Full Circle

THE INDEPENDENT MAGAZINE FOR THE UBUNTU LINUX COMMUNITY

SPECIAL EDITION

Command & Conquer

Volume One

Through Issue 25

COMMAND AND CONQUER

Written by Robert Clipsham

Last month we showed you how to stay safe using a command line. Now that you know this, you can start using it to your advantage! Over the next few issues, we'll show you the basics of file management, which will be of use later on when doing more advanced things.

The first command in this issue will just prove a statement from the start of last month's article, "you are in your home directory". Whenever you see ~ this means your home directory, so to prove this, type (in a terminal):

```
pwd
```

I get this output:

```
$ pwd
/home/robert
```

Of course this will display your home directory, rather than mine. But what use is this to you? What does it mean to be 'in your home directory'? Any commands you execute will be run in the current directory. This won't mean much for now, but it will make all the difference later on. One command that uses your current directory is 'ls', which will give you a list of files in the directory you specify, or in the current directory if you don't specify one.

It's not always useful to be in your home directory though, so let's move away. To do this, we use the change directory command, cd.

```
$ cd ~/Documents
```

If you now type 'pwd', you will see you are now in your documents directory. The '~/' was not needed for that command although it can be a handy shortcut to save time. In this example, you were already in your home directory, so 'cd Documents' would have worked. If you were in another directory though, say '/home/robert/Pictures/2007/December/Christmas' for example, it would take a long time to change to the documents directory without '~/'. However, now that you are in your Documents directory, how do you move back to your Home directory? There are several ways do to this.

```
$ cd
$ cd ..
$ cd ~/
$ cd /home/robert
```

These all do the same thing if you are in your Documents directory. 'cd' with no arguments will always take you to your home folder. 'cd ..' takes you to your previous directory, so here we moved from '/home/robert/Documents' to the previous directory, '/home/robert'. The third uses the ~ shortcut, and can be used with or without the trailing '/'. The final command uses the full path, which will always take you to the exact location, providing it exists.

Now for some time saving! Rather than typing out a long directory like

'~/Pictures/2007/December/Christmas', you can just type the first few letters!

```
$ cd ~/Pi<tab>
```

Replace <tab> with you pressing your tab key, and notice how it automatically changes to Pictures? You can use this technique with most directories to save time.

Troubleshooting

You may have encountered some problems performing these simple commands. Don't worry though, it is probably something very simple. The first problem you may have encountered probably happened when you tried changing to the Documents directory.

```
-bash: cd: documents: No
such file or directory
```

Everything you enter at the command line is case sensitive! *Documents* and *documents* are two completely different directories in the eyes of the terminal, so make sure you have the correct capitalization! You may also have had this error if you don't have a Documents directory because, for example, you have deleted it. The other error you may have encountered is when trying to use tab complete. If your computer gave a beep when you hit tab, it can mean one of two things. The first is that the directory doesn't exist. If the directory doesn't exist, it won't be able to tab complete it! The other possiblity is that you have multiple directories starting with Pi in your home directory. If this is the case then hit tab again, and you will get a list of possible files and directories, so you can type a few more letters and hit tab again. If there are a lot of possible matches, you will see something like:

```
Display all 388 possibilities?
(y or n)
```

Unless you want to see them all, type 'n', then hit enter and type a few more letters to narrow down the number of possible matches.

Robert Clipsham is a self confessed geek, whose hobbies include: programming/scripting, chatting on IRC and not writing his articles on time.

getdeb
software you want

Goal

GetDeb extends the existing software options for Ubuntu (and derived) Linux distributions by providing major updates and software not yet available on the official Ubuntu repositories.

Quality

GetDeb packages are built using Debian/Ubuntu building rules, this reduces development effort and assures the same level of quality. However when new packages are developed or major upgrades are performed we do not follow a strict quality assurance process, this is the accepted cost required to achieve shorter release times. Still with a broader user base problems are quickly identified as fixed. It should also be noted that we do not provide system core packages or major libraries which could cause dependency problems or other major issues, in case you find a broken package recovery should be easily achieved by reinstalling the Ubuntu official package.

www.getdeb.net

COMMAND AND CONQUER

Written by Robert Clipsham

Now that you know how to navigate a command line, find out where you are, and how to list the files in your current directory, we'll show you how to manage your files using a shell. To begin with, open up a terminal and do the following:

```
$ touch foo
$ mkdir bar
```

The first of these commands will create an empty file called 'foo'. This is often useful if you run a web server along with web applications, and are asked to create a file with the given name to prove you have permission to install the application. The second command is used to make directories, and here makes a directory called 'bar'. If you want to make sure these commands worked, you can use the command you learned last issue to see a list of files/directories within your working directory (No, I'm not going to tell you what it is!). Now that we have a file and directory to practice with, it's time to do something with them. The first thing I'll show you is how to make a copy of a file or directory.

```
$ cp foo foo2
```

This will copy the file 'foo' (which we created earlier) to 'foo2'. Now let's move our new file 'foo2' to the 'bar' directory, and at the same time rename it to 'foo'. To do this, we use the mv tool.

```
$ mv foo2 bar/foo
```

Notice how you can use the mv tool to rename files, as well as move them. If you just wanted to move 'foo2' to the 'bar' directory without renaming it, you could do either of the following:

```
$ mv foo2 bar/
$ mv foo2 bar/foo2
```

As you are not renaming it, you don't need to append the filename to the directory, but you can do so, if you like to be sure what's going on. You need to be careful when using mv and cp, as they will overwrite any existing files you may have. This is not an issue with our practice files, but when you do it with real files, it could cause problems. You should back up before interacting with any files using the command line if you are worried about losing things. If you would like to be notified when your command calls for an overwrite, use the -i switch. For example:

```
$ cp -i foo bar/
```

If you did 'mv foo2 bar/foo' before, you will be prompted to make sure you want to overwrite the file. There are several other options that you might find useful which can be found in the man pages. One you may want is -v, which will show all files as they are copied or moved. You can use wildcards to copy or move multiple files.

```
$ mkdir new-directory/
$ mv bar/* new-directory/
```

This will move everything from bar/ to new-directory/. If you would rather move the full directory, and not just the files and directories within it, use the same command, but without the '*'.

Finally, let's remove all the example files using the rm command.

