Full Circle

THE INDEPENDENT MAGAZINE FOR THE UBUNTU LINUX COMMUNITY

SPECIAL EDITION

Command & Conquer

Volume Two

Issue 26 - 50

COMMAND & CONQUER

Written by Lucas Westermann

Before I get into any new topics, I'd like to take a moment to thank reader Harold De Bruijn for pointing out the existence of *pacpl* (http://pacpl.sourceforge.net/), and the fact that it supports the converting of tags. It's also in the Ubuntu repositories. This can be used instead of ffmpeg in the m4a function I explained in the past issue, since it saves the tags in the newly converted format.

Now, back to this month's article. When you mention the command line, most people think of complicated commands, or black terminals with white (or often green) text scrolling huge amounts of text. Or they think of apt-get, aptitude, elinks, etc, since they are widespread applications and very commonly used. However, how many people think of MOC (Music on Console), or irssi (IRC client)? Both programs are CLI-based, and are very useful (and lightweight, if that's what you're going for). Tired of having your music stop playing when your Xserver dies (or you turn it off)? MOC will continue playing music since it runs as a CLI server, or you can run it from a tty screen without any kind of X. Or has your Xserver died and you need help to fix it, but don't have any way to ask anyone? Irssi will let you go on IRC (so you can visit #ubuntu on freenode, for example). Not only are these applications useful for when you are without a GUI, but they also are extremely customizable (colors, extra functions, etc). I, for example, use a script to pull the information from MOC into conky, but you can also write a script that is executed at the end of a song to display the new song title and place it in a text file (for more real-time updates).

First, let's start with MOC. To install moc run the following command:

```
sudo apt-get install moc
```

Once the program is installed you can open it by running:

```
mocp
```

The default view will be, on the left side, a file browser, and on the right side will be the playlist (empty for now). Use the Tab key to switch between the file browser and the playlist. From the file browser menu you can browse to your music folder. If you want to add all your music at once, just hit Shift+a, so that it adds the directory to the playlist, hitting just 'a' will add the currently selected file. Once it loads all the music and tag information, you should have a nice list of your music. But wait, what if I want to play one specific song, do I have to scroll through this all? Simple answer? No. If you hit 'g' and then type the name of an artist, song, etc, it will filter the results in the playlist, and you can highlight the correct result and hit Enter to play it. You can use the file ~/.moc/config to specify a beginning layout, and if you want shuffle on or off, repeat, etc. I won't go through the almost endless list of options (because, well, I have no idea where to start, there's just so much), my ~/.moc/config file can be seen at: http://fullcirclemagazine.org/moc-config/

> **most people think of complicated commands, or black terminals...**

Basic Keyboard Controls:

g – search
space bar – pause/play
enter – choose/start playing selected file
tab – switch between file browser and playlist
n – next song
b – previous song
C – clear playlist
A – add folder recursively to

playlist
a – add file to playlist
s – shuffle toggle
h – help menu

IRSSI

Now, on to irssi. To install this program run the command:

```
sudo apt-get install irssi
```

(starting to see a pattern here?). Once it's installed you can run it with:

```
irssi
```

Once the program opens you're greeted with a pretty blank screen. To start, you will want to type:

```
/connect irc.freenode.net
```

(or substitute the server with any other one you'd like to connect to). Once you're there, if you know where you want to go, type:

```
/join #ubuntu
```

(Or, again, any other channel). If you join multiple channels, you'll see a list just above the input field. If you want to go from window-2 to window-1, for example, hit meta (usually the windows key, but ESC works as well), and 1 to get to the first screen. Or, if you want to see both at once, you can enter

```
/window show 1
```

which will display the first screen with whatever screen you had running before. There are so many controls and commands and options that I can't cover them all here, but they are explained fairly well on the actual website (see the Further Reading section for a link). One last thing I will cover are themes, since they are usually pretty interesting. First, find (or write) a theme to your liking, and then copy (or create) the file in the folder ~/.irssi/ (something like “rainbows.theme”). Once you have created the theme (or copied it), you can then set the theme in irssi using the command

```
/set theme <theme name>
```

where <theme name> is the name of the file (minus the .theme extension), so it would be

```
/set theme rainbows
```

for the example I gave above.

Basic commands:

```
/connect <server URL> - connect to a server (Freenode, DALnet, etc.)

/join <channel> - connect to a channel (#ubuntu, #kubuntu, etc.)

/quit – quits irssi

/disconnect – disconnect from server

/part <parting message> – leaves the channel (the parting message is optional)
```

I hope this article will have intrigued you to try out a few CLI programs. They are extremely lightweight, flexible, and fun to play with! I would especially recommend trying them out with a tiling window manager if you want to be as lightweight as possible. Something like Xmonad, Awesome, etc. is always a good experience to try out - especially if you're fond of coding and CLI apps. I find it the best environment for both. Even if you're not planning on using these programs, I'd recommend having at least irssi around in case you need access to help without a GUI being available.

Further Reading:
MOC - http://moc.daper.net
irssi - http://irssi.org

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

COMMAND & CONQUER

Written by Lucas Westermann

For this month's Command & Conquer article, I'll cover a few things that are sometimes mentioned online with instructions, or things that aren't enough for an entire article on their own, but should still be mentioned. I hope that this information is useful for those readers who want to do more with their shell, or who want to customize it, yet occasionally run across a term that they don't know.

I'll begin with prompt customization. Say you've spent a long time finding a prompt for your terminal that's to your liking, and you've finally gotten all the escape characters set and you're ready to try it out. This is my .zshrc PS1:

```
export PS1="%{$fg[blue]%}&#9484;&#9472;[%{$fg[green]%}%n%{$fg[cyan]%}@%{$fg[green]%}%m%{$fg[blue]%}:%{$fg[magenta]%}%~%{$fg[blue]%}]-%{$fg[red]%}[%{$fg[cyan]%}%*%{$fg[red]%}]%{$reset_color%}%{$reset_color%}"$'\n'"%{$fg[blue]%}&#9492;&#9472;>%{$reset_color%} "
```

which looks like the image below.

You have two ways to do this:
1) you can open a new terminal and view the shell (which, if you're running in a tty session, or are doing too much at once, or hate changing shells before you're finished, isn't a great option) or
2) you can “source” it. I prefer option #2 since it takes immediate effect in the original terminal, and is fairly simple to do. It is done by using the command “source” followed by the path (or name, if it's in your current working directory) to the configuration file (.bashrc in this case).

```
source .bashrc
```

This command works with most configuration files that you could edit, but there are some cases where it won't do anything at all (I can't say I know all possible uses of it, so you will have to try for yourself).

Another useful thing to know is that you can use while loops and the like in the actual shell too. For example, if you wanted to list all the files in a directory and insert it into a different line of code, you could do this:

```
ls|while read line; do `cat ${line}`; done
```

Of course, you will get an error message if you run into a directory, but that can be solved with a simple check (using an if statement). This, however, I will leave to you.

A couple of other useful commands that I find myself using a lot are:

```
df
```

which displays filesystem usage (I usually use it with the “-h” argument, so that it displays in Gigabytes).

```
watch
```

which runs a command once ever 2 seconds (by default, but can also be changed with “-n <num>” argument).

```
scrot
```

which is essentially a command-line based screenshot tool, but with a lot of options and possibilities (see the manpage for more info, there are too many options to cover here).

And, of course, the other commands I've covered in the past months are also frequently used, but the 3 above haven't been mentioned

before, and are useful. One last thing I want to cover in this article is what a tiling window manager is, since quite a few coders I know prefer them over normal window managers. A tiling window manager is a window manager that arranges all windows in “tiles” (re-sizing so that all windows fit into the space, and so that windows don't overlap). Some (not all) tiling window managers offer a “floating” mode, where windows act like they usually do (set size, overlapping, etc.). The reason why coders (myself included) prefer this behaviour is because it lets you view all your code at once, or to have multiple scripts open at the same time so that you can switch easily, or so you can have one terminal open to test commands and another to write the script. Not only that, but you can control the window manager using only the keyboard, allowing for faster working, since you never take your hands off the keyboard. A few such window managers are AwesomeWM, DWM, Xmonad, ratpoison, and ion. I use Awesome, since it offers a floating mode that isn't always on top or always below, but can be both (Xmonad seems to offer only one or the other out of the box, and I couldn't find a work around). However, there are lots of options, and most are well documented, in case any reader feels like giving one a go.

Further Reading:
Awesome - http://awesome.naquadah.org/
Xmonad - http://www.xmonad.org/
Ratpoison - http://www.nongnu.org/ratpoison/
Ion - http://modeemi.fi/~tuomov/ion/
DWM - http://dwm.suckless.org/

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

Live-Office is an open-source, Web-based, professional, groupware application that facilitates the organization of all your personal information. There is no need to install additional applications on your computer, for all the required components are installed on the Web server. One needs only an Internet connection and a Web browser with JavaScript enabled. Live-Office can be accessed on our server, or easily installed on your server. All your personal data will be centralized and securely stored in our, or your, online database, which will offer various modules and widgets that provide you with the ability to save all your personal data in one place. Events, ToDos, Contacts, Favorites, Documents, and Notes are a few examples of these. These modules and widgets can be accessed easily via a high-end, intuitive, user interface. This unique key feature helps you to gather and visualize all your information at the same time. For example, you can open your organizer and address book side by side, add a personal contact in your address-book window, and add a meeting in your organizer window.

With Live-Office, you can customize your user experience. You don't like blue as your background desktop color? Change it to a wallpaper. You prefer using applications in your native language? Live-Office comes with multi-language support.

We will soon be adding new modules and widgets, such as Budget and Billing, and Password reminder, as well as new languages. In addition, we are currently working on Live-Office Desktop Edition - an off-line version built in Java. This version will let you manage all your personal data off-line and synchronize it when needed.

Contact us or visit our website to know more about how you can contribute to Live-Office: info@live-office.net or: http://www.live-office.net

COMMAND & CONQUER

Written by Lucas Westermann

Recently a reader had requested that I cover the basic layout of a help or man (manual) page. For the benefit of this article, I will focus on the 'help' and 'man' pages for the “ping” command. The command:

```
ping -h
```

will display the help command for ping (the “-h” switch, along with the “--help” argument, are the defacto default). The help information will look something like what is shown in the box above right.

The first square brackets containing “-LRUbdfnqrvVaA” is a list of possible switches that don't require arguments (mainly because they format output), and for an explanation as to what any of these switches does, it's required to check the man page as well. The next series of square brackets that show a switch and value combination (e.g. “-c count”) requires you to substitute the “count” section with an actual value. The words are intended to give you an idea of what the switch does.

```
Usage: ping [-LRUbdfnqrvVaA] [-c count] [-i interval] [-w deadline]
            [-p pattern] [-s packetsize] [-t ttl] [-I interface or address]
            [-M mtu discovery hint] [-S sndbuf]
            [ -T timestamp option ] [ -Q tos ] [hop1 ...] destination
```

count – number of ping tries
interval – time between pings
deadline – maximum time to wait for reply
pattern – allows you to fill the ping with a pattern of bytes
packetsize – defines the size of the packet to send
ttl – sets the IP Time to Live
interface or address – Set the source IP or device
mtu discovery hint – ''do'', ''want'' or ''dont'' are options
sndbuf – defines the size of the send buffer
timestamp option – allows you to set special IP timestamp options
tos – sets the Quality of Service option (either decimal or hex value accepted)
hop1... - can be a list of destinations

Ideally, the help page is intended as a quick reference, in case you're unsure what switch corresponds to the input you wish to supply. If, however, you're new to the command and don't know what half of the switches do, it's best to read through the man page, since it offers an explanation of each and every switch, as well as possible uses, help website, etc. There are some help pages that offer descriptions of arguments and switches, as well as more information, but that isn't always the case.

To view the man page, use this command:

```
man ping
```

The synopsis section displays something similar to the help command, but it is followed by a description section, which explains what the command is intended to do, or intended to be used for. The options section lists all the switches, and a short description of what each does. To navigate through the rest of the manpage, use the up and down arrows, or page up and page down. The text on the very bottom looks like this: “Manual page ping(8) line 21/356 15%”. It tells you

that you are viewing the manual page for ping, and are at line 21 out of 356, or 15% of the way through the document. Once you're done viewing the rest of the options section, you come upon a section named “ICMP Packet Details”, which defines what they are. There are a few more definitions before you reach the bug heading, but each one is fairly well explained, so I will not explain each one. The bugs listed are currently open (known bugs that haven't been fixed in that version), and the See Also: section offers a few other commands to look at that will be useful in combination with the command you are currently reading up on. The history section is straight-forward enough, while security and availability tells you a little more about the command.

