Mastering Linux, part 11

In this month's instalment of the Mastering Linux series, *Jarrod Spiga* delves deeper into scripting and shows you how to customise the X-Windows desktop.

On the cover DVD

Bonus DVD software

PDFs of every instalment of the Mastering Linux series.

Skill level

Intermediate

Requirements

An installation of Linux (Fedora Core 3 was used in the writing of this article)

Time to complete

2 hours

The scripting methods demonstrated in last month's instalment of the Mastering Linux series are fine if you need to execute commands in a strictly linear fashion. What wasn't shown was how to use control structures and conditional statements to alter the behaviour of your script under certain circumstances.

IF THEN ELSE

The `if-then-else` control structure will be familiar to anyone who has done any kind of programming or scripting. Shell scripts also support this most basic of control structures. Broken down, this control structure executes certain commands if a condition is true, otherwise it executes another set of commands:

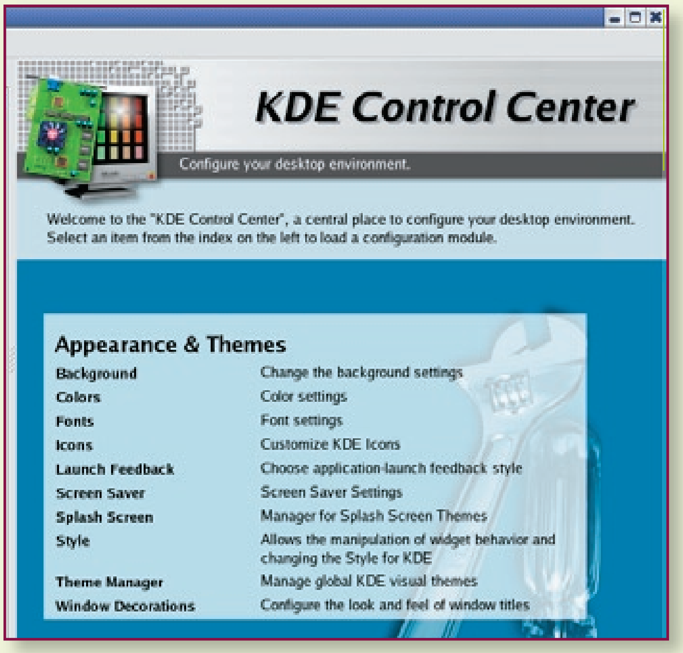
```
if [ $CONDITION = alpha ]; then
echo "This code is executed if $CONDI-
TION is identical to alpha."
elif [ $CONDITION = beta ]; then
echo "This code is executed if $CONDI-
TION is identical to beta."
else
echo "This code is executed if $CONDI-
TION is neither alpha or beta."
fi
```

This example shows how if-then-else structures work. Note the location of the brackets (`[ ]`), which are there to tell the shell what the test expression is. They also stop the shell from assigning the test value to the variable. Also note the required semi-colon (`;`) after the test expression.

The test expression is not strictly limited to "identical" comparisons. The table below shows some of the other expressions that are frequently used:

Expression	Definition
x = y	True if the text value of x is identical to the text value of y
x != y	True if the text value of x is not identical to the text value of y
x -gt y	True if the numeric value of x is greater than the numeric value of y
x -lt y	True if the numeric value of x is lesser than the numeric value of y

Customising the desktop



Welcome to the "KDE Control Center", a central place to configure your desktop environment. Select an item from the index on the left to load a configuration module.

Appearance & Themes

Background	Change the background settings
Colors	Color settings
Fonts	Font settings
Icons	Customize KDE Icons
Launch Feedback	Choose application-launch feedback style
Screen Saver	Screen Saver Settings
Splash Screen	Manager for Splash Screen Themes
Style	Allows the manipulation of widget behavior and changing the Style for KDE
Theme Manager	Manage global KDE visual themes
Window Decorations	Configure the look and feel of window titles

1 *The KDE Control Center is the one-stop shop for all of your settings customisations.*

By now, you've probably customised the way your desktop looks and behaves. After all, the default settings won't suit the way that everyone uses their Linux systems. Some customisation is required in order for you to work more efficiently within X-Windows — and it doesn't hurt to jazz up the look of things while you're at it.

If you use GNOME, most of the settings for your desktop can be found under the GNOME > Preferences menu. From KDE, click KDE > Control Center. 1 Instead of displaying your options in a menu, KDE's Control Center has an expandable tree that contains all of your customisation options.

WINDOW APPEARANCE

One of the quickest ways to make a drastic alteration to your desktop's looks is to change the theme currently being applied to the Window Manager. The theme preferences dialog can be brought up by double-clicking on the Theme icon in GNOME's Control Center, or under Appearance and Themes > Window Decorations if you're using KDE.