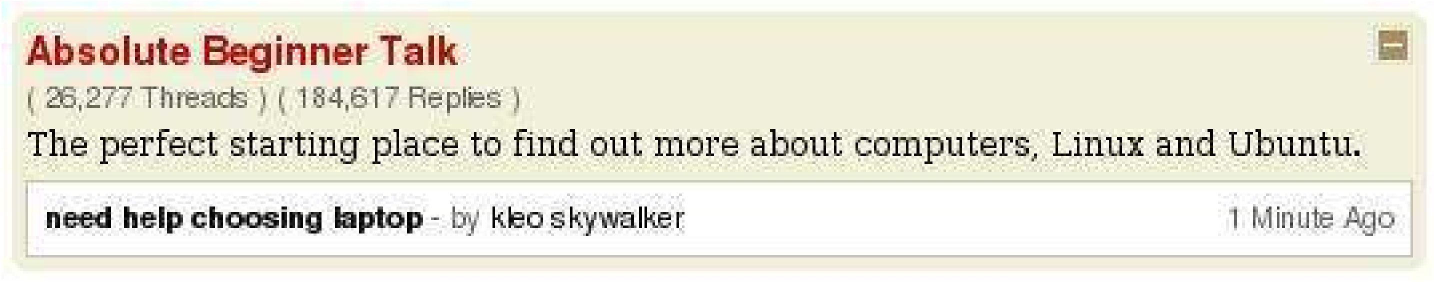
```
$ rm -rf bar/
$ rm -rf new-directory/
$ rm foo
```

Using rm with -rf causes rm to remove files and directories recursively, and to force removal without prompting. This is normally used for removing folders and all their contents. Be very careful that you enter the right file or directory when using rm; if you hit tab and don't look at what it has completed to, you could end up removing something you didn't want to! Make sure that if anyone is giving you support and tells you to use rm, you know what you're removing, especially if it has sudo in front of it. Don't forget to backup any important files, even if you think you aren't going near them!

Robert Clipsham is a self confessed geek, whose hobbies include: programming/scripting, chatting on IRC and not writing his articles on time.

Threads: 873,580, Posts: 5,633,487, Members: 649,029, Active Members: 63,991

the place for Ubuntu, Kubuntu and Xubuntu beginners...



Absolute Beginner Talk
(26,277 Threads) (184,617 Replies)
The perfect starting place to find out more about computers, Linux and Ubuntu.

need help choosing laptop - by kleo skywalker 1 Minute Ago

COMMAND AND CONQUER

Written by Robert Clipsham

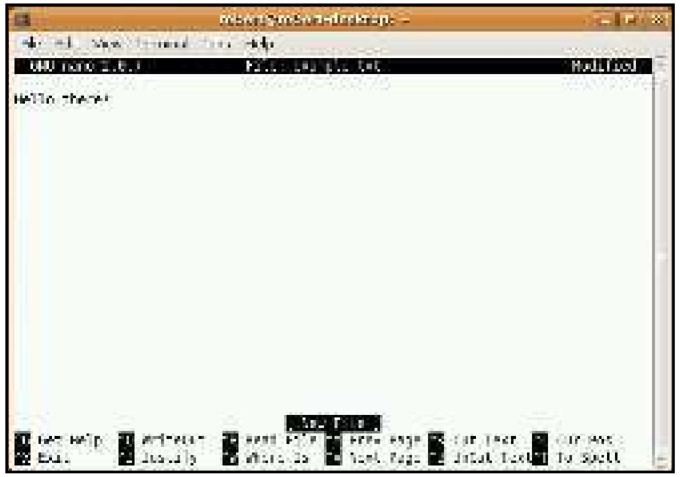
If you've been following the tutorials so far, you should now know the basics of staying safe, and managing files at a command line. This month we show you how to edit files using Nano and Vi/Vim.

To get started, I'll show you the easier of the two editors to use, Nano (right).

```
$ nano example.txt
```

At the top you will see three things. On the left 'GNU Nano' followed by a version number, in the center the name of the file, and on the right the status of the file. At the bottom, there are two rows of commands for the editor, then a status row just above them. To edit the file, just start typing like you would with any other text editor!

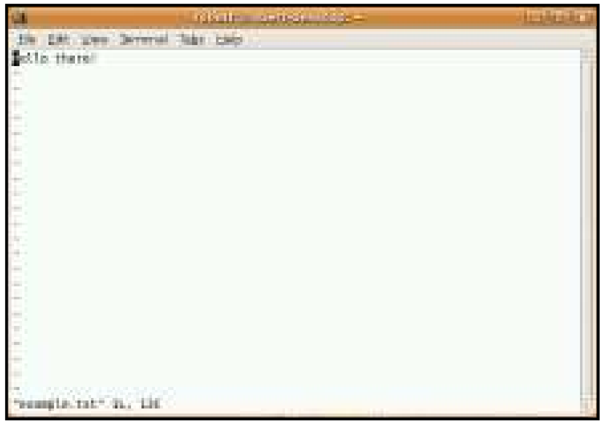
Other than editing the file, the common functions (and their key combinations) are listed at the bottom. For example, ^O (Ctrl+O)



will 'WriteOut' or save the file, and ^W (Ctrl+w) will allow you to search for a certain term in the file. Enter some text into the file, then save it. Nano is the easier of the two editors to use, but its functionality is much less than that of Vi or Vim.

By default, Ubuntu comes only with the basic Vim package, without all the bells and whistles. This is fine for this tutorial, but for advanced editing, you'll need the full package (we'll show you how to get this using a command line in the next tutorial).

Text editors do two main things: edit files and automate tasks. The tasks could be anything from find/replace to just saving a file. In graphical editors, the latter is done by using menus or clicking buttons. In Nano, you use certain key combinations to perform these tasks. Vim has two modes for this functionality: an insert mode and a command mode. Vim (below) starts in command mode by default, so you won't be able to edit the document.



```
$ vim example.txt
```

To switch to insert mode, press 'i' or the insert key. You will now be able to edit the file. Enter a few lines of extra text, then switch back to command mode by pressing the escape (esc) key.

Command	**Function**
:w	Save the document.
:q	Quit Vim.
:q!	Quit Vim without Saving
h,j,k,l	Move around document, you can also use the arrow keys
:e [filename]	Open the given file.
:help [command]	Open the main help page or get specific help
:set [option] [value]	Get a list of options that have been set, or set an option

There are a hundreds, if not thousands of commands in vim, providing a lot of functions. Some of the basic commands are shown in Fig.1 (left).

So, for example, if you wanted to save your changes, type ':w' (all of vim's command start with ':'). You can also mix commands, so for example ':wq' saves and quits vim.

These are just a few of the basic commands Vim has. There are also commands for copy/paste, find and replace, syntax highlighting, showing line numbers, and more advanced functions too. If you want more information, use :help or read the documentation at http://www.vim.org/docs.php.

Robert Clipsham is a self-confessed geek, whose hobbies include: programming/scripting, chatting on IRC and not writing his articles on time.

