**

Bash scripting guides:

- **How To Use Date Command In Bash Scripting In Linux**
- **How To Debug Bash Scripts In Linux And Unix**
- **Bash Scripting – Parse Arguments In Bash Scripts Using getopts**
- **How To Create GUI Dialog Boxes In Bash Scripts With Zenity In Linux And Unix**
- **Bash Scripting – Case Statement**
- **Bash Scripting – Conditional Statements**
- **Bash Scripting – String Manipulation**
- **Bash Scripting – Printf Command Explained With Examples**
- **Bash Scripting – Indexed Array Explained With Examples**
- **Bash Scripting – Associative Array Explained With Examples**
- **Bash Scripting – For Loop Explained With Examples**
- **Bash Scripting – While And Until Loop Explained With Examples**
- **Bash Redirection Explained With Examples**
- **Bash Scripting – Variables Explained With Examples**
- **Bash Scripting – Functions Explained With Examples**
- **Bash Echo Command Explained With Examples In Linux**
- **Bash Heredoc Tutorial For Beginners**

BASHBash ScriptBash scriptingBash shellBash tipsBash tutorialCLICommand lineDialog BoxLinuxShell scriptingshell scriptsWhiptail
0 comment
6

Karthick

Karthick is a passionate software engineer who loves to explore new technologies. He is a public speaker and loves writing about technology especially about Linux and opensource.

Previous post

How To Deploy Kubernetes Cluster On AWS With Amazon EKS

Next post

How To Add Worker Nodes To Amazon EKS Cluster

You May Also Like

Leave a Comment

☐Save my name, email, and website in this browser for the next time I comment.

* By using this form you agree with the storage and handling of your data by this website.

Social Networks

Facebook Twitter Linkedin Reddit

Newsletter

Subscribe our Newsletter for new posts. Stay updated from your inbox!

WE RESPECT YOUR PRIVACY.

How To Deploy Kubernetes Cluster On AWS With Amazon EKS

Creating a Kubernetes Cluster with Elastic Kubernetes Service

Written by M.S.M. Sivam **Published: Last Updated on** *403* Views
0 comment
4

In this article, we are going to learn about how to deploy a Kubernetes Cluster on AWS with **Amazon EKS** and how to install and configure AWS CLI and Kubectl to interact with the EKS cluster from commandline in Linux.

Before deploying Kubernetes cluster on AWS cloud using Amazon Elastic Kubernetes Service, make sure you have an AWS account. If you don't have an AWS account, check our **Introduction to Amazon Web Services (AWS)** article to know how to create one.

1.

   1.
2.

1. Setup EKS Cluster (Master Node)

To create an EKS cluster in AWS, you need to have an IAM role created in hand.

1.1. Create IAM role

Amazon EKS-managed Kubernetes clusters make calls on your behalf to other AWS services to manage the resources you utilize with the service.

You must first create an IAM role with the following IAM policy before you may construct Amazon EKS clusters:

Login to your AWS console and search for 'IAM'. Select the IAM service to get into IAM Console.

Choose Roles in the left side options and then click Create role.

Select AWS services and select 'EKS – Cluster' in the Use cases options. Once selected, click 'Next' to proceed.

Make sure 'AmazonEKSClusterPolicy' is added, by default it will be added as we selected EKS-Cluster in the use cases. Click Next to proceed further.

In the next step, set the role name. Here, we are naming the role as 'ostechnix_eks'. Review all the parameters and click 'create' at the end to create the role.

Here we don’t add any tags for this role. If we are dealing with multiple resources, it will be useful having tags to manage, identify and filter resources.

A new role named 'ostechnix_eks' has been just created.

1.2. Create EKS Cluster

Go to AWS Console and search with 'EKS'. Select the 'Elastic Kubernetes Service' to get into the EKS console.

From the 'Add cluster' drop down box, choose 'create' cluster option.

You will get the ‘Configure Cluster’ page where you can name the cluster, select the Kubernetes version and select the cluster service role that we created in the previous step.

Here, we named the cluster as 'ostechnix', and selected the Kubernetes version 1.21.