Not all man pages are laid out in the same manner, but they follow the same conventions, which are:

```
Name
Synopsis
Description
Options
Useful information pertinent
to the command (definitions,
explanations, etc.)
Bugs
See Also
History
Security (if applicable)
Availability
```

This is useful in case you want to look up a specific term from the help page, since you know exactly where to go. Also, if you ever write your own tool, or would like to add a man page to a script you wrote, you then know how to format it. Hopefully this article has helped you to get a better understanding of how manpages work, and how to make sense of the often confusing help pages.

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

COMMAND & CONQUER

Written by Lucas Westermann

Have you ever found yourself tapping an extra key on your laptop, only to realize that it isn't bound to anything, and then finding out that the shortcuts program doesn't recognize the key? Well, I ran into a similar problem when setting up Openbox on my netbook, and so I thought I would show you how to bind keys to functions, even if it seems that they are not recognized.

Step One: Key Recognition

First we need to find out if the key is recognized by the kernel. Open a terminal and run the command:

```
xev | grep -A2 --line-buffered '^KeyRelease' | sed -n '/keycode /s/^.*keycode \([0-9]*\).* (.*, \(.*\)).*$/\1 \2/p'
```

This will output the keycode followed by the keypress name (XF86AudioMute, XF86MonBrightnessDown, a, b, and so forth). If it displays NoSymbol after the keycode, there is no keypress bound to that keycode yet, and you can skip to step two. If neither the key returns nothing, it is time to try showkey.

Switch to tty0 (by hitting ctrl + alt + F1) and log in as your user. Once logged in, enter the command:

```
showkey
```

This program will return keycodes of keys pressed, and automatically quit 10 seconds after the last keypress. Once the command is run, hit the key(s) you want to test, and record any keycodes that it returns. If neither of these options returned a keycode, it's time to see if the key has a scancode.

To do this, press the key you want to test, and then check dmesg with:

```
dmesg|tail -5
```

If something like this appears in the dmesg output -

```
atkbd.c: Unknown key pressed (translated set 2, code 0xf1 on isa0060/serio0).

atkbd.c: Use 'setkeycodes e071 <keycode>' to make it known.
```

- you can map the scancode to a keycode. You can do this by either using HAL or setkeycodes (kernel tool), as shown in the dmesg output. The further reading section at the end of this article offers a link to HAL's keymap quirks page. I will not go into detail in the article, since it is quite rare (in my experience) that it is necessary to do this.

Step Two: Binding Keycodes

I will focus on binding keycodes to keys in Xorg, since most multimedia keys aren't required in the tty0 console. To start, you must create the .Xmodmap file. This can be done by using the touch command, or just editing it in gedit and then saving the file. Entries in the file should be in this format:

```
keycode <Xkeycode> = keysymbol
```

A few examples would be as below:

```
keycode 153 = XF86MonBrightnessDown

keycode 154 = XF8MonBrightnessUp
```

Step Three: Testing Keycodes

First run the command:

```
xmodmap ~/.Xmodmap
```

Then you should be able to add the keys to whatever function you need. If not, revise the keycodes and keysymbol names, just in case

you made a typo. A full list of symbols can be found here:

```
/usr/include/X11/keysymdef.h
```

And for extra function keys:

```
/usr/include/X11/XKeySymDB
```

Once you have ensured that the keys work, continue to step four.

Step Four: Making It Permanent

To make the changes permanent, you have to run the xmodmap command every time you log in. I would recommend adding it to your .xprofile.

An alternative tool to xmodmap is xbindkeys, and it is fairly straightforward. There is a GUI available called xbindkeys_config, but I'm not sure if it's in the Ubuntu repositories.

Further Reading:
HAL keymap quirks:
http://people.freedesktop.org/~hughsient/quirk/quirk-keymap-index.html

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

QUICK REVIEW – PREY

In the world of the first person shooter (FPS) video game there is very little innovation. Most of the time a new FPS game will have elements that made older, or previous, games in this genre a success.

That is not the case with Prey. Prey is unique. Prey is different from anything you have played before. Recently ported to Linux, this old favourite of mine is now enjoying a new lease on life.

You play a Cherokee warrior named Domasi Tawodi (a.k.a Tommy), a man who wants to leave his Cherokee heritage in the past, leave the Reservation and move on into the civilized world, but there is one problem: his girlfriend Jen wants to stay, because it is her home. All of a sudden, while they are talking about it in a bar, it gets ripped up by an alien ship taking him, Jen and his grandfather onto it so its inhabitants can feed on them.

Obviously, Tommy would not give up his life so easily, so he tries to rescue his Grandfather and girlfriend.

One thing that makes this game outstanding (apart from the fact it uses a heavily modified Doom 3 engine) is its use of gravity (if you take a look at some of the screen shots you'll see why) and portals (which makes it very possible to shoot yourself if you don't know what you are doing). These are used in a number of short and long puzzles, but nothing the average part-time gamer can get stuck on. Another aspect of the game I loved was the fact after a certain point in the game you cannot die. You read it correctly, 'you cannot die' so you will no longer 'die' then quickly press your quick-load key then try again with a miniscule amount of health. I don't want to give away too much about this game other than the fact that it is awesome.

Ten years plus in development and this is what we get: a really slick game. It is one of the few examples of the modern video game that I think will go down in the record books as a definite classic.

Christopher Hart

Rating: 9/10

COMMAND & CONQUER

Written by Lucas Westermann

I recently got back from my summer vacation, and, after roughly 300 packages were updated, I noticed in conky that the root partition was getting to be pretty full. So, I thought that it might be useful to write an article on a few tips I have picked up over the years that I use when a hard disk gets full.

Starting off, we'd most likely need to check to see how much space is left on the disk. To do this, I use the command-line tool “df”. This check can also be done in gparted, but I will focus on the command-line aspect. So, down to business. In order to see a list of all mounted filesystems and their usage, use the command:

```
df -h
```

This will print out a list of mounted partitions, how much space is used, how much is free, the percent used, and the mount point. I think it's fine to use a disk up until it's 90% full (your home folder can usually safely be at around 95%). The root partition requires some space to be left free for logs, root folder, and so forth, and will warn you when it gets “full” (all the space is used up except for what is set aside for logs). This generally means you can't install any more packages or move any files around, which we don't want. If you find you can't free up sufficient space, you may need to re-size the partition itself in order to get enough space. I generally am fine with about 10GB for my root partition (my home partition is generally 25GB or more).

If the root partition is the one getting full, your first step should be to clear out extra cached packages (aptitude, apt-get, and synaptic all store downloaded packages in the cache so that it doesn't need to re-download them if you re-install the package). Open a new terminal window (leaving the output of “df -h” visible), and run

```
sudo aptitude autoclean
```

or

```
sudo apt-get autoclean
```

to clear out all packages from the cache that are no longer downloadable (read: out of date). Once it's completed, I then run “df -h” again in order to check to see how much space was freed. If it has freed up enough space, I leave it at that and move on. If, however, it freed up hardly any space, and you run Ubuntu without the backports repository, beta, or lots of cutting-edge packages, you could likely get away with clearing out all cached packages. I used to do this before I switched to Arch, since Ubuntu tests the packages thoroughly and leaves the stable ones in the normal repositories. If, however, you use getdeb repositories or such, I recommend not doing this in case you find an issue that causes you to downgrade. However, do not do this regularly, simply because you may one day need the cached package again for whatever reason. You can clear out all packages from the cache by issuing:

```
sudo aptitude clean
```

or

```
sudo apt-get clean
```

As a side note, the reason why I list both aptitude and apt-get commands is simply because I prefer using aptitude for my cleaning purposes, and I haven't used Ubuntu for a while, and as such don't know if apt-get does it similarly now or not.

So, after clearing out the packages you should now have a bit of extra space. If you still need some space, or the root partition wasn't the one that needed more free space, you

can list your files and folders by size, and manually delete large files you no longer need (old .ISO files, archives, icon themes, untarred archives, etc.). I picked up this trick from Linux Journal (not sure which issue anymore) -- it works well. First, if you have both root and home on a separate partition and you want to free up space on the root partition, do the following:

```
cd /
du -ckx|sort -n
```

What this does is first change the directory (cd) to your root partition, and then display disk usage ("du") with size blocks of 1kb ("-k"), and display a grand total ("-c") for the partition (no changing to the home partition, etc.). This will all be sorted from smallest to largest (so the last file listed is the largest) due to the "sort -n" command we pipe the du output to. I don't recommend deleting anything from the root partition without thorough investigation of what it is (unless it's the cache of a program you no longer use, when it should be safe to remove -- but again, it's best to check first). This is simply due to the fact that you can cause serious system errors by just deleting away. Secondly, if you want to display information in your home partition/folder, you can run the following command:

```
cd ~
du -ck|sort -n
```

This will show the size of all folders and files within your home folder, sorted from smallest to largest. Once you find out which are the largest folders, you can find out what size the folder is (in KB/MB/GB) by running:

```
du -h ~/<foldername>/
```

The "-h" switch stands for human-readable. Also, the tilde (~) stands for /home/<username>/ (saving you some typing), and you need to replace the "<foldername>" with the actual name of the folder you want information on. Once you've found out which folders and files are hogging all the space, you can choose to delete the ones you no longer need (or to backup the ones you want to keep, but don't use, to a different storage medium). After all this, you should have freed up a bit of space, and can continue to install packages!

I highly recommend going through a list of installed packages at some point though, and deciding if you wish to get rid of some you don't use. This can be done, quite simply, with aptitude (since it shows an "i" if the packages are installed). Run the command:

```
sudo aptitude search '~i'
```

or

```
sudo apt-cache search '~i'
```

A quick note on the apt-cache command: I am not sure if it actually works in the same way aptitude does. Also, before uninstalling any packages you don't recall installing, check first! It may very well be a package Ubuntu requires.

I cannot stress enough to be extremely careful in what you delete or uninstall, especially if you're not sure what it is. It's always better to be safe than sorry.

I am open to requests on articles, so if you run into a command-line issue you think others might be experiencing, drop me an email with the issue, and I may write an article on it. The submitter will, of course, get credit, unless specified..

Further Reading:

Apt-get cheat sheet:
http://www.cyberciti.biz/tips/linux-debian-package-management-cheat-sheet.html
Aptitude search '~i' info:
http://www.linuxquestions.org/questions/debian-26/aptitude-how-to-get-a-list-of-all-installed-packages-458119/#post2310207

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

COMMAND & CONQUER

Written by Lucas Westermann

Before I start on the focus of this article, I'd like to take a moment to thank David Rowell for pointing out that another space-waster on some systems is the thumbnails directory (this applies only to systems where thumbnails are generated). In Ubuntu, the default directory is ~/.thumbnails. I believe, however, that Thunar (on Xubuntu) stores it in a different location - the same is so for Konqueror on Kubuntu. The thumbnails don't get removed once the image/video that the thumbnail applies to is removed, at least, this was the case in Gnome 2.24/2.26. So, if you store/stored a lot of media on your hard disk, chances are the thumbnails folder can be rather large. To solve this, simply delete the directory with

```
rm -r ~/.thumbnails
```

and the next time you open a folder with media in it, the thumbnails will be regenerated, which could take a few minutes (depending on the number of files and the CPU of your system). If you're not sure how big the thumbnails folder is, you can check using my tip from last month's article:

```
du -h ~/.thumbnails
```

Now, on to the topic of this month's article. I know many people use Ubuntu, or another form of Linux, on notebooks these days, and so I thought it could be useful to cover how to disable power management for hard disks, which can cause a lot of wear and tear on notebook drives. There are threads on most distributions' forums regarding this issue, and, as far as I know, it hasn't been solved. The downside of this fix is that the hard drive doesn't spin down. This can cause data loss if the laptop is dropped (especially if the hard drive is in the process of writing), and can also cause your laptop to be a few degrees warmer, since the hard drive generates heat. The positive aspect of this is that the hard drive will last much longer than it will with the power management on, and the hard-drive performance will increase a little. I will be covering how to see if your laptop is affected by this bug, and how to disable power management. Also, I'll cover how to use smartmontools to check the health of your hard disk.