COMMAND AND CONQUER

Written by Robert Clipsham

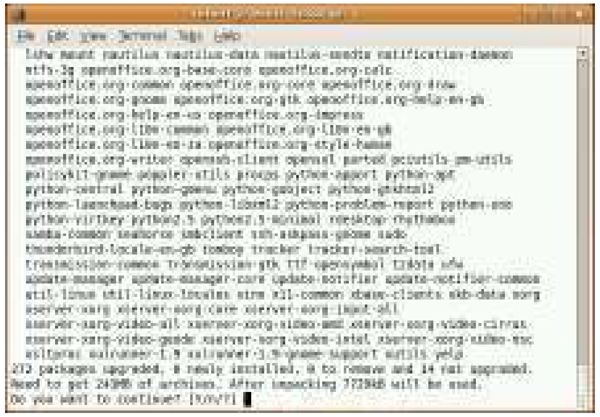
This month, we'll show you how to do package management using a command line. When it comes to package management using a command line, there are two main tools available, apt-get and aptitude. I won't go into a comparison of them here, as a quick Google Search will give more than enough information on it. I have chosen to use aptitude for this tutorial, but you can make your own decision based on the information you find.

First off, let's update our system. Open your favorite terminal emulator, then do the following:

```
$ sudo aptitude update &&
sudo aptitude safe-upgrade
```

This may look quite complicated at first, so let's break it down. First of all, you'll notice we're using sudo. If you've been following the tutorial from the start, you'll know that this runs the command as the 'root' user, giving it full control over your system. This is necessary, as otherwise you won't be able to install the packages. The next bit tells aptitude to 'update' its package database, so it has a list of the most recent packages. The '&&' is a command line operator, which tells the command line to run another command. The next command tells aptitude to run a 'safe-upgrade' of all the packages on the system. This is the command that actually updates the packages. There is also a 'full-upgrade' option available, which is less conservative than safe-upgrade, and could cause unwanted actions. See the man page for more information on it. You will be asked to enter your 'sudo' password, which will be the same as the password you use to log in (you must have a privileged account to do this; it won't work on restricted accounts).

You will see lots of text fly past, telling you where aptitude is getting its package lists from, and some status information on it. The next set of output will be from the second command. It will look something like:



```
  libw mount nautilus nautilus-data nautilus-sendto notification-daemon
  ntfs-3g openoffice.org-base-core openoffice.org-calc
  openoffice.org-common openoffice.org-core openoffice.org-draw
  openoffice.org-gnome openoffice.org-gtk openoffice.org-help-en-gb
  openoffice.org-help-en-us openoffice.org-impress
  openoffice.org-l10n-common openoffice.org-l10n-en-gb
  openoffice.org-l10n-en-za openoffice.org-style-human
  openoffice.org-writer openssh-client openssl parted pciutils pm-utils
  policykit-gnome poppler-utils procps python-apport python-gpf
  python-central python-gnome python-gobject python-gtkhtml2
  python-launchpad-bugs python-libxml2 python-problem-report python-uno
  python-virtkey python2.5 python2.5-minimal rdesktop rhythmbox
  samba-common seahorse smbclient ssh-askpass-gnome sudo
  thunderbird-locale-en-gb tomboy tracker tracker-search-tool
  transmission-common transmission-gtk ttf-opensymbol tzdata ufw
  update-manager update-manager-core update-notifier update-notifier-common
  util-linux util-linux-locales vim x11-common xbase-clients xkb-data xorg
  xserver-xorg xserver-xorg-core xserver-xorg-input-all
  xserver-xorg-video-all xserver-xorg-video-amd xserver-xorg-video-cirrus
  xserver-xorg-video-geode xserver-xorg-video-intel xserver-xorg-video-nsc
  xulrunner xulrunner-1.9 xulrunner-1.9-gnome-support wutils yelp
272 packages upgraded, 8 newly installed, 0 to remove and 14 not upgraded.
Need to get 243MB of archives. After unpacking 772MB will be used.
Do you want to continue? [Y/n/?]
```

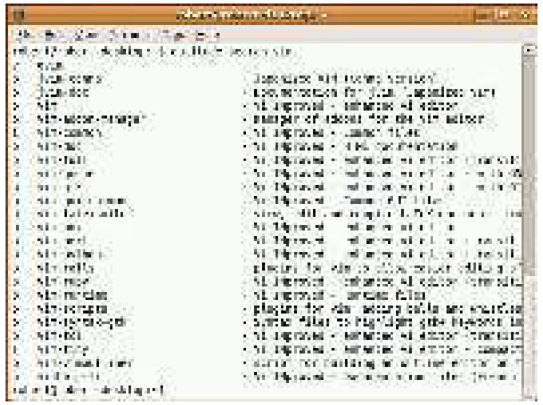
Read the information it gives, and if you're ok with it, type 'y' (without the '') and press enter. Otherwise, type 'n', and nothing will happen. As you can see from the screenshot, there are numerous updates I need to do; however, if you've been keeping your system up to date, then it may just take you straight back to a command line, as there are no updates.

The next thing to show you with aptitude is how to add and remove packages. However, this is of no use if you don't know what packages are available.

Let's start by searching for a package. In the last issue, I mentioned that Ubuntu didn't come with the full vim package, so let's find that.

```
$ aptitude search vim
```

Notice that sudo isn't needed to search for packages? You don't need it, as you aren't changing any system files. This should show you a list that looks something like this:



This is a list of packages matching the pattern you gave, along with their description and status. Here's a table (above right) showing the meaning of some of the different status-codes (you can also find these in the man pages).

Now to install vim:

Status	Description
p	No trace of the package exists on the system.
c	The package is deleted, but its configuration files are still on the system.
i	The package is installed
v	The package is virtual

```
$ sudo aptitude install vim
```

This acts just like safe-upgrade in the way it shows dependencies that it will install. If you are sure you want to install it, type y and press enter. Vim will now be installed on your system ready for use. To remove it at a later point, you can use:

```
$ sudo aptitude remove vim
```

And it will be removed. Simple!

Robert Clipsham is a self-confessed geek, whose hobbies include: programming/scripting, chatting on IRC and not writing his articles on time.

COMMAND AND CONQUER

Written by Robert Clipsham

If you've ever misplaced a file, this month's Command and Conquer is just what you need. There's a bounty of graphical search tools out there, but why use a GUI when you can use the command line!