A decent range of decorations is installed by default, and it's even possible to combine elements from a couple of different themes to display at once. This is fairly intuitive if you're using KDE — especially if you also browse through the various colour schemes under Appearance & Themes > Colors. But if GNOME is your

x -eq y	True if the numeric value of x is equal to the numeric value of y
-e value	True if a filesystem object by the name of value exists
-f value	True if a file by the name of value exists (not a directory, link, etc)
-d value	True if a directory by the name of value exists
-x value	True if the file by the name of value has executable permissions

The optional `elif` command can also be used to modify the test expression to check if it meets some different criteria, and is an abbreviation of `else-if`. Similarly, the `else` command is also optional and the code contained within this sub-structure will only be executed if no other test expression has been `true`.

You must include the `fi` command at the end of each structure, as this informs the shell that the structure is closed.

To make your script easier to read and debug, it's a good idea to use indents to separate sections of the control structure, especially when working with multiple, nested if-then-else control structures. Nested structures give you more control, but get increasingly complicated to work with as your script grows — especially if you forget the fi command in one of your structures. It's usually more efficient to use case control structures.

CASE BY CASE

The case control structure works by comparing many values to a variable. If a match is found, the set of commands in that branch of the structure are executed, and then the structure is terminated. If no match is found, the default case (if one exists) is assumed true. The case structure looks like the following:

```
case "$VALUE" in
'0')
echo "This is executed if $VALUE is
identical to 0";
;;
'1')
echo "This is executed if $VALUE is
identical to 1";
;;
*)
echo "This is executed as the
default case if no other test holds
true";
;;
esac
```

Case structures are ideal when writing a script that manages services or daemons. For instance, the following snippet of code will start or stop a daemon by calling one of two different scripts. It will also inform the user of the correct syntax for the script should the user enter incorrect arguments:

```
#!/bin/bash
case "$1" in
'start')
~/scripts/startup-script
;;
'stop')
~/scripts/shutdown-script
;;
*)
echo "Usage: $0 [start|stop]"
;;
esac
```

AROUND IN LOOPS

The most basic of loop structures is the `while` loop. It's constructed in a similar manner to if-then-else, but it continues to loop through the list of commands in the structure while the test condition holds true:

```
while [ $CONDITION = true ]; do
echo "This code is executed in the
while loop."
# don't forget to include code that
modifies the value of $condition
```

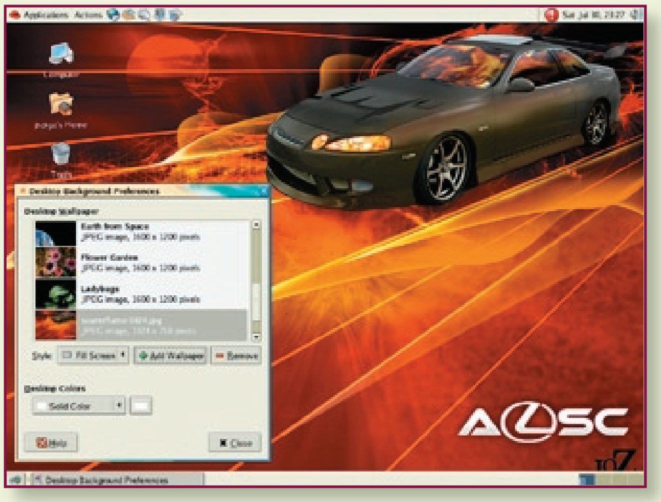
environment of choice, it's a little tricky. It involves you selecting a base theme and then clicking on the Theme Details button. From there, you can choose different themes for your controls, window borders and icons.

If you can't find a theme that you truly like, head on over to **http://themes.freshmeat.net** and download some of the user-contributed themes. Installation instructions are usually supplied with each theme.

WALLPAPER

2 Everyone likes to have their own desktop wallpaper displayed beneath their icons and windows. If you're using GNOME, you can change your wallpaper by selecting Desktop Background. To select your background image, click on the button underneath where it says Select picture. You will then see the list of images stored at /usr/share/backgrounds/images, but it's pretty easy to navigate to the location where your desired background is stored.

In KDE, the settings are under Appearance & Themes > Background and you can select your background image from under the Wallpaper tab. KDE also gives you the choice of setting a different wallpaper for each of the virtual desktops that are running — simply deselect the Common background checkbox. You can also specify multiple wallpapers if you wish, and KDE will automatically change the background at a frequency that you can define under the Setup Multiple. . . button.