Before we start checking any values, you must first install the tool we'll be using. Smartmontools is in the main repository of most distributions, including, of course, Ubuntu. To install it, run the following:

```
sudo apt-get install smartmontools
```

Once it's installed, you should probably check the S.M.A.R.T. (Self-Monitoring, Analysis, and Reporting Technology) values of your hard drive by running the following command:

```
sudo smartcl -H /dev/sda
```

You should replace /dev/sda with whatever hard disk you want to check. This will return information in the following format:

```
smartctl version 5.38
[x86_64-unknown-linux-gnu]
Copyright (C) 2002-8 Bruce
Allen
Home page is
http://smartmontools.sourcefo
rge.net/

=== START OF READ SMART DATA
SECTION ===
SMART overall-health self-
assessment test result:
PASSED
```

As you can see, my laptop's SMART hasn't been tripped (meaning the hard disk's health is fine). If it says the hard disk didn't pass, you may want to think about replacing it in the near future. If it says your hard disk doesn't support SMART, then you can stop

worrying, since you will be unable to change any of the settings.

Before proceeding to the next section, I will take a moment to note that you should read the link to the UbuntuForums thread below before applying any of these fixes, since it should be done only when you have a good understanding of what is happening. Also, you have to take into account the period of time you've had the hard disk, etc. If you choose to follow the information in this article, you are doing so at your own risk. I am writing this article from the experience that most laptops I've used have required this fix. This experience includes explaining the steps of this fix to laptop owners. Be advised that some newer laptops may not need this fix, and may even suffer a shorter hard-drive life if it is applied to them. The Web has lists of laptops that suffer the power-management problem. These can tell you whether other owners of your laptop model have reported this problem in their machines.

In order to check the start of the Load_Cycle_Count, type the following command:

```
sudo smartctl -a /dev/sda|grep Load_Cycle_Count @
```

This will spit out one or two lines of code that look like this:

```
225 Load_Cycle_Count 0x0032 099 099 000 Old_age Always - 14091
```

The first number is the ID#, the name is the ATTRIBUTE_NAME, the hexadecimal string is the FLAG, the first value (099 here) is the VALUE, the WORST is the next 099, the 000 is the THRESH, the Old_age is the TYPE, Always refers to UPDATED, the "-" is in the WHEN_FAILED column (would be a date, if the hard disk failed), and the 14091 is the RAW_VALUE. Now, I'll explain what some of these terms are. The VALUE is the SMARTCTL percentage-type value. The WORST is the lowest stored value in the life of the hard disk, and the THRESH is where SMART decides that the hard disk is failing (so once VALUE reaches 000, it's failing). The TYPE refers to the type of THRESH (choice between Pre-fail, where it warns you before the hard disk fails, and Old_age, where the hard disk will have simply run the course of its life). UPDATED is how often/under what conditions the attribute is updated, WHEN_FAILED shows the date at which the attribute passed the THRESH level, and the RAW_VALUE is how many times it actually occurred.

Anyway, record your RAW_VALUE somewhere for safe keeping, and check the value again at a later date. The best way to check would be to write a simple script to run as root in CRON once a day at the same time to give you an idea of how often it's increasing. You can also, however, check manually how much it increases in 5 minutes, etc. If it increases by more than 5 increments in 5 minutes while the laptop is being used, chances are that it's not giving you a proper value, and you could then divide the RAW_VALUE you have by the increase (so if it's increasing by

10 each minute, divide by 10). Once you have an idea of how much it increases on average (per day), you should then calculate how much the value will be in 3 years (average lifespan of a hard drive), taking into account, of course, how long you have had the laptop! If the value is under the Load_Cycle_Count that the hard disk should be able to handle (it's generally around 600,000 but you should Google your hard disk's Load_Cycle limit just to be sure), then you

will not need to worry about the fix. If, however, it greatly exceeds the limit, you should apply the fix in order to keep your hard drive running for as long as possible. For example, my Samsung n110 (running ArchLinux) increases at a rate of about 1 per minute, so per day it's an increase of 1440, 1440*365=525600, 525600*3=1576800. I didn't, however, account for the fact that the laptop is about 4 months old. Since the number is so large, I decided to not bother finding a more-accurate value, since it wouldn't make too much of a difference. This value is well over any reasonable limit for hard disks, so I've turned the APM option off. In order to do this, run the command:

```
sudo hdparm -B 255 /dev/sda
```

Or, if you want to just set it on the lowest possible setting (waits the longest period of time before going into power-saving mode) run:

```
sudo hdparm -B 254 /dev/sda
```

In case you ever want to undo this, the default setting of APM for most hard disks is 128, so running

```
sudo hdparm -B 128 /dev/sda
```

will set the APM back to its default setting.

This concludes most of what I wanted to cover. If the fix works for you and decreases the Load_Count, then you may need to add a script to run it on boot up, but this is covered in the thread I listed below. Also, I urge any reader who isn't 100% sure that it's required on their hard drive to read through at least some of the thread, in order to grasp a better understanding of this process.

For those of you who are wondering why I included this information in an article after warning the reader repeatedly that it shouldn't be used lightly, the answer is simple: the smartctl command is extremely useful. It can give you lots of information about your hard disk, and it can offer you information on the status of your hard disk's life. I added the information about APM simply because it uses a lot of the commands that I use to check hard-disk life/information, and because it is a useful thing to be made aware of. I'm not saying anyone should just follow the instructions; I am making the reader aware of the possible issue, and offering a way to check/fix it, in case she finds it's necessary. If you buy a new laptop/laptop-hard-drive once a year, and are fine with it, then chances are you won't need to even consider this. That being said, not many people will do that. I hope the introduction to smartmontools was useful for everyone, and that the explanation on the Load_Cycle issue was useful for some (hopefully, fewer than it would have been a year or two ago, but who knows?).

As the last note of the article, I am, as always, open to suggestions, questions, comments, opinions, and pretty much anything else to do with the CLI. If you're a reader who has any of the above, feel free to email me at lswest34@gmail.com. Be sure to include the word “FCM” in the title and refer to the title of Command & Conquer in the subject header (just to ensure that I read it). I'd also like to take a moment to point out that this is my 10th Command & Conquer article. Thanks to anyone and everyone who has been following this series since I started writing it, after taking over for Robert Clipsham.

Further Reading:

Official Ubuntu thread on load_cycle_count: http://ubuntuforums.org/showthread.php?p=5031046

hdparm manpage, accessed with:

```
man hdparm
```

smartctl manpage, accessed with:

```
man smartctl
```

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

COMMAND & CONQUER

Written by Lucas Westermann

Before I start on the topic for this month's article, I have to admit to a mistake! Reader Stefan Eike pointed out that I missed out a “t” last month, in the command:

```
sudo smartctl -H /dev/sda
```

So thanks to Stefan for pointing that out, and sorry to anyone who may have run into issues with that command.

I got an email from proofreader Brian Jenkins on November 15th, offering his opinion that an article dedicated to GNU Screen would be cool to see, since he had started using it and felt it extremely useful. So, Brian, here's your article! I have to thank him again for reminding me of Screen, I seem to have always managed to overlook it when deciding on an article. After he suggested this topic, and I decided that it was a great idea to write an article or two about (I will most likely be doing a follow-up article next month with a bit more information about Screen), I decided I'd use Screen as much as possible for the weeks that followed, and to configure it as best as I could - after all, you can't write about a program you never used!.

In this article, I'll focus on installing, using (keybindings, etc.), setting up a .screenrc, and the pros/cons of Screen. Next month's article will be focusing on more advanced uses of Screen (multi-user sessions, Screen over SSH, etc.). That way, everyone should have the knowledge required to understand the next segment, and I can focus more on the how and why instead of the usage of Screen. So, to begin with, what is GNU Screen? GNU Screen is a terminal multiplexer. In case that means absolutely nothing to you, a terminal multiplexer essentially creates a series of “virtual” terminals within a terminal emulator/tty screen, and these virtual terminals can be attached/reattached in a new terminal, or a different account, etc. You may be asking yourself: “Why not just have two or more terminals open?” Which works, and, I have to admit, I am in the habit of using multiple terminals, but Screen offers you the ability to have multiple virtual terminals in a single screen session, which act a bit like tabs (yes, I know there are tabbed terminal emulators as well). However, Screen also allows you to detach and reattach the entire session (tabs included) in a new terminal, in a different account, or in a tty screen.

Of course, the best way to find out what Screen is, is to actually install and use it. In order to install Screen on your system, you can run this command:

```
sudo apt-get install screen
```

Once it's installed, you can get your first taste of Screen by simply running it with:

```
screen
```

You'll notice that it opens...a blank terminal? Screen looks exactly like a terminal (if run without arguments/configuration), yet you can see that it is actually Screen by hitting C-a d (that is: “ctrl + a”, and then “d”). You'll now see the terminal you had open before with a line that reads:

```
[detached]
```

Which is simply telling you that the screen session that was started was detached, and not killed.

Now, for a complete list of keybindings for Screen, you'll have to check the link in the Further Reading section. A few that I find myself using a lot are:

```
Ctrl + a, d — detaches a screen
```

```
Ctrl + a, 0-9 – switches to that virtual terminal inside a screen session

Ctrl + a, Ctrl + a – Toggles to the previous window

Ctrl + a, Ctrl + c – creates a new window with a shell and switches to that.

Ctrl + a, k – kill current window (close the window)
```

Once you've detached your screen, you may be wondering how to get it back. If you enter the following command into the terminal, you'll be presented with a list of screens:

```
screen -ls
```

My list looks something like this:

```
There is a screen on:

17153.pts-0.lswest-netbook (Detached)

1 Socket in /tmp/screens/S-lswest.
```

Or, if I enter the command from within the screen session:

```
There is a screen on:

17153.pts-0.lswest-netbook (Attached)

1 Socket in /tmp/screens/S-lswest.
```

After seeing that list, you may be a bit confused. Essentially, it's listing the files each screen sessions creates in /tmp/screens/S-<username>/. It also displays the state of that screen (attached, detached, etc.). In order to re-connect, or "attach" a screen session, you have to enter the command:

```
screen -r <name of screen>
```

So, for the example above, the command would be:

```
screen -r 17153.pts-0.lswest-netbook
```

Of course, we're lazy, and so we'll stick to just using the numerical ID (17153, in this case). The ID should be sufficient for accessing a local screen session, however, I believe the rest will be required if you are somehow remotely connecting to a session.

One slightly more advanced thing to suggest, that people might find useful, is to have a screen window number in their Bash or Zsh prompt (since I'm an avid fan of Z-Shell). You can do that by adding the "$WINDOW" variable to the prompt line, so that it displays the value of the currently open window (e.g. If you have 3 windows open in a screen session, and you're in a shell on screen 1 (it counts from 0, so 1 would be the second one open), the value displayed will be 1). My prompt is set up using the text shown below.

This is a Z-Shell prompt, so it won't work for a Bash setup, but it gives you an idea of how I use it. Basically, the file checks to see if $WINDOW

```
if [ x$WINDOW != x ]; then
    #&#9484;&#9472;[5:lswest@lswest-netbook:~]-[15:21:07]
    #&#9492;&#9472;>
    export
PS1="%{$fg[white]%}&#9484;&#9472;[%{$fg[cyan]%}$WINDOW%{$fg[white]%}:%{$fg[green]%}%n%{$
fg[cyan]%}@%{$fg[green]%}%m%{$fg[white]%}:%{$fg[yellow]%}%~%{$fg[white]%}]%{$fg[yellow]%
}-
%{$fg[red]%}[%{$fg[cyan]%}%*%{$fg[red]%}]%{$reset_color%}%{$reset_color%}"$'\n'"%{$fg[wh
ite]%}&#9492;&#9472;>%{$reset_color%} "
else
    #&#9484;&#9472;[lswest@lswest-netbook:~]-[15:21:07]
    #&#9492;&#9472;>
    export
PS1="%{$fg[white]%}&#9484;&#9472;[%{$fg[green]%}%n%{$fg[cyan]%}@%{$fg[green]%}%m%{$fg[wh
ite]%}:%{$fg[yellow]%}%~%{$fg[white]%}]%{$fg[yellow]%}-
%{$fg[red]%}[%{$fg[cyan]%}%*%{$fg[red]%}]%{$reset_color%}%{$reset_color%}"$'\n'"%{$fg[wh
ite]%}&#9492;&#9472;>%{$reset_color%} "
fi
```

returns a value, and if so, it displays it in the prompt, otherwise it doesn't. The commented sections display the appearance of my prompt for either option. I find it a useful little thing to do when using Screen.