The first command we'll look at is grep. Grep is used for searching for a specified string within a file. For example, if I had a plain text file called cookies.txt with a recipe for making cookies in it, and wanted to find how many eggs I needed, I could do:

```
$ grep eggs cookies.txt
```

This will give a list of all lines that contain the word 'eggs'. As a more realistic example, say I had a configuration file (I'll use apache as an example here, though it could be anything) and wanted to change the name of the error log. Just knowing that it is in the file is no use, I need to know what line number to look at so I can change it.

```
# grep errorlog /etc/apache2/apache2.conf
```

Notice that I'm root when running this command. You may be able to run this command as your normal user, depending on the permissions for your configuration file. Try running it as a normal user first! You will notice that this command gives no output. This is because grep is case sensitive.

```
# grep -n ErrorLog /etc/apache2/apache2.conf
```

Using -n will make grep give line numbers, so you can find what line the directive is on. Notice that I have changed the capitalization of ErrorLog in this example. You could also use the -i option to make grep ignore case. You can also search all the files in a directory using -r:

```
$ grep -ir eggs recipes/
```

would find all recipes containing eggs in my recipes directory. What if you want to search by file name? This is where find comes in.

```
$ find recipes/ -type f -name '*.jpg'
```

This will find all the files (-type f) with a name ending in .jpg. Notice how I have enclosed *.jpg in single quotes? This prevents your shell from expanding the *. Try the following, and notice the difference.

```
$ echo *
```

```
$ echo '*'
```

The first should give a list of all the files in your current directory, while the second should give *. While find on its own may not seem that useful, when coupled with xargs it becomes a very powerful tool.

```
$ find recipes -type f -name '*-cake.txt' | xargs -I % cp % old-recipes/
```

This command takes the output of “find recipes -type f -name '*-cake.txt'”, then pipes (|) it into xargs. Using -I % tells xargs to replace % with each line it receives.

```
$ find recipes -type f -name '*-cake.txt'
```

```
recipes/chocolate-cake.txt

recipes/cheese-cake.txt

recipes/fairy-cake.txt
```

If that (above) is the output, then these are the commands xargs will run:

```
$ cp recipes/chocolate-cake.txt old-recipes/
```

```
$ cp recipes/cheese-cake.txt old-recipes/
```

```
$ cp recipes/fairy-cake.txt old-recipes/
```

If you have a lot of files that match a certain pattern, this can be a very useful method to automate tasks such as a selective backup. If the output of find should be appended to the end, then you can remove -I % and it will be appended automatically to the end of the command. The final command that we'll cover in this issue is locate. Locate might not be installed, so you may need to install it before it works. Locate is a very fast way of finding files with a certain file name. If you have apache installed, try the following:

```
$ locate apache
```

A huge list of files will fly past, so it might be useful to combine its output with grep to find exactly what you're looking for.

```
$ locate apache | grep etc
```

This will narrow down the results to show only those containing 'etc'.

Robert Clipsham is a self-confessed geek, whose hobbies include: programming/scripting, chatting on IRC and not writing his articles on time.

Full Circle Magazine
The Independent Magazine for the Ubuntu Community

Ubuntu 8.10 is out! You can find information on the new release by clicking here.

New Thread

Page 1 of 5 **1** 2 3 > Last »

Threads in Forum : Full Circle Magazine

Forum Tools Search this Forum

The official Full Circle forum, hosted at Ubuntu Forums

http://url.fullcirclemagazine.org/c7bd6f

COMMAND AND CONQUER

Written by Philip Royer

You've been sitting in a doctor's office for nearly an hour. Each second seems like an eternity. Worry is starting to grip your gut. The test results are being discussed by two nurses in the hallway, and the grim looks on their faces don't look promising. You wonder what they've found and hope it isn't life threatening. You have too much to lose: your family, your job, your nice car. You jump as a doctor steps into the room holding a clipboard and looking intently at the scattered data. He looks up and into your eyes. "I'm sorry sir, but it looks like this disease is terminal." Your heart sinks, knowing that this sickness only ends in death. Everything you've worked for... gone.

I remember my first experience with the computer terminal being very similar. I was VERY timid about entering codes manually into the computer. With all the code readout scrolling by, I was worried that something might go wrong. What if I mis-spelled something and it destroys my computer? Over time, though, that worry was replaced by a confidence I could have had from the start.

This is what I think every time I hear the word "terminal." It's never a happy word. So, it's no wonder that when I looked at some directions for installing something in Linux, I cringed in fear when they said to open the Terminal. But is the fear of entering a string of letters and technically powerful words a reason to be afraid? Is this a rational fear? Why is it that so many are in fear of switching to Linux because of "having to use the command line?" Let me try to smash some of these fears and correct any misinformed thinking by helping you, the user, to understand the terminal a little better.

What is the Terminal?

The terminal is an application that enables you to talk to the computer using text-based commands. This means that you're eliminating the need to use a graphical interface, or a bunch of friendly buttons, to initiate commands. It's called command line, meaning, instead of clicking buttons and icons, you enter commands with text. For example, to update your system, you would enter:

```
sudo apt-get update
```

There are many other such commands that you can use to launch applications. Back in the earlier days of Linux, most things were done using the command line. This is one reason many non-geeks didn't, or still don't, want to move to Linux.

The truth is, Linux is so far past that primitive stage of being all command line that it has reached the point of being a usable desktop by anyone, geek or non-geek. So if Linux, or more specifically, Ubuntu, has reached a point where the desktop is no longer command-line driven, why use the terminal at all?

Why Use the Terminal?

As a more in-depth Linux operator, the terminal can be your greatest friend. It will talk to you when you have a problem. It will tell you what is wrong. Maybe not in the same way humans interact, but in a very similar way.

Let me elaborate a little. When you click on an icon on the desktop, or click a button, it sends various commands to the computer. All these commands you don't see, because they are going on in the background. If, for example, I was to click the update button on my menu, to update the system, the only thing I would see would be a status bar indicating how much time before my system was updated. Now if I were to type "sudo apt-get update" (the command line equivalent) I would get a very long and exhaustive list of Web address that my computer was searching for updates.

You're probably thinking, "Why would I want to see all that?" The answer is simple: it tells me what's going on behind the scenes. Well, why would I need to know that? Because if there was a problem with an installation, and a problem was preventing it from completing, then there will be error messages in the terminal readout that I would not see in the desktop scenario. If I was just to have the installation say "I'm sorry, your installation of Gobbledygook Plus couldn't be completed", then I wouldn't know what went wrong. But, by installing it using the terminal, error messages would appear, informing me of the problem, and enabling me to fix it, or get help. But, the use of the terminal is not for everyone.

Should I use the Terminal?

Though the Terminal is very useful in many computing situations, I do not recommend it for everyone. For basic users, it might be too difficult to grasp, and, depending on the commands involved, may accidentally mess up your machine. But, if you are having a problem with your computer, such as a program isn't running, or an installation crashes, you can post the output of the terminal on the Web to get help from others.