If you did not find the role, refresh the roles. Select the role and click 'next' to proceed.

In this ‘Specify Networking’, you need to configure the networking. Here we are proceeding with default options.

Select the existing VPC, VPC is Virtual Private Cloud where you can create AWS resources in the Virtual Network that you have defined. Proceed with the default subnets available in the default VPC.

Choose IPv4 as the Cluster IP address family. It is the default one.

Choose 'Public' for Cluster End Point access which enables only public access to your cluster. If you choose 'Private', it enables only private access to your cluster. Here, we are proceeding with Public which is the default one.

You can proceed with the default 'Networking Add-ons' and click 'Next'.

You will get 'Configure logging' page where you can select which log types that you want to enable. By default, all the types are disabled. Click 'Next' to proceed.

You will get 'Review and Create' page. Review all the details we configured and click 'Create' at the bottom.

Cluster creation will be in progress. It will take couple of minutes to get created.

A new EKS Cluster named 'ostechnix' is created. You can verify in AWS Console□Amazon EKS□Clusters.

Next, we need to install and configure AWS CLI and Kubectl to interact with the EKS cluster from commandline.

To configure AWS CLI credentials, you need to create security credentials in AWS IAM.

2. Create Security Credentials

Log into AWS console and search with IAM. Select IAM to get into the IAM console.

Select 'My security credentials' option available in the right to create and manage your security credentials.

Click on 'Access Keys' drop down box and click 'Create New Access Key' option.

Once you click the 'Create New Access Key' option, the key will be created. Download the key to configure AWS CLI in Linux machine. You can view the key by checking 'Show Access key'.

In this demonstration the key is,

- **Access Key ID:** AKIAV7XU2AIJBX4EYKUO
- **Secret Access Key:** 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

3. Install AWS CLI

AWS CLI is a command-line interface that brings all AWS services together in a single terminal, allowing you to operate numerous AWS services with a single tool.

Use the below `curl` command to download the installation file. Here, we are using 'CentOS Stream' to install and configure AWS CLI.

```
[root@ostechnix ~]# curl "https://awscli.amazonaws.com/awscli-exe-linux-
x86_64.zip" -o "awscliv2.zip"
```

Unzip the installer file using the below command.

```
[root@ostechnix ~]# unzip awscliv2.zip
```

Install AWS CLI using the below command.

```
[root@ostechnix ~]# ./aws/install
You can now run: /usr/local/bin/aws --version
```

Verify the version using the above mentioned command.

```
[root@ostechnix ~]# /usr/local/bin/aws --version
aws-cli/2.4.17 Python/3.8.8 Linux/4.18.0-358.el8.x86_64 exe/x86_64.centos.8
prompt/off
```

4. Configure AWS CLI

Use the below command to configure the AWS CLI. It will ask for the Access Key ID and Secret Access Key that we generated in section 2.

```
[root@ostechnix ~]# /usr/local/bin/aws configure
AWS Access Key ID [None]: AKIAV7XU2AIJBX4EYKUO
AWS Secret Access Key [None]: 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
Default region name [None]:
Default output format [None]:
[root@ostechnix ~]#
```

5. Install Kubectl

Kubernetes communicates with the cluster API server via the kubectl command line utility.

Use the below curl command to download Amazon EKS vended kubectl binary from Amazon S3.

```
[root@ostechnix ~]# curl -o kubectl https://amazon-eks.s3.us-west-
2.amazonaws.com/1.21.2/2021-07-05/bin/linux/amd64/kubectl
```

Add execute permission to the binary using below command.

```
[root@ostechnix ~]# chmod +x ./kubectl
```

Copy this binary to the folder in your path and export the `$PATH`.

```
[root@ostechnix ~]# mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl &&
export PATH=$PATH:$HOME/bin
```

Export the PATH to `~/.bashrc`:

```
[root@ostechnix ~]# echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc
```

Kubectl is installed, you can verify the version using the below command.

```
[root@ostechnix ~]# kubectl version --short --client
Client Version: v1.21.2-13+d2965f0db10712
```

6. Configure Kubectl

Right now kubectl does not know where the EKS Master node is located in AWS. We need to let kubectl know where the Master server is located by mentioning the **region** name and **cluste**r name.