The last thing to cover for this month is the creation of a .screenrc file, in order to change defaults and settings of Screen. My .screenrc file looks like the text shown right (based heavily off rson's .screenrc from the ArchLinux forums).

The comment above "hardstatus alwayslastline" is an example of what the final result looks like. All the other commands are fairly well commented. The resulting screen looks the prompt below.

This is a basic .screenrc, and it would take an article or two to cover even half of what you can do with those configs, so I'll just leave the .screenrc as it is with comments, and check the further reading for a link to a site that attempts to explain all the possible settings for .screenrc files.

The very, very last thing I need to cover in this article is how to quit screen. This can be done two ways:

1. Ctrl + a, \ - quits screen and kills all windows

2. close all windows except for a shell, and then just type

```
exit
```

If anyone has any more questions, or would like to request an article covering an aspect of Screen, feel free to email me at lswest34@gmail.com. The same goes for anyone who has article ideas of any sort, or any questions about the CLI. I wish everyone happy holidays, and a good new year.


```
# Screenrc - Screen config file
# Author: Lswest
# Created: 24-11-2009 16:08:50
#
#

# General Settings

startup_message off          # Disable startup message
vbell on          # Give visual alert instead of sound
defutf8 on                              # Always use utf8

# Hardstatus

backtick 10 1 300 "/usr/bin/updateCheck" # List number
of available updates

# 0 Zsh   1 IRC           --INSERT--  No Packages to
Update
hardstatus alwayslastline "%{= dd}%-w%{+u}%n %t%{-}%+w
%=%{= dW}%h%{-}%20`%10`"

# autostart screen sessions
screen -t Zsh 0 /bin/zsh
screen -t IRC 1 /usr/bin/irssi
#
```

Further Reading:

http://www.gnu.org/software/screen/manual/html_node/Default-Key-Bindings.html#Default-Key-Bindings – The manual page for keybindings on the GNU homepage.

http://www.math.utah.edu/docs/info/screen_9.html – Short and concise list of things for .screenrc files, and Screen in general

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

COMMAND & CONQUER

Written by Lucas Westermann

Now that we've covered the introduction to GNU Screen last month, we're ready to advance into slightly more useful configurations. I'll be covering only a few aspects of Screen that I use and find useful, but they are by no means as advanced as they get, nor are they the only aspects worth using in Screen. I'd be happy to have readers send in their favourite configuration/setting for Screen. I can post them at the start of Command & Conquer each month, so that we can learn something new from them. The things I'll be covering in this article are the following: automatically starting windows with commands, using Screen over SSH to daemonize commands, sharing Screen sessions, splitting the window, and the benefit of Screen over TTY screens or a normal shell. So, lets get started and fire up our Screen sessions, and our .screenrc files!

If you were looking through my configuration file last month, you must have noticed the following two lines at the end of my .screenrc:

```
# autostart screen sessions
screen -t Zsh 0 /bin/zsh
screen -t IRC 1
/usr/bin/irssi
```

The lines there add a new window (the number after the title) to Screen with the title (specified after the “-t” flag), and the following command. So, the .screenrc launches Screen with window 0 running my Z-shell, with a title of “Zsh”, and window 1 with irssi running, and a title of “IRC”. Screen will default to the newest window, so my Screen sessions always start in IRC, since that's what I'll most likely use. I don't know if there is a limit to how many windows Screen can have, but I would recommend not having more than the number of shortcuts to quickly switch between them (so a maximum of 9 windows). This feature is especially useful if you're the kind of person who regularly uses certain CLI-based programs (mutt, irssi, midnight commander, etc.), and want to have them readily available in one easy-to-access window without opening a horde of terminals.

If you're an SSH user, you've probably run into the problem where you access a machine, and run a command, and you end up losing your connection, and then frustrate yourself by having to re-run the command since you lost the output. This is why I, personally, feel that Screen (or another terminal multiplexer) is a must for any SSH users who will be connected for longer than a few minutes, and who may need to run more than one command. Even if you're one of those one-command and a few-minutes users, Screen might still be a very useful thing for you. If you connect via SSH, and run Screen, you can set up the commands to run, detach the Screen, and disconnect from the SSH server. If you want to re-connect and get the info back, all you have to do is re-attach the Screen session. I recommend detaching before disconnecting from SSH, but Screen should automatically detach and keep the Screen running when the connection closes. This is useful for system administrators who may need to run a script to update permissions or whatnot, and will save them the need to send the process to the background, or to keep the SSH connection open the entire time. If I remember correctly, you can even configure the SSH shell to automatically run Screen for any SSH login, meaning you're always going to be in a Screen shell.

Another extremely useful feature of Screen is the ability to share sessions. This is great if you're editing a script and need input from another user.

You can allow them to SSH to the box, and share the screen. To do this, the host (first user), has to do the following:

```
Ctrl + a
:multiuser on
```

The Ctrl +a is the actual keyboard combination, and you have to then type ":multiuser on". You then need to allow the remote user to connect to the Screen session with the following:

```
Ctrl +a
:acladd <ruser>
```

Substitute "<ruser>" with the username for the remote user. Once you've done that, the remote user can connect to the Screen session using:

```
screen -x $USER/<screen ID/name>
```

You need to replace $USER with the username of the original user (the "host"), and the Screen ID/name with the name or ID of the Screen session the user started. If you want to know how to set a Screen name, I'll be adding a few quick tips at the end of the article on securing Screen, and adding names to Screen sessions to make management easier.

The very last feature I'm going to cover for the moment is the ability to split the window in Screen. This means you can have two shells running parallel with a tiling window manager feel to it. Screen only supports horizontal splitting out of the box; there is a patch to enable splitting vertically, but it requires you to re-compile Screen by hand. In order to split the screen horizontally in Screen, hit the following key combination:

```
C-a S
```

For those who didn't read my last article, that means ctrl +a, then S (shift + s). This will take the current window and split it in half down the center. Once you have your two panes, you can switch between them with:

```
C-a <Tab>
```

where <Tab> is the actual tab key. If anyone wants to enable vertical splitting, I'll post a link to a tutorial in the Further Reading section.

The last thing I want to cover for this article is why someone should use this instead of a TTY screen, or a normal shell. The simple answer is personal preference. The long answer is that it allows SSH sessions to run multi-task without any chance of losing the processes when disconnecting, and it also allows you to minimize the amount of RAM being used. Also, it allows you to have a better overview of what is running - if you give the Screen windows titles, and keep them running in one terminal emulator, you'll have a status-bar type list of running programs. I'm by no means saying you have to use it, but for those people to whom the features of Screen appeal, I highly recommend using it and getting into the habit of using it regularly (which, I have to admit, I haven't managed yet). As always, feel free to email any questions you may have to lswest34@gmail.com. Any suggestions for new articles can be sent there too.

Last command for Screen:

```
screen -S <name of screen session>
```

The above command creates a Screen session using the name you supply after the -S flag, for quick and easy access (great for when you're running lots of Screen sessions).

Further Reading:

Vertical splitting: http://scie.nti.st/2008/8/22/gnu-screen-with-vertical-split-support
http://www.gnu.org/software/screen/manual/html_node/Default-Key-Bindings.html#Default-Key-Bindings – The manual page for keybindings on the GNU homepage.
http://www.math.utah.edu/docs/info/screen_9.html – Short and concise list of things for .screenrc files, and Screen in general

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

COMMAND & CONQUER

Written by Lucas Westermann

Before I begin this month's article, I want to take a moment to explain that, as of next month, I will be expanding Command & Conquer to both GUI and CLI applications, tips and tricks, and general know-how. This is due to a number of reasons, but the main reason is that I feel it will speak to more of our reader base that way. This means that, in the future, my articles will cover both the GUI and CLI ways (where applicable). It may also necessitate a change in name, but unless I come up with something just as catchy, chances are it will remain the same.

Correction: In last month's Command & Conquer I referred to a "where" command. Turns out on my system I aliased "which" to "where" ages ago - and forgot about it! That means: if you had any issues with the "where" command, it's because it was actually "which". Sorry about any confusion! And thanks to both *Harold* and *Sue* for pointing this out to me.

This brings us to this month's article. I want to cover just a few basic and essential (in my opinion) commands that everyone (command-line shy or not) should at least be aware of, due to their prominence and flexibility in Linux systems. I also want to take a moment to stress that, regardless of your personal opinion of the command-line, solutions are often presented in a command-line way online, due to the fact that it applies to a wide range of distributions, as opposed to GUI methods that require you to be running a specific desktop environment or set of programs. So while you may choose to not use the command-line on your own, you may be forced to at some point, and having a rudimentary understanding of what you're doing is extremely useful.

Here are some of my "essential" commands, and a description of each:

• **man**: displays a manpage (like a manual) for a specific command (usage: man <command>)

• **vi**(m): Vi (or the Vim derivative) are basic text editors that offer a lot of functionality, and allow you to edit files without starting up a GUI app. Due to the command strings for saving, exiting, and opening files, beginners may feel more at home with "nano".

• **ls**: lists all files within a directory (by default only non-hidden files, but command-line arguments can change that).

• **cd**: stands for change directory and allows you to navigate a file tree (usage: cd <directory or path>)

• **ping**: a useful program to test network connectivity/status of a server. (usage: ping -c 5 www.google.com) – supply the -c 5 if you don't want ping to run endlessly.

• **iwconfig/ifconfig**: commands used to configure wireless and ethernet devices from the command-line (wpa_supplicants are required for wpa networks, and dhclient for obtaining an IP address). Due to the large number of options, please see the manpage of iwconfig or ifconfig for usage.

• **halt**: a program similar to

> **Here are some of my "essential" commands, and a description of each...**

"shutdown -h now", but requires no further information. (Usage: sudo halt). If it's not available on your system, use shutdown.

• **alias**: extremely useful in shortening long command and argument strings you use often. Usage: alias <alias_name>="<command>")

• **rsync/scp**: allows you to copy files from a remote host over ssh (or within a local system in the case of rsync). Rsync also offers a progressbar, information on files, and much more information than is available for cp.

• **cp/mv**: cp allows you to copy files from one directory (or an entire directory) to another, and mv allows you to move a file or directory (either to rename it, or to move it to a new location).

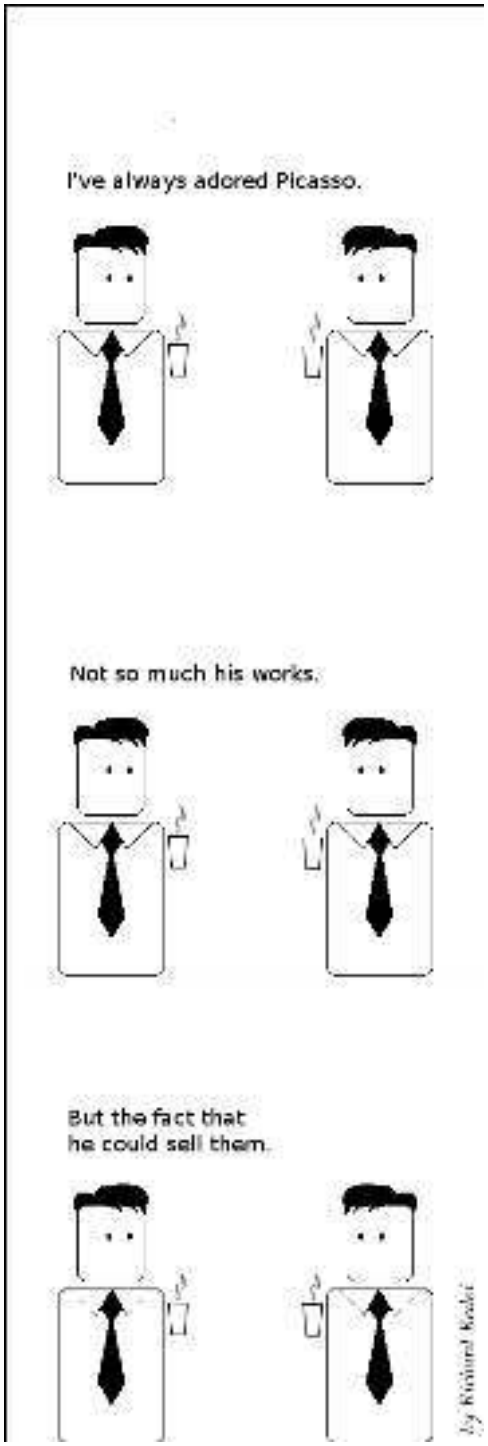
• **cat**: a program that returns the contents of a file (useful in

combination with less or more when dealing with large files). Extremely useful for quick log reading (an example: sudo cat /var/log/errors.log|tail -150|more). This will show the last 150 lines of /var/log/errors.log, and display it in more (allowing you to slowly move through the list). For more on troubleshooting errors, see C&C in FCM23, page 6.