Must I use the terminal?

Use of the terminal is not a requirement in Ubuntu, it's an aid to help you when you have computer problems. Even as an advanced user, I rarely use the terminal. Just because you have a spare tire under your car, doesn't mean you have to use it all the time. You only use it when you have a flat tire. It's that simple.

Hopefully, you will no longer fear the terminal. Instead you will see it as a tool that you can get by quite easily without using. The terminal is a useful tool, but it doesn't have to dominate your computer usage. One should never fear things that are unknown, because if we did, we would never learn anything new.

Unfortunately, Robert is being intimidated by real-life lately, and is unable to continue writing Command & Conquer. So, we're looking for a stand-in for a few months. If you would like to take his place for a few issues, please contact Robert at: mrmonday@fullcirclemagazine.org

COMMAND AND CONQUER

Written by Lucas Westermann

In this instalment of Command & Conquer, we will cover the basic uses of *grep, sed, awk, cat*, and *cut* for formatting output. This can be useful when putting together things such as Conky, or scripts that display theme information in the terminal.

The first command we should look at is cut. If, for example, we wanted to display the distribution name in a theme script, we would find it in /etc/issue. If, however, we run cat /etc/issue we see that there is one line too many, and there are escape characters included in the line. So if we run /etc/issue|head -n 1, we remove the extra line by piping the output of cat through head, which then displays only the first line of the output. So far so good, but what about the escape characters? This is where cut comes in handy. To use cut, we must supply a delimiter, and then tell it what to do with this. The command we would use is:

```
cat /etc/issue|head -n 1|cut --
delimiter=' ' -f 1,2
```

This command then tells cut that the delimiter to use is a blank space, and to display the first two fields (basically, cut slices output up into segments according to the delimiter, so fields 1 and 2 are the first fields before and after the first delimiter in the output, in our case, Ubuntu 8.10). Cut can also be used to display only a certain number of characters when using the -c flag.

With sed the same could be done with:

```
cat /etc/issue|sed '{s/\\n// ;
s/\\l// ; /^$/d}'
```

This may look like gibberish, but the first two expressions (each expression is separated by a semi-colon) tell sed to substitute "\n" with "" (nothing), and the same for "\l", removing those characters from the output. "/^$/d" is a command that tells sed to delete any blank lines ("^$" is the regular expression for a line that begins with a blank and ends with a blank and has nothing between those -- a blank line). So 's/\\n\b//' is merely telling sed to substitute ("s/") "\n" ("\\n") with "" ("//"). The reason the command is in braces is because we are actually applying three expressions on the output, and want it returned only once, so we put the expressions in braces ("{}"), and separated by semi-colons.

Lastly, the same output can be achieved using awk:

```
cat /etc/issue|awk '/\\n/
{print $1,$2}'
```

This command again uses regular expressions, but is slightly easier to understand than sed. Basically, awk '/\n/ {print $1, $2}' finds any line that has "\n" in it, then prints the first two fields (the default separator is a space, but you can set your own using the -F flag). This saves us having to format out the extra line and the \l of the output. You could also forgo piping the output of cat /etc/issue into the command (or either of the others), as they can all be applied to a file specified at the end of the command. I used cat in order to leave the commands less jumbled.

This is intended only as an introductory look at the abilities of awk, sed, and cut. Their flexible implementations make it hard to write a brief in-depth tutorial for the three of them. The above explanations are intended to illustrate how the commands work, and not fully explore their potential uses. A real-world implementation of these commands would be in the first half of a custom theme script (the example below also displays theme info, but that part isn't necessarily pertinent to this article; it was left there to keep the script complete). The example also contains a challenge for anyone who wishes to attempt it: Figure out how to use one of the three commands to remove the indentation in the memory part of the script, and, if you want more practice, try replacing every occurrence of cut, sed, or awk with a different command that does the same (i.e. replace a cut command with awk). There is no prize, but it is good practice to figure out the inner workings of the commands.

http://fullcirclemagazine.org/issue-21-shell-script/

Further Reading:

Sed - http://www.grymoire.com/Unix/Sed.html

awk - http://www.linuxjournal.com/article/8913 or http://www.linuxfocus.org/English/September1999/article103.html

cut - http://learnlinux.tsf.org.za/courses/build/shell-scripting/ch03s04.html

The man (manual) page of each command, can be accessed with:

```
man [command]
```

this is useful if you're unsure of how to invoke a command.

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. When he finds time, he also publishes a blog at http://lswest-ubuntu.blogspot.com.

COMMAND AND CONQUER

Written by Lucas Westermann

Ever had a video clip you wanted to reduce in size and add to a slideshow for a presentation? Or, maybe convert into another format so that you could play it? Today, I will show you how to do both of those things using the command-line tool 'ffmpeg'. I will also introduce you to the 'mogrify' command, which is an image-editing tool contained in the imagemagick package. It allows you to do many things, but I will cover the basics – mostly the resizing of images (e.g. for thumbnails or other small images).

To use these tools, you will need to install ffmpeg and imagemagick through either Synaptic Package Manager or Add/Remove Applications, or, in the spirit of this article, through the command-line with:

```
sudo apt-get install ffmpeg
imagemagick
```

It's safe to run the install command – if you're unsure whether or not you've already installed it – since apt-get will not overwrite the existing program, but merely inform you that it's already installed. Also, it will ask you for your password (since you're using sudo), and, if you're new to this, you may be surprised that it does not show anything when you enter your password. This is normal, just type your password and hit the enter key.

For this article I will be converting a short clip from "Freedom Downtime" which I used in a presentation. ffmpeg offers a lot of options (which you can read about in detail in the roughly 13-page-long manpage – by using the command "man ffmpeg"), but the option I use most frequently is the option to convert files. The format for such a command is:

```
ffmpeg -i inputfile.filetype
outputfile.filetype
```

That command would just convert the input file to whatever output file you specified in 'output.filetype' – without changing the size (since, if the size is not specified, ffmpeg defaults to the size of the source). However, to convert "Freedom Downtime" (starting size of 640x480) to a smaller clip (say 320x240), the command would look like this:

```
ffmpeg -i freedom\
downtime.mpg -s 320x240
freedom\ downtime\ resized.mpg
```

Of course, you could also change the format of the file at the same time by changing the file extension. This shouldn't take too long (obviously depending on the size of the clip). My clip was a few minutes long, and took maybe 30 seconds to convert on my laptop.