Use the below command to check the status of the EKS Cluster.

```
[root@ostechnix ~]# /usr/local/bin/aws eks --region ap-south-1 describe-cluster
--name ostechnix --query cluster.status
"ACTIVE"
```

Update the kubeconfig file to use kubectl to interact with the EKS cluster. It will fetch all the configurations from Master node to kubeconfig file.

```
[root@ostechnix ~]# /usr/local/bin/aws eks --region ap-south-1 update-kubeconfig
--name ostechnix
Added new context arn:aws:eks:ap-south-1:411756528146:cluster/ostechnix to
/root/.kube/config
```

Verify the kubectl by checking the service using the below command. You can ensure the Kubectl is able to connect with the EKS cluster.

```
[root@ostechnix ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.100.0.1   <none>        443/TCP   105m
```

Conclusion

In this article, we have learned how to provision Amazon EKS cluster and how to setup AWS CLI and Kubectl for the EKS cluster in Linux platform.

I've successfully deployed Kubernetes cluster on AWS cloud, now what? You might wonder. Please check our next guide to know how to add Node Groups and configure the worker nodes in AWS EKS cluster.

- **How To Add Worker Nodes To Amazon EKS Cluster**

Resource:

- **https://docs.aws.amazon.com/eks/latest/userguide/create-cluster.html**

Amazon EKSAmazon Elastic Kubernetes ServiceAWSAWS CLIAWS CloudCLICommand lineDevOpsEKSElastic Kubernetes ServiceIT AutomationK8sKubeKubectlKubernetesKubernetes clusterKubernetes seriesKubernetes tutorialLinuxOpen sourceVirtualization

0 comment

4

M.S.M. Sivam

Solution Architect - BigData Infrastructure in one of the Leading MNC. Highly passionate about BigData and Cloud Technologies.

Previous post

Introduction To Amazon EKS (Elastic Kubernetes Service)

Next post

How To Create GUI Dialog Boxes In Bash Scripts With Whiptail In Linux

You May Also Like

Leave a Comment

☐Save my name, email, and website in this browser for the next time I comment.

* By using this form you agree with the storage and handling of your data by this website.

Introduction To Amazon EKS (Elastic Kubernetes Service)

Kubernetes On AWS

Written by M.S.M. Sivam **Published: Last Updated on** *648* Views
0 comment
4

Amazon Cloud (AWS) offers a number of services that aid with container orchestration, including Amazon Elastic Container Service (ECS), Amazon Elastic Kubernetes Service (EKS), Amazon LightSail, and Amazon Elastic Container Registry (ECR). In this article, we will learn about Amazon EKS, which is Kubernetes in AWS cloud.

1.

    2.

2.

1. What is Amazon EKS?

Amazon EKS (Amazon Elastic Kubernetes Service) is a managed service that allows you to run Kubernetes on AWS Cloud without having to set up, administer, or maintain your own control plane and nodes.

Kubernetes is an open-source technology that automates the deployment, scaling, and management of containers, i.e. containerized Applications.

To achieve high availability, EKS runs and scales the Kubernetes control plane across different AWS Availability Zones. In Amazon EKS, control plane instances are automatically scaled based on load, and unhealthy control plane instances are detected and replaced, as well as automated version upgrades and patching will be done automatically.

Amazon EKS can be integrated with other AWS services to provision various facilities. For example, worker nodes can be provisioned by AWS EC2 Instances, ECR (Elastic Container Registry) for Container Images, VPC (Virtual Private Cloud) for isolating resources.

2. Amazon EKS Cluster Components

Amazon EKS cluster consists of two important components.

1. EKS Control Plane
2. EKS Nodes

2.1. EKS Control Plan

The Amazon EKS control plane is made up of nodes that execute Kubernetes software like **etcd** and the **Kubernetes API server**. The control plane operates on an AWS account, and the Kubernetes API is accessible through the Amazon EKS endpoint for your cluster. Each Amazon EKS cluster control plane has its own set of Amazon EC2 instances and is single-tenant and unique.