2 *The default storage location for wallpapers is /usr/share/background/images. To add images from other locations, hit the Add Wallpaper button.*

RESOLUTIONS AND REFRESH

If you're using a CRT screen, you'll generally want to use the highest screen refresh setting the monitor can handle in order to minimise eye-strain while using your Linux system. If you're using an LCD screen, the

```
CONDITION = false
done
```

The comment line is an important reminder — if you don't include any code that modifies the test variable(s) within your while structure, the script will get stuck in the loop and continue to execute forever (or until you manually stop it). Using a nested if-then-else structure in the while structure is a good way to determine whether the test variables actually need changing or not.

BREAK THE CIRCLE

Conditions will undoubtedly arise when you'll want to stop a portion of a loop from executing, or even stop the loop from processing altogether. The `continue` statement can be used to cause the loop to stop executing that iteration and proceed with the next. The break statement can be used to immediately terminate the loop.

The following commands should echo the numbers 1, 2 and 4 on the screen.

```
val = 0
while [ "$val" -lt 5 ]; do
val = $(($val+1))
if [ "$val" -eq 3 ]; then
continue
fi
echo -n "$val "
if [ "$val" -eq 4 ]; then
break
fi
done
```

The number 3 is not shown because the continue statement stops the loop from executing the echo command, forcing the loop to start processing the next iteration. The number 5 is not shown because the `break` statement terminates the processing of the loop before the while statement gets to terminate.

WORKING WITH SETS

The third type of loop that you'll encounter in your shell-scripting endeavours is the `for-do` loop. This is used when you need to perform a number of commands for every element in a set and is ideal for performing operations on a group of files, such as:

```
for FILENAME in *.txt; do
echo "Now working on $FILENAME.."
tail -10 "$FILENAME"
done
```

When executed, the `for` statement assigns the first value in the set to the variable: in this case, the first `.txt` file is assigned to the `FILENAME` variable. The `done` statement lets the loop iterate, allowing for the next value to be assigned to the variable, and so forth until all of the values in the set have been processed.

And on a final note, there's no need to use a wildcard substitution to create your set in the for-do loop. As this example shows:

```
for FILENAME in 1.txt 2.txt 3.txt;
do
```

you can specify each item in the set as you go. apc

Next month . . .

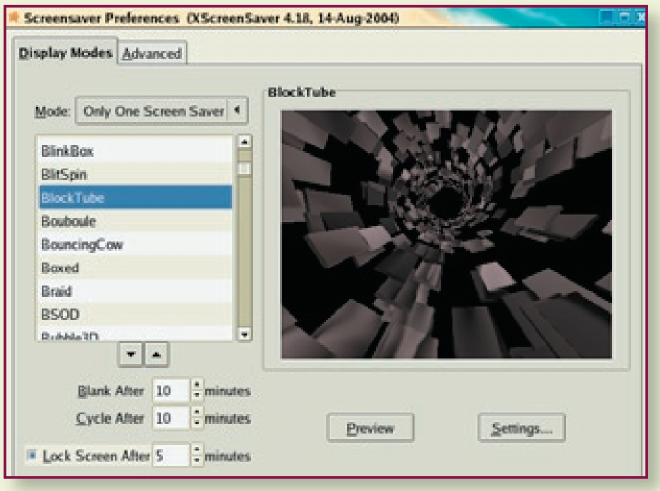
Part 12 of the series will cover Linux Systems Administration, focusing on how to manage various operations running on your Linux system. It will also explain how to check that system services and applications are functioning correctly.

Customising the desktop

resolution should be set to your screen's native resolution for the best picture quality.

The Screen Resolution preference within the GNOME menu allows you to customise both of these settings. Under KDE, the same settings can be found under Desktop > Size & Orientation.

If you select an unsupported resolution and refresh-rate combination using GNOME, hit Enter after applying your settings to reset them. Under KDE, waiting 15 seconds will action the settings reset.



Mode: Only One Screen Saver

BlinkBox
BlitSpin
BlockTube
Bouboule
BouncingCow
Boxed
Braid
BSOD

BlockTube

Blank After 10 minutes
Cycle After 10 minutes
Lock Screen After 5 minutes

3 *For security reasons, it's wise to lock your screen shortly after your screensaver becomes active.*

SCREENSAVERS

Both desktop environments come with dozens of screensavers. While monitor burn-in is much less of a problem nowadays, your screensaver is useful for securing your system should you step away from it for a few minutes.

For instance, if you're logged in to an app using root credentials and you leave your system unattended, it's possible for anyone to use the resources on your system as the root user — even if they otherwise wouldn't have known the root password.

As a result, screensavers are often used to prevent others from seeing what you're doing, and to lock the system when idle. In order to lock your system, ensure that the Lock Screen After or Require Password to Stop option is set (depending on your environment). The amount of time that you specify next to this option relates to the number of minutes that the screensaver is active for before the screen is locked (the amount of idle time required before your screen will lock is the Blank time, plus the Lock Screen time).