Mogrify is a very handy tool, especially if you find yourself posting a lot of pictures on web forums that don't allow uploads of images exceeding a certain size, or do not allow linking to large images for previews in posts. I use mogrify mainly for making thumbnails of images, but it can do many more things, such as adding text, adding effects (charcoal, colorized, etc.) and much more (it's all outlined, once again, in the manpage, reached with "man mogrify"). Mogrify supports resize arguments in either percentages or pixel values. So, for example, if you have a 1280x800 pixel image which you want to resize to 640x400, you can do so with either:

```
mogrify -resize 50 in.jpg
out.jpg
```

or:

```
mogrify -resize 640x400 in.jpg
out.jpg
```

or even just:

```
mogrify -resize 50% in.jpg
out.jpg
```

However, if the pixel measurements and the aspect-ratio you supply are different, it could result in a smaller image than expected, as it will scale to the nearest values that are still in proportion. Mogrify also has a thumbnail argument, which does roughly the same as the resize option, yet it removes any unnecessary comments, etc. from the file header to reduce image size. So using that tool the command would be:

```
mogrify -thumbnail 50 in.jpg
out.jpg
```

You can also use it to convert images using the “-format” argument. So,

```
mogrify -format jpg *.png
```

will convert all the .png files in the current directory into .jpg format (names will be the same).

As you can see, contrary to common belief, command-line tools can help with graphical projects, and often do it more quickly or more effectively than a GUI program with confusing menus and/or different layouts in newer versions. The command will (usually) stay the same, and arguments are very rarely changed. And so, command-line tools are much more universal – which is why users on ubuntuforums.org usually supply commands instead of GUI methods for solutions, since the commands apply to Kubuntu, Xubuntu and Ubuntu, as well as other systems too. Hopefully, you've found this useful, and next time you need to convert a video or an image, you'll remember mogrify and ffmpeg. After all, practice makes perfect.

Further Reading:

http://www.imagemagick.org/www/mogrify.html – Very useful guide on imagemagick in general, from the imagemagick site

http://www.ffmpeg.org/documentation.html - The official ffmpeg documentation with useful information, FAQs, etc.

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. When he finds time, he also publishes a blog at http://lswest-ubuntu.blogspot.com.

COMMAND AND CONQUER

Written by Lucas Westermann

I've noticed that there were quite a few posts on Ubuntu Forums in the last couple of weeks about how to troubleshoot errors that occur. Therefore, I decided I'd cover some basic things I do when trying to find where a problem is occurring, so that I can google for a solution (unless I can fix it without that). For general knowledge, logs are stored under /var/log/, and there are system logs (for everything), and then a collection of logs for applications or processes.

The first thing that should be done - if an application crashes on startup (e.g. Firefox freezes and crashes after you launch it) - is to launch the application from the terminal - then any errors that arise are displayed in the terminal. If that gives you an error message, the best solution would be to either copy and paste the gist of the error into google for a search, or else, if you understand the error, to use that information to find out what you need to do to resolve the issue.

A more difficult problem to troubleshoot is if, for example, you insert a USB stick and it isn't recognized by Nautilus. The first command that should be run is:

```
dmesg|tail
```

See if the output of that refers to the insertion of a USB stick, or anything that relates to your specific problem. If it doesn't appear in the output you can try either extending the amount of output you see by adding the -n argument to tail, and the number of lines you want displayed. So seeing 14 lines of output would be:

```
dmesg|tail -n 14
```

Otherwise, you can remove and re-insert the stick in a new USB slot, or check the outputs of

```
sudo fdisk -l
```

```
lsusb
```

to see if they show any indication of having recognized your USB stick. If the drive is recognized by the system you can, for a more specific error message, try mounting the USB stick manually, and see why it is failing.

The above commands and ideas can apply to almost any issue that you might encounter, as long as you have a basic understanding of where to look. The next suggestion, however, is for slow boot times and to see what exactly is going on when you start the computer - in case something is hanging and causing a large delay.

This is done by a program called boot chart which is in the Ubuntu repositories. You can just install it with:

```
sudo apt-get install bootchart
```

Once it's installed, you merely have to restart your computer and then you can view the resulting chart in eye of gnome (default image viewer) - by navigating to the

/var/log/bootchart/ folder and opening the correct image (they are named by date).

Also, hardware issues can be checked by the program lshw, which lists hardware information. The most useful way to do it is to run it using the -C flag, and then the section (display, network, etc.). So, for example, wireless issues would be checked with:

```
sudo lshw -C Network
```

This command displays details on your network devices (ethernet and wireless), and lists as much information as possible, from capabilities to drivers, and so forth. The most important is probably to check that it isn't disabled, and that the driver is listed (it will be in the last line of the device section and denoted with “driver=[drivername]”).

On a last note, if you run into any errors or problems you are unable to troubleshoot or fix, attach as much information as possible that could be relevant to any request you make. Too much information is better than too little. For example, if you're working on a wireless connectivity issue, or a wireless device not being recognized, post the output of commands such as ifconfig, iwconfig, lshw -C Network; if it's a USB wireless dongle, then also the output of lsusb, if it's PCI then lspci, etc. This helps because whoever decides to help you won't have to ask for any more information if you supply enough in the first post, and any subsequent replies may be able to answer your question without a long back and forth, which usually can last for a day or two (depending on timezones and time of day the post was created). Just keep in mind that the more information is supplied, the more information someone has to work with to solve a problem that they can neither see nor physically troubleshoot.

This article was created to be a useful guide to give people somewhere to start when trying to solve problems on their own, or to improve their chances of getting support when they need to ask for it. It is by no means exhaustive, nor are the commands listed explained thoroughly. Any commands can be investigated in the man pages (using the “man” command covered in an earlier article of C&C), and using the commands will also greatly help your understanding of them.

Further reading:

http://www.troubleshooters.com/tpromag/200007/200007.htm

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. When he finds time, he also publishes a blog at http://lswest-ubuntu.blogspot.com.

COMMAND & CONQUER

Written by Lucas Westermann

This article will be dedicated to automating things you do every day, and to helping you create a backup solution that works for you. To begin, a "bash script" is a script that contains everyday bash commands, and some functionality that isn't used often from the command-line (such as if-then-else loops, while loops, etc.). Cron is a daemon that runs all commands listed in a user's crontab file (so if you want to run something that requires root powers, you need to edit the crontab for the root user). For the sake of this article, I will use a simple one-line bash script that condenses all the .java files in a folder into one text file. However, you could change it to condense log files. I will explain how the bash script works, and I will then cover how to set up cron to run the script every 6 hours.