- **rm**: allows you to remove files or full directories (use rmdir for empty directories).
- **su**: allows you to switch users from the command line (usage: su <username>)
- **locate**: allows you to quickly find files on your computer (run sudo updatedb beforehand for best results)
- **ln**: allows you to create hardlinks and softlinks of directories or files (for softlinks, use the "-s" switch). Softlinks are like shortcuts, and hardlinks are physical references on the disk to another location (think of the original file as directions from place A, and a hard link is a set of directions to the same end-position, but from place B). Or, for monopoly players among you, hard links are "pass GO" transitional cards and softlinks are "go to <place>, do not pass GO".
- **echo**: like the php command of the same name, this just returns whatever string is supplied afterwards. Most frequently used for quickly adding a line to, or creating, a file.
- **pwd**: print working directory is a command that displays the location you're at on your filesystem.
- **mkdir**: command to create a directory.
- **touch**: creates a blank file at the specified location (usage: touch <path and file name>). If the file exists, touch will not replace it.
- **grep**: allows you to search within output (or a file) for a specific string.
- **find**: a slower, more thorough search than locate. (Usage: find <path> <arguments>)

I realize this seems like a long, long list, but there are some commands I included that are duplicates, and some that are only ever used within another command. Besides, if you get stuck in a command-line environment, these commands may very well help you get back to a GUI environment. There is one command that I haven't included, since it's a non-standard command. I always install a text-based web browser, just in case. I personally prefer lynx, but links, elinks, and w3m are all very useful. If you know even a third of this list, you should be prepared to solve (or at least troubleshoot) most xserver problems, in order to restore your GUI environment. If you have another "essential" command that you think I should cover/mention, send me an email at: lswest34@gmail.com. And, as always, please include "C&C" in the subject line. I hope you've enjoyed this article, and feel better prepared for any command-line work you may end up doing.

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.



COMMAND & CONQUER

Written by Lucas Westermann

After Issue 33 was released last month, Ubuntu member Chris Johnston was kind enough to send me an email pointing me to Byobu (https://edge.launchpad.net/byobu). I admit I haven't had quite as much time to play around with it as I would like, but I have gotten a general idea of what it offers. Also, before I started this segment, I saw a lot of mentions of tmux on the Arch Linux Forums, so I'll be covering that here as an alternative. I won't be going into much detail on the features, as Byobu is similar to Screen's key bindings and settings, and tmux is extremely well documented, and my configuration file is commented and should be clear enough. Requests for a more in-depth article on either can be sent to me via email, since I'm not sure how much demand there will be for a thorough walk-through of their functions.

Byobu

"Byobu is a Japanese term for decorative, multi-panel screens that serve as folding room dividers. As an open-source project, Byobu is an elegant enhancement of the otherwise functional, plain, practical GNU Screen. Byobu includes an enhanced profile, and configuration utilities for the GNU screen window manager, such as toggle-able system status notifications." (Taken from the Launchpad page.)

What this means is that Byobu isn't a re-write of Screen, but rather something that goes along with Screen, adding a few features. The main features I noticed that are different from Screen, is that by default, it comes with a status bar that offers more information on the system than Screen offers by default, as seen above right.

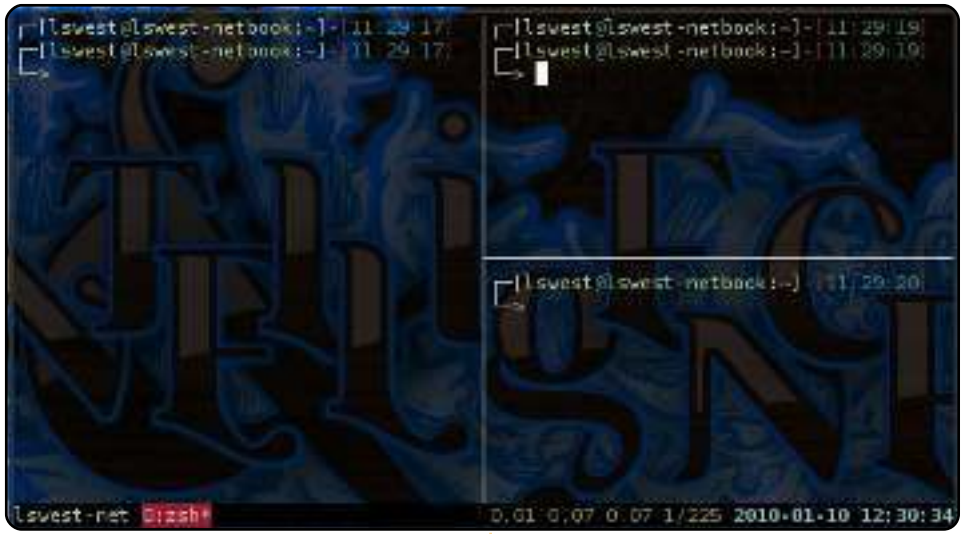
Also, Byobu offers an ncurses-based menu to create screen profiles. This can be accessed by hitting F9 (as seen on the task bar, "Menu: <F9>"). I won't add a screen-shot of the menu, since it's rather straightforward and self-explanatory.

tmux

"tmux is a terminal multiplexer: it enables a number of terminals (or windows), each running a separate program, to be created, accessed, and controlled from a single screen. tmux may be detached from a screen and continue running in the background, then later reattached. tmux uses a client-server model. The server holds multiple sessions, and each window is an independent entity which may be freely linked to multiple sessions, moved between sessions, and otherwise manipulated. Each session may be attached to (display and accept keyboard input from) multiple clients. tmux is intended to be a modern, BSD-licensed alternative to programs such as GNU screen." (From the homepage at http://tmux.sourceforge.net/)

What tmux offers—that is different from GNU Screen—are easier-to-understand commands, vertical and horizontal splitting, and sane defaults (status bar, etc.); moreover, it can be changed dynamically from the command-line, and it requires less memory than GNU Screen. It requires about 2.4 MB of memory for the first session, but each new window requires only 1 MB of memory. May not seem like much, but on an older box it can really make a difference. Also, by default, the key bindings all start with ctrl + b instead of ctrl + a —which can be changed, and is changed in my configuration file, which I'll post a link to at the end of the article. Due to the licensing, it is also included by default in BSD systems, for those who are interested in that kind of stuff. Also, a few key bindings are different, but the man page of tmux is extremely clear, and it offers a complete list of commands that can be accessed with the following command:

```
tmux list-commands
```



Above right is a screen-shot of tmux running (as you can probably tell, I actually use tmux on my netbook, as opposed to Byobu running in a Virtual Machine):

Configuration file (~/.tmux.conf): http://lswest.pastebin.com/fa64f955

List of commands for tmux: http://lswest.pastebin.com/f7d0cad21

I figured that I should include a few other choices in this series, since Linux is all about choice, and the freedom to use what you want.

Personally, I find tmux a bit easier to use and understand, but GNU Screen is the more widely-known program, which is why it was covered in-depth and tmux wasn't. However, the man page covers all the usual info, as well as keyboard shortcuts, configuration options, etc. I highly recommend looking at the man page before posing questions on how to do something, since most of what you'll need to know is in there (and pretty easy to find). Both of the programs are in the Universe repository in Ubuntu 9.10. As always, any article suggestions or questions can be sent to me at lswest34@gmail and I will do my best to answer the questions, and fulfil the requests.

Further Reading:

Byobu: https://edge.launchpad.net/byobu
and:
http://blog.dustinkirkland.com/search/label/Byobu

tmux: http://www.openbsd.org/cgi-bin/man.cgi?query=tmux&sektion=1 (online man page)
and:
http://tmux.sourceforge.net/ (homepage).

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

COMMAND & CONQUER

Written by Lucas Westermann

After finishing my Screen segments, I realized that it may be interesting for my readers to see what other things the bash or zsh shells can do. Therefore, I'll be covering the various shells that exist for Linux (along with a short description), and an in-depth section on customizing/configuring *Z Shell* (Zsh) and *Bourne Again Shell* (Bash), since those are the two shells I've seen most widely used, and the two shells I'm most comfortable with. It also leaves room for you, my readers, to play around with a few new shells on your own and to learn for yourselves what they can do.

The following shells are available:

Bourne Shell (sh) – Original Unix shell. Offered no notable features outside of what one would expect from a shell.

Almquist Shell (ash) – BSD-Licensed re-write of Bourne Shell. Similar feature set as above.

Bourne-Again Shell (bash) – Standard shell used in Linux distributions. Offers a superset of Bourne Shell functionality. Written as part of the GNU Project.

Debian Almquist Shell (dash) – Modern replacement of the Almquist shell for Debian-based Linux distributions.

Korn Shell (ksh) – A shell written by David Korn.

Z Shell (zsh) – Considered the most “complete” shell available (offers the most features). Could be described as a superset of sh, ash, bash, csh, ksh, and tcsh (TENEX C shell).

C Shell (csh) – A shell written by Bill Joy, and is special in the sense that its syntax is similar to the c programming language.

This is by no means an exhaustive list of shells, but they're the ones I believe are still actively developed/used among the community. You may wonder why anyone would bother to change their default shell. The main reason why I prefer Z Shell over Bash is simply because it offers certain features I prefer (a better tab auto-completion than Bash, easier colour syntax for prompts, a right-hand prompt as well as a left-hand one, etc.). As with so many things to do with Linux, it ultimately boils down to choice. Maybe you're a skilled c programmer, and prefer to have a shell that has a similar syntax, and have therefore opted for the C Shell. I won't say one is better than another, simply because it has a feature or two that others don't, and vice versa. I will, however, only cover how to configure Bourne-Again Shell and Z Shell in this article, because I have experience with these, and because they seem to be the most widely used shells out there.

The first thing I need to cover is how to install and test a new shell, and how to change the default shell. To install, you just need to apt-get whatever shell you'd like to try out. Once it's installed, checking the manpage will give you the location of the configuration file. Also, since you'll most likely want to see the default prompt, you can switch shells by simply running the binary name for the shell. (sh, ash, bash, zsh, csh, ksh, and so forth). It will drop you into that shell without changing the default. I always recommend viewing the default configuration file, and making a local version for customization, in case something goes awry. I also recommend testing a new prompt via the command line, before committing it to the

configuration file. This is as simple as exporting the PS1 via the command-line. Just keep tweaking it until you're happy, and then copy the end product into the configuration page. Once you're happy with the configuration, and certain that there are no major problems with the configuration file, you are ready to change the default shell (as long as you want to). To do so, run the following command:

```
sudo chsh -s /path/to/binary $USER
```

You need to replace "/path/to/binary" with the path to the shell (i.e. /bin/bash), and $USER with your actual username/username of the account whose shell you'd like to change. In case you're not sure what shells you have available (and recognized by the system), you can view them using:

```
chsh -l
```

This may not show all the shells, since it merely prints those listed in /etc/shells, but most packages should update that file accordingly.

You may be wondering what exactly you can configure in a shell, and why you would bother. A few things that I will cover are: exporting environment variables for use in window managers (openbox instead of Gnome, for example), aliasing commands for easier use, customizing the prompt itself, and adding functions to the shell.

Configuring Z Shell

Follow this link: http://lswest.pastebin.com/WBm22Wig to view a full .zshrc file. A note on the bindkeys: this is due to the fact that Zsh lacks support for home/end/page up/page down, and displays only escape sequences when pressed, unless you define bindkeys such as I have. You may need to find the right escape sequence for the key. If you have vi emulation enabled (which is what I use, and is done with bindkeys -v), you can see the escape sequence by hitting ctrl + v, and then the key you wish to have the escape sequence for. Chances are the ones I use will work for most others though, so you can always try them first.

I'll be referring to the file for examples (using the line numbers as viewed on pastebin).

The first thing I'd like to cover is how to export variables, since it's a useful thing to know, and pretty easy to do. To export a variable, all you need to do is use the syntax:

```
export $VARIABLE="value"
```

As you can see in my configuration file on lines: 11, 15, 117, 82, 116, 131 and 132. You need to, of course, replace "$VARIABLE" with the actual variable (such as DE, or OOO_FORCE_DESKTOP), and "value" with the actual value. You can put quotes around the value, or leave them away if it's only one word (as you can see in the file). The last two exports in my configuration file are extremely useful when using openbox, since it sets the Desktop Environment to Gnome for xdg-open (the program that auto-selects the default application for filetypes). In other words "xdg-open" and a file path will open nautilus when set to Gnome, thunar when set to xfce, and konqueror when set to kde. The OOO_FORCE_DESKTOP export also sets OpenOffice to use the gtk theme, instead of using the QT theme, which is the default option unless the DE is gnome.