EKS Control plane is available across several availability zones; if any of the control planes has a problem, EKS automatically identifies and replaces those unhealthy control plane nodes, as well as providing on-demand, zero-downtime updates, and patching.

2.2. EKS Nodes

Amazon EKS nodes run in your AWS account and connect to the control plane of your cluster through an API server endpoint and a certificate file issued for your cluster. Node Groups should be created to provision the nodes in the EKS cluster.

A node group is made up of one or more nodes. In an Amazon EC2 Auto Scaling group, a node group is made up of one or more Amazon EC2 instances and all the instances must be the same type with the same Amazon Machine Image (AMI). And, a Node Group should use the same IAM role.

3. EKS Cluster Deployment Methods

You can create the EKS Cluster in two ways.

- **Amazon EKS - eksctl:** It is a simple command line utility for constructing and maintaining Kubernetes clusters. AWS and Weaveworks collaborated on **eksctl**, a tool that automates much of the process of setting up EKS clusters. Eksctl also supports resource provisioning through a config file, which is the preferable method because it allows you to version control your EKS cluster configuration.
- **AWS Management Console and AWS CLI:** This is the easiest method to deploy Amazon EKS cluster where you can launch EKS as service in AWS and add nodes by creating Node Groups in the AWS console itself.

4. How Amazon EKS Works

The easiest way of considering Amazon EKS is 'Kubernetes as a Service' by AWS Cloud.

As mentioned above, Amazon EKS consists of two main components; EKS Control Plane/Master and Data plane/Worker nodes which are building the EKS cluster. Both the planes are run in their own Virtual Private Clouds (VPCs).

The nodes in VPCs are in charge of running the container images or workloads. AWS also offers the networking infrastructure needed to connect these components and form a Kubernetes cluster.

Pods can be scheduled on any mix of self-managed nodes, Amazon EKS controlled node groups, and AWS Fargate in an Amazon EKS cluster.

Amazon EKS nodes run under your account and use the cluster's API server endpoint to communicate with the control plane.

The following details and diagram illustrate the deployment of applications in the EKS Cluster.

Provision Amazon EKS Cluster - Provisioning Master

You can create the Cluster in AWS Management Console or using AWS CLI or one of the AWS SDKs

Deploy Compute - Provisioning Worker

You can launch compute nodes from AWS Fargate or Amazon EC2 Instances. To setup worker nodes for executing application containers, EKS provides the following options.

- **Self-Managed:** The user is responsible for provisioning EC2 instances that are linked to the cluster. This provides you more options when it comes to setting worker nodes.
- **Managed:** For Amazon EKS Kubernetes clusters, managed node groups automate the provisioning and lifecycle management of nodes (Amazon EC2 instances).
- **AWS Fargate:** Fargate is an AWS-managed serverless computing engine that allows you to execute container applications without having to maintain servers.

Connect with Amazon EKS Cluster

Kubernetes communicates with its cluster via a command-line tool named Kubectl. You need to install kubectl in your machine and configure it to connect Amazon EKS and run Applications.

Monitor Kubernetes Apps

After creating your complete EKS Cluster, you have to deploy Kubernetes dashboard which is Web Based Management Interface to manage and monitor your EKS Cluster.

5. Amazon EKS Features

Here I have listed some important features of Amazon EKS.

Managed Control Plane

Amazon EKS provides a highly available control plane with automatic scalability option. As EKS is running in three Availability Zones, it automatically detects the unhealthy control plane and replaces it.

Service Integration

AWS Controllers for Kubernetes (ACK) allows you to manage AWS services directly from your Kubernetes environment. Using AWS resources, ACK makes it simple to construct scalable and highly available Kubernetes apps.

Eksctl - Single line Management

It is a command line tool which can be installed in your Windows or Linux machine to create, run, and manage your EKS cluster. It simplifies the Cluster Management and Operations.