To create a bash script, the method I prefer is to write it using nano, so that's what I'll use in the examples. If you prefer a GUI editor, replace "nano" with "gedit" (minus the quotes). To start, do the following:

```
nano FileCondenser
```

This will start a new interface in the terminal with a completely empty file. Then enter the following lines (I will explain them in a moment):

```
#!/bin/bash find
$HOME/workspace/Year\ 11 -
name '*.java'  | while read
line; do cat "${line}"; done
```

Save the file with Ctrl+X, and hit Enter. To make the script executable, run the following command:

```
chmod +x FileCondenser
```

Now you can execute the script by running, in the terminal:

```
./FileCondenser
```

or by creating a symbolic link to /usr/bin (and executing it like any other program you use in the terminal) by issuing:

```
sudo ln -s /<path to
script>/FileCondenser
/usr/bin/FileCondenser
```

where you replace "<path to script>" with the actual path (the absolute path is required for symbolic links).

The script essentially uses find to locate all the files that end with .java in the eclipse folder, then pipes it (redirects the output) to the while loop, which states that while there is a line within one of those files to read, output the line, and once there are none, stop.

As I mentioned above, you can change the path to /var/logs/, and have it read:

```
find /var/logs/ -name
"*.log"|while read line; do
cat "${line}"; done
```

which will condense all the

> **The script essentially uses find to locate all files that end with .java...**

files that end with .log from the /var/logs folder (and its subdirectories) and outputs it to the terminal. Or you can redirect the output to a file.

To add the script to your crontab file, I'd suggest creating the symbolic link so the command is shorter, but that's not necessary. To edit the crontab file run the following command:

```
crontab -e
```

It will then open the user's crontab file. If you want to add it to the root user's crontab (for backup scripts and the like) run the following:

```
sudo crontab -e root
```

Once the crontab is open, it

is important to know the format of the entry. It goes <minute> <hour> <day> <month> <day of week> <command>. All positions must be filled by either wildcards or values. A couple of examples:

Every 5 minutes:

```
*/5 * * * * FileCondenser >
$HOME/condenseFile.txt
```

Every Sunday at 6:

```
00 18 * * sun FileCondenser
> $HOME/condenseFile.txt
```

The entry I'd suggest using for a periodic backup of logs would be:

```
* */2 * * * FileCondenser >
$HOME/condenseFile.txt
```

which will run the script every two hours, and store the output in a file in your Home directory called condenseFile.txt. On a final closing note, a simple backup script that I can think of would be one that archives a certain folder, or collection of folders (like your home folder or your root directory), and stores it on a backup partition, or on an external hard drive, etc. An example for this would be:

```
tar cvvzf
/media/Backup/Music\
Backup/backup.tar.gz
$HOME/Music
```

This command assumes your backup hard drive/partition is mounted at /media/Backup and that it contains a folder called “Music Backup”. The tar command creates a gzipped tar archive at /media/Backups/Music Backup/ called backup.tar.gz, which contains the contents of your Music folder. You could run this command from a crontab without a script, but I would assume if you want to make system-wide backups (of your root partition, for example) you'd want to include a long list of folders to ignore - possibly reading the ignore list from another file; something like that, and so a script would just keep it clean. Not only that, but you could easily run it manually in the terminal as well. I recommend testing any commands or scripts on a small test folder before committing them, otherwise you may run into problems.

Further Reading:

Python version of the script for those interested:
http://lswest.pastebin.com/m5b536464

A bash script tutorial:
http://www.linux.org/docs/ldp/howto/Bash-Prog-Intro-HOWTO.html

Linux Shell Scripting with Bash by Ken O. Burtch (ISBN: 978-0-672-32642-4)

Cron tutorial:
http://www.clickmojo.com/code/cron-tutorial.html

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. When he finds time, he publishes a blog at: http://lswest-ubuntu.blogspot.com.

contents ^

COMMAND & CONQUER

Written by Lucas Westermann

FCM#24 correction: /var/logs/ mentioned last month should actually be: /var/log/ – apologies for any problems.

Have you ever thought to yourself “There has got to be an easier way to do this,” as you're sifting through tons of old commands in the history – looking for a specific one? There is an easier way. Not only that, but there are a few useful things you can do with your terminal and shell to make life easier. Now, to start with, you'll notice I used the terms “terminal” and “shell,” and I have known some people to think they are one and the same, so I will take a moment to explain.

A terminal is the actual program that displays the shell prompt, but you can easily change which shell it displays. Also, any configuration of the prompt has to be done on the shell, not the terminal. Two commonly used shells are Bash (Bourne Again Shell), which is the default one in most cases, and Zsh (creatively named “Z Shell”). If you want to try out a new shell, all you have to do is install it and then run, for example:

```
zsh
```

from a running bash shell. If you decide you like it so much that you want to change (like I did), you can change by running:

```
sudo chsh <username>
```

where of course “<username>” is your actual username. It then asks you something similar to this:

```
Changing the login shell for
<user>
Enter the new value, or
press ENTER for the default
Login Shell [/bin/zsh]:
```

where you just need to enter the path to the shell file (usually in /bin/). The entry in the square brackets is your currently chosen shell (as you can see, I use the Z shell).

Back on topic: The history of your shell can sometimes be extremely long and confusing (I think mine is at 1000-or-so commands since I last cleared it, and that was not too long ago either). I very often find myself running the same commands over again, sometimes with lots of arguments, sometimes with very few. For the long ones, I've gotten in the habit of pulling up the list of history and searching it for the command I want. For example:

```
history|grep cd
```

returns:

```
996 cd Dropbox/Scripts/C
```

which displays for me the last cd command I ran, or all of the cd commands I've run if there were more than one. You'll

There has got to be an easier way to do this...

notice that it also displays a number on the left (in my case, it's 996). So if you want to re-run the command, you have a few options. You can copy and paste the line (which is, honestly, not very efficient since you need to take your hands off the keyboard), or you can run:

```
!996
```

which will automatically run the command in the history list with the identifier 996. Woah, I'm back in my C scripts folder, talk about easy! Only four characters for an entire command. This most definitely appeals to my (extremely) lazy side. It's also much faster than using the up or down keys to go through the history line by line to find the right command.

Hopefully, this has also appealed to you. One last note, you can also just run:

```
!cd
```

to run the last cd command again.