The next useful trick is to add aliases to your configuration file, so that you can use extended arguments for a command easily. This can be seen particularly well on line 84, since I use the alias trayer (thereby ignoring the actual binary file of that name), and use it to launch trayer with a specific set of arguments. If, however, you discover you want to use the original trayer binary without the alias, you can override the alias temporarily by using the following:

```
\<alias name>
```

So, in this case it would read \trayer. It's similar to how you escape certain characters so that a shell sees it only as text. An extremely useful alias I use on all my *nix boxes, is the alias for ls (on line 64), since it gives me a much more detailed listing of files within the folder.

Now we come to the most widely use customization of Shells. The prompt itself. The prompt I prefer to use in Z Shell is in the following format:

```
┌[lswest@laptop:~] - [14:24:29]
└>
```

It's a double-line prompt, giving me more room to write commands, and it offers me the current user logged in, the hostname, and the current working directory (after the colon). For those wondering how I manage a double-line prompt, the magic happens here: "$'\n'", where I break the main section of the prompt, and add in an escape sequence for a new line, and then continue the prompt. This doesn't work (last time I tried), with just the escape sequence in double quotes. Also, a right-handed prompt can be added using the RPROMPT variable (I have it commented out in my configuration file, but it is still there).

I update the configuration file regularly, and the copy that's on pastebin at the moment is an iteration or two behind, but the major change is that my current prompt also offers me the time I ran a command. If you look at the configuration file, you can see that there are actually two prompts listed in an if statement. Basically it checks to see if I'm using screen, and if so, it displays the current screen window value before my username, making it easy to keep track of where I am. A complete list of escape sequences for Zsh is available on the man page for zshmisc, but here is a list of ones I often use (taken from http://www.acm.uiuc.edu/workshops/zsh/prompt/escapes.html):

Literal characters

%% - A %
%) - A)

Directories

%d - The current directory ($PWD)

%~ - $PWD, but will do two types of substitutions. If a named dir 'X' is a prefix of the current directory, then ~X is displayed. If the current directory is your home directory, $HOME, just ~ is displayed.

%c - Trailing component of $PWD. If you want n tailing componenets, put an interger 'n' after the %.

%C - Just like %c and %. except that ~'s are never displayed in place of directory names.

Hostname info

%M - The full machine hostname.

%m - The hostname up to the first . (dot). An integer may follow the % to specify how many components of the hostname are desired.

Current time info

%t - Current time of day, in 12-hour, am/pm format.

%T - Current time of day, in 24-hour format.

%* - Current time of day in 24-hour format, with seconds.

Current date info

%w - The date in day-dd format.

%W - The date in mm/dd/yy format.

%D - The date in yy-mm-dd format.

%D{string} - string is formatted using the strftime function. See strftime(3) for more details. Three additional codes are available: %f prints the day of the month, like %e but without any preceding space if the day is a single digit, and %K/%L correspond to %k/%l for the hour of the day (24/12 hour clock) in the same way.

Miscellaneous info

%h - Current history event number.

%n - Equivalent to $USERNAME.

%l - The line (tty) the user is logged in on.

%# - A `#' if the shell is running with privileges, a `%' if not. The definition of privileged, for these purposes, is that either the effective user ID is zero, or, if POSIX.1e capabilities are supported, that at least one capability is raised in either the Effective or Inheritable capability vectors.

Zsh offers a few default colours that can be accessed with names such as red, cyan, etc. But it also accepts the usual \e[0;31m style formatting (as discussed in the Bash section).

Last, but possibly most useful, is the ability to add functions to a shell. The set up is exactly the same as for Bash scripts. The method is to define a function with “function name() { #code }”. I have a few functions in my Zshrc file, such as: m4a, flvmp3, google, etc. As you can see, you can also define a function without using the descriptor “function”, but it makes it more readable. My configuration file is by no means a good example of an organized file. Ideally, I'd have kept all exports together, all functions together, all aliases, and so forth. Instead, I add things to the file as I think of them, leaving me with a bit of a mess. I'll probably get around to tidying it up eventually (seems to happen about once a year)

Configuring Bourne-Again Shell

Exporting and aliasing are exactly the same for Bash shells as for Zsh shells, so to see how to do that, please read the first two explanations of the Configuring Z Shell section. The only sections in the .zshrc file I have a link to that aren't relevant to Bash prompts are the bindkeys section, and the PROMPT sections.

As for customizing the prompt in Bash, it's similar to Zsh, except for the list of escape sequences you can use, and how the variable behaves when it comes to double-lines. The following is a list of escape sequences for bash (taken from: http://www.cyberciti.biz/tips/howto-linux-unix-bash-shell-setup-prompt.html):

\a : an ASCII bell character (07)

\d : the date in "Weekday Month Date" format (e.g., "Tue May 26")

\D{format} : the format is passed to strftime(3) and the result is inserted into the prompt string; an empty format results in a locale-specific time representation. The braces are required

\e : an ASCII escape character (033)

\h : the hostname up to the first '.'

\H : the hostname

\j : the number of jobs currently managed by the shell

\l : the basename of the shell’s terminal device name

\n : newline

\r : carriage return

\s : the name of the shell, the basename of $0 (the portion following the final slash)

\t : the current time in 24-hour HH:MM:SS format

\T : the current time in 12-hour HH:MM:SS format

\@ : the current time in 12-hour am/pm format

\A : the current time in 24-hour HH:MM format

\u : the username of the current user

\v : the version of bash (e.g., 2.00)

\V : the release of bash, version + patch level (e.g., 2.00.0)

\w : the current working directory, with $HOME abbreviated with a tilde

\W : the basename of the current working directory, with $HOME abbreviated with a tilde

\! : the history number of this command

\# : the command number of this command

\$: if the effective UID is 0, a #, otherwise a $

\nnn : the character corresponding to the octal number nnn

\\ : a backslash

\[: begin a sequence of non-printing characters, which could be used to embed a terminal control sequence into the prompt

\] : end a sequence of non-printing characters

To make a multi-line prompt in Bash, all you need to do is place an escape sequence newline character ("\n") where you'd like the line to break. You can also customize PS2 and onwards, which appear when you start a multi-line command (e.g. A for loop). As for colors, the escape sequences are available (from: http://wiki.archlinux.org/index.php/Color_Bash_Prompt#List_of_colors_for_prompt_and_Bash):

You can, of course, place the colors inside variables and use that within the configuration file. The bash version of my Zsh prompt (without the timestamp) would be as follows:

```
export PS1="\[\e[0;37m\]┌─[\[\e[0;32m\]\u\[\e[0;36m\]@\[\e[0;32m\]\h\[\e[0;37m\]:\[\e[0;33m\]\w\[\e[0;37m\]]\[\e[0;36m\]\n\[\e[0;37m\]└─>\[\e[0m\] "
```

I apologize for not having an example bash file to display, but the configuration syntax for both Zsh and Bash are similar, so that should be a decent example for both. If any reader would like, I would be happy to display your customized .bashrc files, along with a textual representation of the prompt, at the beginning of each month's Command & Conquer. If you're interested, just send me an email at lswest34@gmail.com with your .bashrc, and a textual representation of the prompt, or an actual screenshot. Also, please refer to Command & Conquer in the subject line, so that I put it higher on my priority list. For any users who use urxvt/define custom prompt colors in your .Xdefaults, please share the relevant section as well (if you send a screenshot).

Any questions, suggestions, or problems can be emailed to me at lswest34@gmail.com, and any further ideas for segments are always welcome in my inbox! I wish you all a fun time configuring your prompts, and I'm curious to see what your results will be! I hope I've done a good job at explaining this, and I'll gladly continue on with further customizations to the terminal, if there's enough interest. And, as always, there is plenty more information regarding this in the Further Reading section.

Further Reading

http://en.wikipedia.org/wiki/Alias_%28command%29 – Info on the Alias command

http://www.cyberciti.biz/tips/howto-linux-unix-bash-shell-setup-prompt.html – Bash prompt customization how-to

http://markelikalderon.com/2007/11/24/full-paths-and-the-multiline-shell-prompt/ - Multi-line prompts.

http://wiki.archlinux.org/index.php/Color_Bash_Prompt#List_of_colors_for_prompt_and_Bash – colorizing bash prompts

http://docs.cs.byu.edu/linux/advanced/zsh.html – how to configure Zsh prompts.

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

COMMAND & CONQUER

Written by Lucas Westermann

Following my article last month on customizing prompts and shells, I thought it might be nice to explain how you can customize the colours used by your terminal. It's also a good introduction to your .Xdefaults file, which offers quite a bit of control over user-specific settings. It can be used to set the mouse cursor, urxvt-specific settings, configuring terminal settings, setting DPIs, anti-aliasing, and other X Font preferences, and theming xscreensaver, among other things. There are plenty of examples of terminal colour schemes at Aaron Griffin's website (he's the lead developer of ArchLinux): http://phraktured.net/terminal-colors/. Today, I'll be covering the process by which you can design your own terminal colour scheme. This consists of a few basic steps:

- Understanding the syntax of the .Xdefaults file regarding colours
- Finding hex values of colours, and finding complementary colours
- Some way to display the resulting colour scheme as a test

I'll be focusing on the methods I'm comfortable with, but it's by no means the only way to create these colour schemes. The first thing to do is to check the current colour scheme to see what you have to work with, if anything. To do so, I highly recommend Daniel Crisman's colourscheme.sh (see the first link of the Further Reading section, at the very end of that webpage). To use it, just copy it into a file, and chmod +x the file. For example:

```
vim colours
```

(see footnote [1] on the next page for more information on the above)

<after pasting in the script and exiting vim>

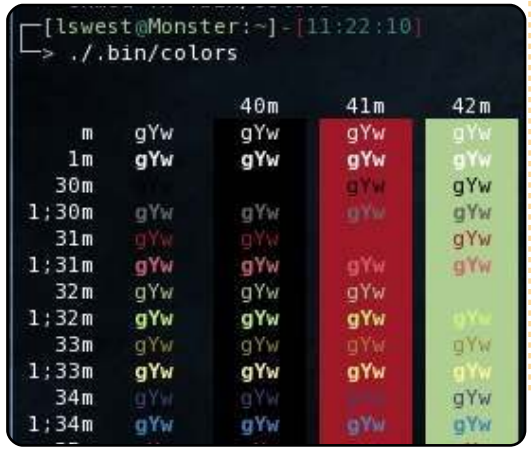
```
sudo chmod +x colours
```

Then to run it all you need to do is the following:

```
./colours
```

It will display something like the one shown below.

The second step for me is always to pick a base colour, which impacts what other colours I can choose, as we want complementary colours. You can always choose a basic colour, green, for example. Then you can open a colour palette, either gcolor2 if you want a stand-alone colour chooser, or you can use GIMP to mix new colours. What's



important to note is the hex value of the colour you decide upon. Once you've decided upon your base colour, it's time to find complementary colours. If you share my problem of being unable to think of complementary colours off the top of your head, you can use the search function on ColourLovers: http://www.colourlovers.com/ and give it the hex value for the colour to find palettes of matching colours. Once you've decided upon your set of 16 colours (and the background/foreground colours, for a total of 18 hex values), it's time to write it into your .Xdefaults. The format to do so for all terminals is this:

```
!---- Terminal Colours
*background:   #000000
*foreground:   #ffffff
*color0:       #000000
*color1:       #9e1828
*color2:       #aece92
*color3:       #968a38
*color4:       #414171
*color5:       #963c59
*color6:       #418179
*color7:       #bebebe
```

```
*color8:        #666666
*color9:        #cf6171
*color10:       #c5f779
*color11:       #fff796
*color12:       #4186be
*color13:       #cf9ebe
*color14:       #71bebe
*color15:       #ffffff
```

The top line is the format for comments in the .Xdefaults file. You can also specify a colour scheme for a specific terminal by appending the name of the binary in front of the asterisk. For example (the same scheme, just for urxvt-only):

```
urxvt*background: [70]#000000
urxvt*foreground:  #ffffff
urxvt*color0:      #000000
urxvt*color1:      #9e1828
urxvt*color2:      #aece92
urxvt*color3:      #968a38
urxvt*color4:      #414171
urxvt*color5:      #963c59
urxvt*color6:      #418179
urxvt*color7:      #bebebe
urxvt*color8:      #666666
urxvt*color9:      #cf6171
urxvt*color10:     #c5f779
urxvt*color11:     #fff796
urxvt*color12:     #4186be
urxvt*color13:     #cf9ebe
urxvt*color14:     #71bebe
urxvt*color15:     #ffffff
```

In this version, the value in square brackets before the hex value for the background is an opacity setting (so 70% opaque, or 30% transparent). This is only possible for terminals that support transparency, and on systems where you're running a compositing manager (xcompmgr, cairo-compmgr, compiz, mutter, and so forth), in order to render the true transparency.