Security

Amazon EKS integrates with various Services and Technologies to provide the highly secured environment. For example, IAM enables fine grained access control and VPC isolates and protects your EKS cluster from third party access.

Load Balancing

Amazon EKS supports using Application Load Balancer which is ideal for advanced load balancing of HTTP and HTTPS traffic. Also, EKS utilizes Network Load Balancer and Classic Load Balancer.

Serverless Compute

To execute your Kubernetes apps utilizing serverless computing, EKS supports AWS Fargate. Fargate eliminates the need to build-up and maintain servers. It allows you to choose and pay for resources per application.

Hybrid Deployment

EKS on 'AWS Outposts' can be used to execute containerized apps with minimal latencies to on-premises systems. AWS Outposts is a fully managed solution that connects any connected location with AWS infrastructure, services, APIs, and tools. You can manage containers on-premises with the same simplicity that you can manage them in the cloud with EKS on Outposts.

Open-Source Compatibility

Amazon EKS supports common Kubernetes ad-ons and EKS is highly compatible with Kubernetes Community tools.

Managed Cluster updates

Version upgrades for Kubernetes are performed on-the-fly, eliminating the need to construct new clusters or transfer apps to a new cluster.

6. Amazon EKS Pricing

For each Amazon EKS cluster you build, you pay **$0.10 per hour**. By utilizing Kubernetes namespaces and IAM security settings, you may run many apps on a single EKS cluster.

On AWS, you may use Amazon Elastic Compute Cloud (Amazon EC2) or AWS Fargate to operate EKS, and on-premises, you can use AWS Outposts. Accordingly, the price will be calculated.

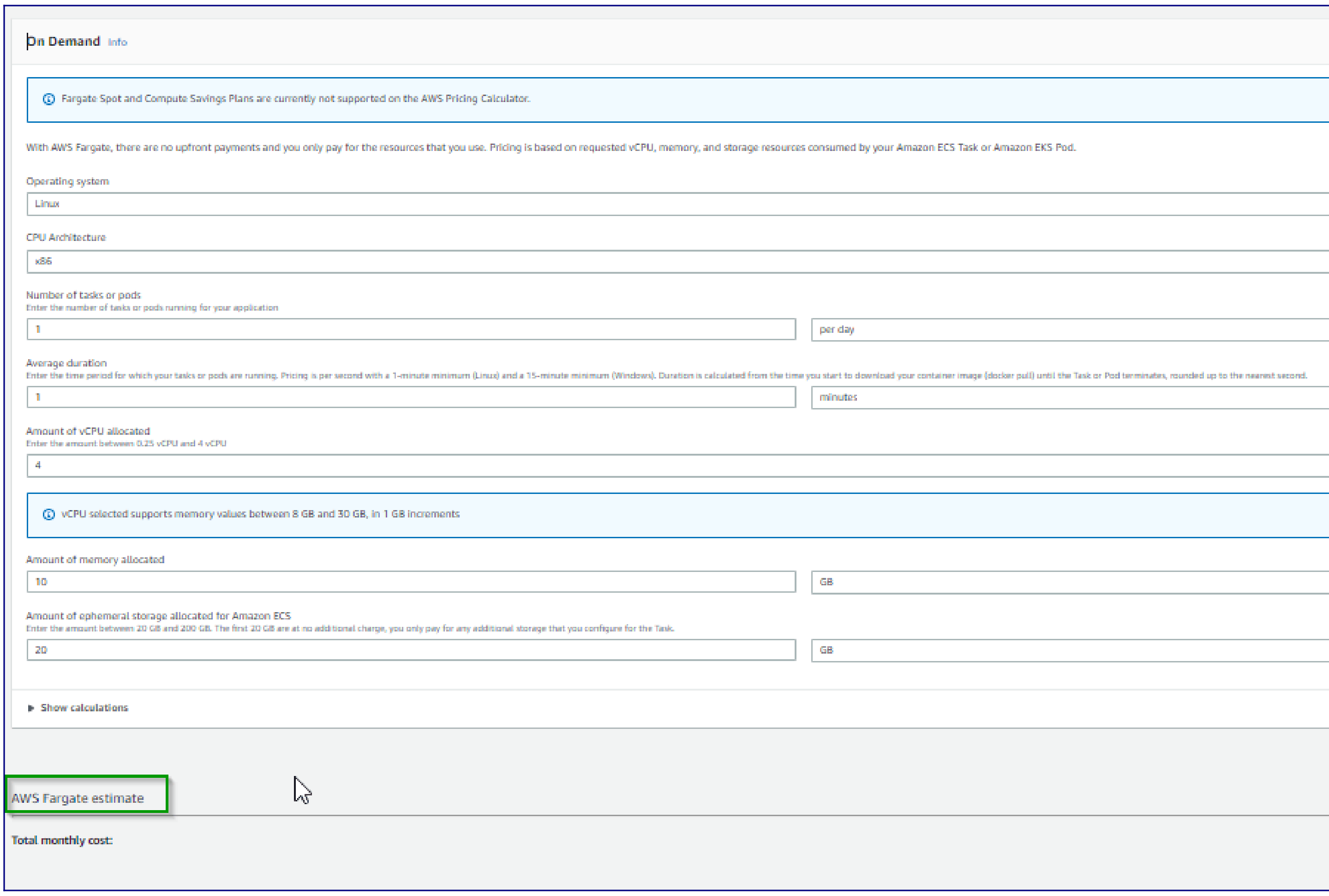
AWS provides the **calculator** to estimate the price. You can use the below link to estimate your price for EKS Cluster.