Another extremely useful thing I've learned is that you can add custom functions to your Bash shell (this works on Z shell as well). To do this you can open your .bashrc (in gedit, vim, nano, or anything that tickles your fancy). I'll be using vim.

```
vim .bashrc
```

You'll then have a very long (and most likely confusing) file open. I generally add custom entries toward the end of the file, just to keep it separate from what should be there, and denote it with a comment, but you can choose whatever works best for you. If you want to merely make running one command easier, something like using the command:

```
update
```

to run the following:

```
sudo apt-get upgrade
```

then I would recommend using aliases, since it's just a tiny bit less typing. (For me, the number of keystrokes is extremely important when making shortcuts). So, if you want to alias that, the following line should be added to your .bashrc . (I added a comment to just make it clearer – you can leave it out if you wish).

```
#Alias for updating the
system
alias update = “sudo apt-get
upgrade”
```

Another alias I use very often is this one:

```
#An alias to make the ls
command more detailed
alias ls = “ls -la --
color=always --classify”
```

As you probably noticed, I have effectively replaced the ls command with a much more detailed output. Now you may be asking yourself “but what if he wants to use the actual ls program without arguments?”. The answer is this:

```
\ls
```

The backslash overrides any aliases bound to that name, and runs the command as-is.

Now, focusing on functions again. These are basically scripts added directly to your shell configuration file, which can be quite useful. The example I'm going to be using is on page eight. Don't worry, I'll explain it.

I've used this script occasionally to convert the audio from .m4a files to .mp3, since I can't see myself typing all those commands by hand (no matter how occasional the use). The function is defined in the first line, and after the first brace (curly bracket), it's then the actual script. It checks to see if the arguments are empty, and if so, it prints the error message at the very end of the script (second to last line). If it does have arguments, check to make sure the first file exists, and then create the output file (the .mp3 in this case). If it doesn't exist, print “file <filename> doesn't exist!”. Once that's done, it checks to see if the output file exists (the .mp3), to ensure that the first loop completed successfully. If not, don't move the file. It then moves the .m4a file to your Music folder, while taking the output name (so that you can tell which m4a goes with which mp3), and exchanging the mp3 with m4a, so that it still plays. It then also says that it was moved, and moves the mp3 to the Music folder. It also checks to see if the Music/m4a directory exists (the if statement before the m4a comment). If it doesn't, it is created before proceeding. Hopefully, people will find this useful, since I have a few leftover m4a files from my iTunes collection, and I convert them as I see them. The only thing I have to note is that the function seems to require the filenames to be in quotes (backslashes and spaces and such don't work). So the function would be run like this:

```
m4a “2-10 You're the
Inspiration.m4a” “You're the
Inspiration.mp3”
```

The resulting files will be in

contents ^

the ~/Music folder, and the ~/Music/m4a folders. However, the mp3 will be missing the id3tags.

I'll leave that as a challenge for any reader who wishes to undertake it. There are command-line tools that let you access tag information, and as a slight hint I will tell you that the archlinux package perl-mp4-info (in the Arch User Repository on the website) is a tool that will read the tags from the m4a file (or at least, it should), and it should be available for Ubuntu. If you can extend the script to automatically copy the tags from the m4a to the mp3 and feel like sharing your solution with me, please email the code to me, and I will add it into the next article (giving due credit, of course). I haven't actually written the code, but I have a fair idea of how to do it. (I haven't felt the need to test my algorithms since I think I have converted all my m4a files now!) I will compare any code I receive to my algorithm, and will note in the next article whether or not it was how I had thought it could be done, or if it's something I never thought of. As far as I know, ffmpeg won't carry over the information.

```
m4a(){
if [[ "$1" != "" && "$#" == 2 ]]; then
    #check if files exist
    if [ -e "$1" ]; then
        #convert audio
        ffmpeg -i "$1" "$2"
    else
        echo "File "$1" doesn't exist!"
    fi
    if [ ! -d "$HOME/Music/m4a" ]; then
            mkdir $HOME/Music/m4a
    fi
    #move the .m4a file to the m4a folder if the files exist in
the current directory
    if [ -e "$2" ]; then
        mv "$1" $HOME/Music/m4a/"`echo "$2"|sed 's/mp3/m4a/'`"
        echo "m4a file was moved to the ~/Music folder"
        mv "$2" $HOME/Music/
        echo "Moved the mp3 to the ~/Music folder"
    else
        echo "File "$2" doesn't exist!"
    fi
else
    echo "Invalid arguments (or too few/many), please run this
script with \"m4a <input> <output>\""
fi
}
```

I hope that these tips have helped you out, and that you've found it at least partially useful. I know they've saved me loads of typing and made working in the command line so much more efficient (I even use Awesome [Ed: a dynamic and tiling window manager: it's in the repo's] on my main system, so that I really hardly touch my mouse these days). I look forward to seeing if anyone feels like taking up the challenge of completing the function above.

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. When he finds time, he publishes a blog at: http://lswest-ubuntu.blogspot.com.

HOW TO CONTRIBUTE

FULL CIRCLE NEEDS YOU!

A magazine isn't a magazine without articles and Full Circle is no exception. We need your opinions, desktops, stories, how-to's, reviews, and anything else you want to tell your fellow *buntu users. Send your articles to: articles@fullcirclemagazine.org

We are always looking for new articles to include in Full Circle. For help and advice please see the Official Full Circle Style Guide: http://url.fullcirclemagazine.org/75d471

Send your comments or Linux experiences to: letters@fullcirclemagazine.org
Hardware/software reviews should be sent to: reviews@fullcirclemagazine.org
Questions for Q&A should go to: questions@fullcirclemagazine.org
Desktop screens should be emailed to: misc@fullcirclemagazine.org
... or you can visit our site via: fullcirclemagazine.org

Please note:

Special editions are compiled from originals and may not work with current versions.

Full Circle Team

Editor - Ronnie Tucker
ronnie@fullcirclemagazine.org

Webmaster - Lucas Westermann
admin@fullcirclemagazine.org

Special Editions - Jonathan Hoskin

Editing & Proofreading
Mike Kennedy, Gord Campbell, Robert Orsino, Josh Hertel, Bert Jerred, Jim Dyer and Emily Gonyer

Our thanks go to Canonical, the many translation teams around the world and Thorsten Wilms for the FCM logo.

For the Full Circle Weekly News:

You can keep up to date with the Weekly News using the RSS feed: http://fullcirclemagazine.org/feed/podcast

Or, if your out and about, you can get the Weekly News via Stitcher Radio (Android/iOS/web): http://www.stitcher.com/s?fid=85347&refid=stpr

and via TuneIn at: http://tunein.com/radio/Full-Circle-Weekly-News-p855064/

Getting Full Circle Magazine:

EPUB Format - Most editions have a link to the epub file on that issues download page. If you have any problems with the epub file, email: mobile@fullcirclemagazine.org

Issuu - You can read Full Circle online via Issuu: http://issuu.com/fullcirclemagazine. Please share and rate FCM as it helps to spread the word about FCM and Ubuntu.

Magzster - You can also read Full Circle online via Magzster: http://www.magzter.com/publishers/Full-Circle. Please share and rate FCM as it helps to spread the word about FCM and Ubuntu Linux.