Once you've added your preferred colours into the .Xdefaults file, you'll probably want to see what it looks like without having to log out and back in. Luckily, you can do so with just a little bit of command-line magic. Entering the command:

```
xrdb -merge ~/.Xdefaults
```

will force xrdb (X Resource Database Manager) to re-load the settings within .Xdefaults, and thereby overwrite any current settings. Re-running the colours script will also give you an overview of your new colour scheme.

This is essentially all there is to it. It may take a bit of trial and error to find a setup that you really like, but that's part of the fun. I also realize that the two links I have for more info about .Xdefaults are for Arch, but I wasn't able to find any similar pages for Ubuntu. Besides, the instructions will be almost identical for either system. The examples I used above are also the colour scheme I use, which I believe I based on someone's .Xdefaults that I found online years ago. There's probably not much left of the original, but I felt I should note that it's not all my work. As usual, any questions, suggestions, or general feedback, can be directed to lswest34@gmail.com. I also ask that anyone who does send an email write "FCM – C&C" in the subject header, so that I don't overlook it. I'd also love to see the results of your .Xdefaults, and I'll gladly feature a few in the next C&C, if you send me a screenshot and the corresponding .Xdefault settings.

Further Reading:

Daniel Crisman's colourscheme.sh from: http://tldp.org/HOWTO/Bash-Prompt-HOWTO/x329.html

Arch Wiki page on .Xdefaults, with a few links and examples: http://wiki.archlinux.org/index.php/Xdefaults

A thread on the Arch Forums, with terminal colour schemes: http://bbs.archlinux.org/viewtopic.php?id=51818&p=1

ColourLovers (for colour palettes): http://www.colourlovers.com/

[1] In order to paste in Vim without it adding spaces due to auto-indenting, run ":set paste", hit "i" to insert in paste mode, and paste your script into the file. To disable paste mode, give in ":set nopaste". Both commands are run in the usual vim way, and without quotes.

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

COMMAND & CONQUER

Written by Lucas Westermann

If you own a laptop, you've probably often checked your emails, read up on news, or done something online - while travelling. If you're like me, you'll cringe inwardly whenever you do so, knowing full well that there may be someone else connected to the free/public hotspot running a packet sniffer and hoping for a few passwords or banking data. I have set a firm “no banking on-the-road” rule for myself and my family, but I'm also worried about our other passwords and private data. This is where SSH port-forwarding can be extremely useful. It uses SSL to encrypt all the data it sends; it uses the public wifi for nothing more than a link to whatever your SSH server happens to be (home server, home PC, work server, virtual private server, etc.); and it protects your traffic both to and from your computer from most packet sniffers and man-in-the-middle attacks.

The first thing you'll need to do is set up an SSH server on your PC, or sign up for a shell account/virtual private server somewhere, if you don't already have it set up. If you already have access to an SSH account, skip ahead to step 7.

First step:

Install OpenSSH server on your Ubuntu system:

```
sudo apt-get install openssh-server openssh-client
```

Second step:

Create a backup of /etc/ssh/sshd_config

```
cp /etc/ssh/sshd_config ~
```

Third step:

Modify the sshd_config file. You can read up on possible options using the man page:

```
man sshd_config
```

The basic configuration should simply be to disable root login, and to specify users who can log in via SSH. To do this, open /etc/ssh/sshd_config:

```
sudo vim /etc/ssh/sshd_config
```

Once it's open, change the line “PermitRootLogin yes” to “PermitRootLogin no” and add the line “AllowUsers user1,user2,user3” somewhere in the file. You need to, of course, change “user1” to the actual username, while user2 and user3 should be replaced with any other accounts who are permitted to connect to the server. For example my line would read:

```
AllowUsers lswest
```

Since I have only one permitted account and user, that is what I would enter.

Fourth step:

Restart the SSH server after you've completed your changes to the configuration file:

```
sudo /etc/init.d/ssh restart
```

Fifth step:

Create SSH keys (if you want to). Since this step is optional, I won't cover the exact commands. If you wish to generate keys, follow the instructions in the wiki (see the link in the Further Reading section).

Sixth step:

Configure your server/PC to allow internet access, and configure dyndns. I have never configured dynamic DNS for any computer, so I will leave those instructions to the wiki article (second link of my Further Reading section). A short-term solution would be to create a cron job to run the following command

```
curl http://www.whatismyip.org
```

and to redirect the output into your Dropbox or Ubuntu One folder, so you can check it from other computers. i.e.:

```
curl http://www.whatismyip.org > ~/Dropbox/IP.txt
```

I explained cron jobs in Issue 24, in case you want a deeper understanding of it. If not, the following steps will set up a cron job to do the above command every day at 12 p.m. (noon):

```
crontab -e $USER
```

Add the following line to the file:

```
00 12 * * * curl http://www.whatismyip.org > ~/Dropbox/IP.txt
```

Once this step is complete, you're ready to begin.

Seventh step:

You'll need the following information:

IP address of your server, username and password, or a username and a key file (from step 5).

To connect and forward all traffic over port 8080 to your SSH connection, run the following command:

```
ssh -D 8080 lswest@localhost
```

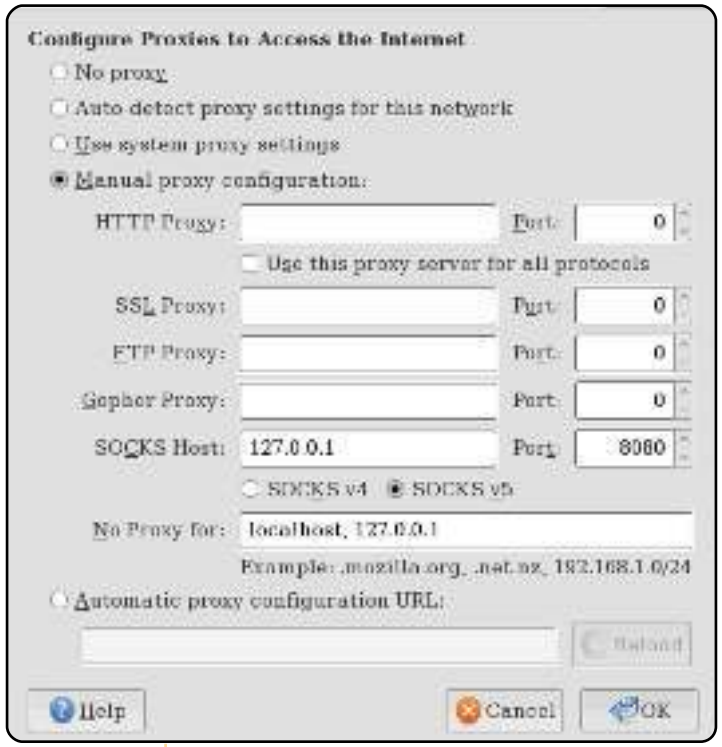
You'll then be asked to accept the rsa id from the server, to which you answer "yes", and then you will need to supply your password (if you don't have a key file generated). Once you've entered your password, you'll be greeted with the normal SSH prompt. You'll need to leave the connection active/window open (unless you run it in screen or tmux - then you can simply detach the session).

Eighth step:

Configure the SOCKS proxy in Firefox. This is simply done by going to Edit > Preferences > Advanced tab > Network sub-tab > Connection Settings. Once that opens, configure the settings shown in the image above right.

Ninth step:

Disconnecting from the SOCKS proxy. Simply change the configuration to "Auto-Detect proxy settings for this network", or to "No Proxy", and disconnect from the SSH server.



Hopefully, this article is useful for those who are very security-conscious, and maybe it will increase awareness for some everyday security problems that many people fail to realize. Naturally, there are more uses for this, and the proxy can be used in Evolution/Thunderbird, as well as many other programs, but I felt that this was the most universal/more useful scenario in which it would be used. As always, I'm happy to answer specific questions, or take requests for articles by email. Send any feedback, questions, and so forth, to lswest34@gmail.com with the words "Command & Conquer" (or just C&C) in the subject line.

Further Reading:

https://help.ubuntu.com/9.10/serverguide/C/openssh-server.html – Wiki article on installing OpenSSH

https://help.ubuntu.com/community/DynamicDNS – Wiki article on installing and configuring dynamic dns

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

COMMAND & CONQUER

Written by Lucas Westermann

Before I begin the actual article, I'd like to mention an email I received from a reader. Alexander was kind enough to point out that there is a GUI tray program called “gstm” that does ssh port forwarding. For those of my readers who prefer GUI alternatives where possible, there you go. It's available in the universe repository. Also, a reader (who, alas, did not share his/her name with me - but you know who you are!) pointed out that in Step 7 of my FCM#37 C&C article I failed to point out that you need to substitute "localhost" with the IP of your server. (The command was "ssh -D 8080 lswest@localhost", where lswest@localhost had to be substituted with your_username@IP_ADDRESS_SERVER). Sorry for any confusion that may have resulted.

This month, I spent quite some time re-writing a few patches for DWM (Dynamic Window Manager) so that they would work with the pango patch, which adds xft font support to DWM's statusbar. In doing so, I learned quite a bit about diff, and have decided to share what I learned with you, my readers. If you're asking yourselves “why should I know how to use diff, since I don't use DWM nor generate patches?”, the answer is simply because diff can be applied to so many situations. Imagine you are writing a script, and you want to add to the script, but require the old version for a different computer - instead of creating and backing up two separate scripts, you can write the script, create a copy of it, make changes to the copy, generate a .diff file, and back up the original script and the .diff file, and save yourself some work in the future. Or, if you're helping a friend, and you can't simply send them the file you need to correct, you can send them a .diff to make the changes.

There are probably many other uses (adjusting configuration files, and so forth) that I haven't thought of yet.

Diff is installed, by default, in most distributions. If it's not present in Ubuntu, just install it with:

```
sudo apt-get install diff
```

Once it's installed, you're pretty much all set. In order to generate a .diff file, you need to have two files you want to analyse. One will be the “original” (I will refer to it as such from now on), and the other will be the “updated” file.

For simplicity, let's say I have a file that contains the following:

VirtualBox How-To (set up, install, and configure a virtual machine).

Virtualization series: Each month write an article for a distribution with screenshots and so forth

C&C:
Cover useful stuff to do with curl, wget, and so forth? And diff?

And I want to change this to:

VirtualBox How-To (set up, install, and configure a virtual machine).

Virtualization series: Each month write an article for a distribution with screenshots and so forth

C&C:
Diff (wget and curl next month)

I'd make the changes I want in the “updated” file. I usually add a “-patched” suffix at the end of the filename. Once the changes are made, and I want to generate a diff, I will type the following into my terminal of choice:

```
diff -up original updated > articles\list\update.diff
```

Replace “original” and “updated” with the actual file names and paths. If you don't want the diff file to be created in the current working directory, append a path to the filename on the other side of the “>”. The “>” tells the shell to redirect all output into whatever you pass after the symbol - in this case, the .diff file. If you want to apply the changes to another copy of the original file (on a different computer, for example), you would need to run one of the two commands in the folder containing the file you wish to patch (they do the same thing):

```
patch -p1 < /path/to/.diff
```

```
patch -Np1 -i /path/to/.diff
```

Where, of course, you exchange the “/path/to/.diff” with the actual path.