Refer the below article for more details about AWS EC2 instance pricing model as you can use EC2 instances for EKS worker nodes.

- **Amazon Web Services (AWS) Introduction**

If you utilize AWS Fargate, price is dependent on the amount of vCPU and RAM consumed from the time you start downloading your container image until the Amazon EKS pod finishes, time calculation will be rounded up to the closest second. Minimum charge will be applied for **one minute** even if you use 10 seconds.

Use **this link** to estimate your AWS Fargate pricing. Here you need to fill in the details like OS Model, Number of PODs, amount of vCPU, Memory.


On Demand Info

Fargate Spot and Compute Savings Plans are currently not supported on the AWS Pricing Calculator.

With AWS Fargate, there are no upfront payments and you only pay for the resources that you use. Pricing is based on requested vCPU, memory, and storage resources consumed by your Amazon ECS Task or Amazon EKS Pod.

Operating system
Linux

CPU Architecture
x86

Number of tasks or pods
Enter the number of tasks or pods running for your application
1 per day

Average duration
Enter the time period for which your tasks or pods are running. Pricing is per second with a 1-minute minimum (Linux) and a 15-minute minimum (Windows). Duration is calculated from the time you start to download your container image (docker pull) until the Task or Pod terminates, rounded up to the nearest second.
1 minutes

Amount of vCPU allocated
Enter the amount between 0.25 vCPU and 4 vCPU
4

vCPU selected supports memory values between 8 GB and 30 GB, in 1 GB increments

Amount of memory allocated
10 GB

Amount of ephemeral storage allocated for Amazon ECS
Enter the amount between 20 GB and 200 GB. The first 20 GB are at no additional charge, you only pay for any additional storage that you configure for the Task.
20 GB

▸ Show calculations

AWS Fargate estimate

Total monthly cost:

On AWS Outposts, you can construct and run your Amazon EKS nodes. AWS Outposts allows on-premises facilities to use native AWS services, infrastructure, and operating models.

The cost for Amazon EKS on AWS Outposts is straightforward and identical to that of the Amazon EKS cluster deployed in AWS, and you pay **$0.10 per hour**.

You can purchase different types of AWS Outposts rack configurations according to your requirement, rack configuration is a combination of EC2 instance type, EBS gp2 Volume and S3 on Outposts. Refer **this link** to estimate your pricing model for AWS Outposts.

Conclusion

In this article, we have understood the basic concepts of Amazon EKS, its working & pricing model and its important features. We will see the Amazon EKS Cluster building procedures in the upcoming article.

https://ostechnix.com/introduction-to-amazon-eks-elastic-kubernetes-service/