I realize that my example isn't really a case where you would decide to use a patch/.diff file to make changes, but I chose it for the sake of simplicity. Another scenario where diff is useful: if you have two files (in my case, it's usually configuration files), and you don't know if they've been changed, and if they have, what changes you've made. To check this, you can simply run the command:

```
diff /path/to/first/file
/path/to/second/file
```

Be sure to actually replace the paths. The output should look something like this:

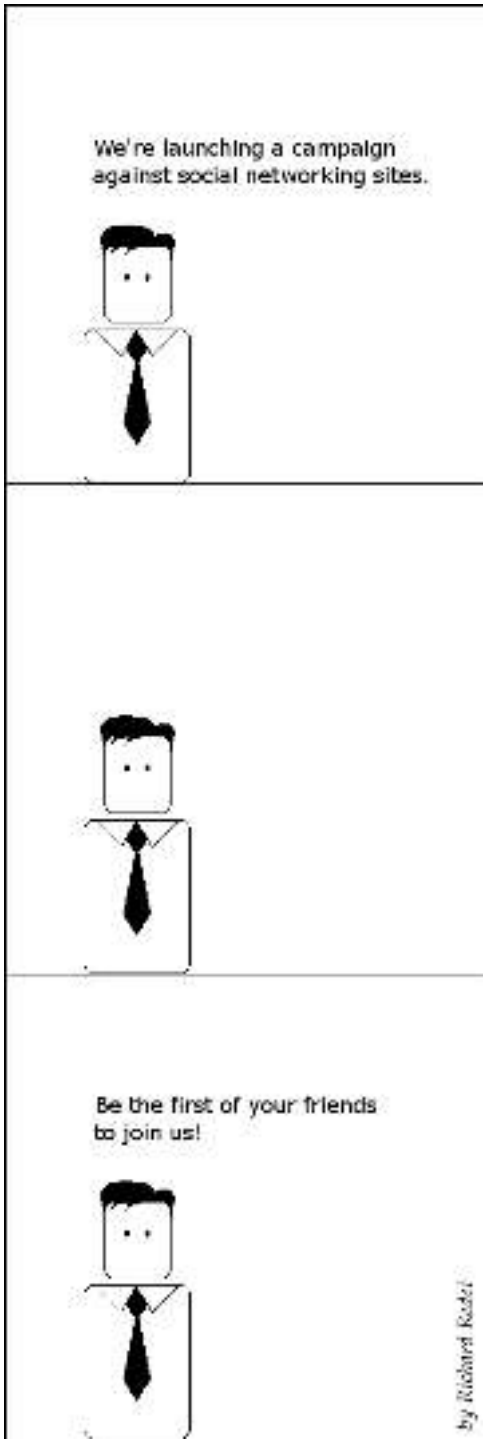
```
5c5
< - Cover useful stuff to do
with curl, wget, and so
forth? And diff?
---
> - Diff (wget and curl next
month)
```

I'll explain the above line by line. The “5c5” is (I believe) a comparison of lines within the first and second files. I'm not certain of this, but it seems to be the case. The next line displays a line that was removed (the “<” denotes deleted), and the line below displays the line that was added (therefore replacing the original line), which can be seen with the “>” symbol.

As you can see, this is a very useful tool for figuring out the differences between two files, especially if they're rather long. You can pipe the output into “more” or “less” for easier reading, or redirect it into a text file. The format will be the same, as long as you don't append any arguments to the diff command.

Hopefully this introduction to diff has helped you realize a scenario in which you could make use of it, and will hopefully make life easier for anyone who decides to use it. As always, any questions or comments can be emailed to me at lswest34@gmail.com. Be sure to include “C&C” or “FCM” in the subject line, so that I reply quickly (and can organize my emails easily!).

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.



COMMAND & CONQUER

Written by Lucas Westermann

Before I start this month's article, I have a few corrections to make for my last article. Reader *Grofaty* pointed out that patch isn't installed by default in Ubuntu, and also wanted to make me aware of *vimdiff* (vim-style diff interface).

Moving on to this month, I thought it would be fun to cover two command-line tools for downloading websites/web-pages, namely, ***cURL*** and ***Wget***. You may be thinking "but I have Firefox, why would I need cURL or Wget?". The main reasons I use them nowadays is when I need to make an offline copy of a website (not just one web page) or to download a web page/file when behind a firewall that blocks that website. There are plenty of other uses for them, such as archiving your own website, parsing websites within scripts, quickly downloading something without opening Firefox, or downloading all files of a type (useful for students who have web-portals with lots of research PDFs). For those wondering what the difference is between cURL and Wget, it's a subtle, but important, difference. cURL pulls down the HTML code and prints it to STDOUT (i.e. returns it as the output of the command), while Wget downloads the .html files. This means that cURL is ideal for parsing certain streams (if you're writing a Google search script, for example), while Wget is useful for making a full archive of a website.

Here are a few examples for cURL:

```
curl -L www.w3schools.com/css
```

This command tells cURL to follow any redirects on the CSS page of w3schools.com (specifically, Location: pointers). On this site, it should follow the "Next Chapter" links automatically.

```
curl -u name:password
https://mail.google.com/gmail
/feed/atom
```

This command gives cURL a user-name and password to allow it to authenticate on the website (in this case, Gmail's atom feed), thereby gaining access to the site without you having to open Firefox.

These examples could be used in a script that accesses Google, searches for something, and returns the results/HTML of the top result. It can also log you into your Google mail account.

And Wget examples:

```
wget -r -l3
http://w3schools.com/css/
```

This command sends Wget to w3schools.com, and follows the links recursively for 3 levels (i.e. 3 Homepage --> CSS Intro --> CSS Syntax). It should be

```
[lswest@Monster:~]-[15:08:47]
> curl -L http://192.168.2.103/test
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
   <title>Test Page</title>
<link href="style.css" rel="stylesheet" type="text/css">
</head>

<body>
<div id="Content">
<div id="Header">
Test Page
</div>
<div>
<ul id="list-nav">
<li><a href="#">Home</a></li>
<li><a href="#">About Us</a></li>
<li><a href="#">Services</a></li>
```

noted that using a recursive website travel in Wget can put a large strain on a webserver, so it should always be used with the levels argument, in order to minimize website traffic.

```
wget -c -U Mozilla www.website.com
```

I didn't include an actual link in this example, because I couldn't think of a site that applied off the top of my head. However, this Wget command pretends to be Mozilla's browser (by altering the user agent) in order to get around restrictions on download managers. The -c option tells Wget to store any partially downloaded files so that the download can be resumed.

```
wget -r -l1 -A.pdf --no-parent http://url-to-webpage-with-pdfs/
```

This command tells Wget to recursively follow a website for one level, and download any pdf files it finds. The --no-parent option tells Wget to never follow a link up to the parent directory (i.e. www.test.com from www.test.com/something), which is useful for avoiding strain on the server as well. The -A option accepts a comma-separated list of file extensions, or wildcards/patterns. In order to reject any files of a certain type, use -R instead of -A.

```
[lswest@Monster:~]-[15:07:18]
-> wget -r -l2 http://localhost/current
--2010-07-17 15:07:47--  http://localhost/current
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanen
Location: http://localhost/current/ [following]
--2010-07-17 15:07:47--  http://localhost/current/
Reusing existing connection to localhost:80.
HTTP request sent, awaiting response... 200 OK
Length: 1061 (1.0K) [text/html]
Saving to: "localhost/current/index.html"

100%[=============================================

2010-07-17 15:07:47 (187 MB/s) - "localhost/current/index.
```

Hopefully this (admittedly short) article has made the power of Wget and cURL clear, and, as always, plenty more information can be found in their respective manpages. For anyone who has requests for command-line tools that I should cover, you can send me an email at lswest34@gmail.com with "FCM C&C" or "Command & Conquer" in the subject line. If I don't already know the tool, I'll figure it out before I write the article. For anyone who comes up with a use for cURL or Wget that they find quite clever, feel free to share it with me in an email as well.

Further Reading:

http://curl.haxx.se/docs/httpscripting.html – Great cURL tutorial/manpage (some examples were borrowed from here).

http://linuxtuts.blogspot.com/2008/03/tutorials-on-wget.html – Great tutorial on Wget (some examples were borrowed from here).

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

COMMAND & CONQUER

Written by Lucas Westermann

Oops!

Reader ***Inkimar*** made me aware that, when discussing wget's usefulness as a PDF-grabber, it was unclear if I was referring to both cURL and wget. CURL may be able to fulfil the same function, but it would output the binary PDF files to STDOUT, resulting in gibberish. I urge anyone who is confused about what program to use to follow this simple guideline: cURL for webpages/online files you want to parse, and wget to download any web-based files you want to store.

This month, I'm going to leave out any extremely new concepts, and aim to consolidate some of the programs and ideas I explained in previous articles. Specifically, how most of my articles can help you when confronted with a tty screen, or when on someone else's Linux installation where the only programs you can rely on are command-line programs. My intention is two-fold. Firstly, I want my readers to feel comfortable with a command-line system, so that, if X dies, they can keep on working to fix it, instead of re-installing from scratch, or following instructions blindly. Secondly, I will be covering a few distributions (and Unix systems) that install nothing more than a command-line interface, leaving the rest up to the user. This way, I can focus on the installation process. This month, I will cover researching problems, and next month I will cover installation-focused programs (fdisk, mkfs, and so forth), in order to coincide with the first Unix virtual machine we'll be creating.

I assume many of you are familiar with the process of solving problems of a technological nature. Generally, it follows these steps: make a guess at what is wrong (or find error logs), Google the error message or problem, see if any results solve the problem for you, and, if you haven't solved the issue, then post on a forum. However, what would you do if you were stuck in a command-line screen without ready access to Firefox, Nautilus, or similar programs? I realize it's not uncommon for there to be multiple computers in a household these days, but I always find it inefficient to use a second computer to diagnose a first. As long as the computer is connected to the Internet, you can do your troubleshooting there. All you need is: ifconfig/iwconfig (and maybe wpa_supplicant), dhclient, cd/ls (or something like midnight commander), vim/nano, and elinks. Sure, midnight commander and elinks aren't generally installed from the get-go (nor is wpa_supplicant I believe), but these are valuable tools to have ready, just in case. Also, they don't take much space at all.

To ensure you're connected to the Internet, you can run a quick ping request:

```
ping -c 3 google.com
```

If you get replies, you're connected, if not, you'll probably need to do some work. I will assume that the network is not connected.

First, you'll need to know how you connect to the Internet. Is it via Ethernet cable, WEP-encrypted wireless, WPA-encrypted wireless, or a password-less wireless network?

If it's an Ethernet-based connection, all you'll need is ifconfig and dhclient. Use the following command to ensure that the Ethernet interface is “up” (enabled):

```
sudo ifconfig
```

If there is an “eth0” interface listed, you're all sorted and just need to run:

```
sudo dhclient eth0
```

The command will request an IP address from your router, after which the Internet should work without a problem. If your interface is not listed, it will be because it's “down” (disabled). To enable it, type:

```
sudo ifconfig eth0 up
```

And then run the same dhclient command as above. If you have multiple Ethernet cards, you can get a list of all possible interfaces with:

```
sudo ifconfig -a
```

This shouldn't be required for most PCs.

If you have a WEP-encrypted wireless (or a wireless without a password), you'll need iwconfig and dhclient. First, make sure you have the passkey/encryption key, and the ESSID ready. Then use the following command:

```
sudo iwconfig $interface
essid $ESSID key $KEY
```

Replacing "$interface" with the interface name (usually eth1 or wlan0, you can check by just running iwconfig without arguments, or ifconfig), $ESSID with the wireless network's name (can be found using "iwlist scan"), and $KEY with the passkey (ASCII password used to connect) or the encryption hex key (actual hex string). If the key is a passkey, you will need to append "s:" to the key. For this example the interface is wlan0, the ESSID is home, and the key is passkey:

```
sudo iwconfig wlan0 essid
home key s:passkey
```

Once you enter the command, you can request an IP using this command:

```
sudo dhclient $interface
```

Make sure to replace "$interface" with the interface name. If it fails, you may need to add extra options to the iwconfig command (channel, etc.) which can be found clearly explained in the man page. Or else you may be trying to connect to a WPA secured network.

To do so, you require wpa_supplicant. First, you need to create the information for wpa_supplicant to process. To do so, you need to run this command:

```
wpa_passphrase $ESSID
$passphrase >
~/passphrase.txt
```

Replacing "$ESSID" with the actual ESSID, and "$passphrase" with the actual passphrase. The file path after the ">" is entirely up to you. It will then create a file in your home folder called passphrase.txt that looks like this:

```
network={

ssid="test"

#psk="testing123"

psk=a9ff0c9d1f2367bccf9959e95
bc08695bf411f82b146c55b9486dd
b17495f39d

}
```

You can then connect to your network with the following command (best run in a second tty screen, since it does not daemonize, i.e. runs continuously):

```
sudo wpa_supplicant -
i$interface-c$file -D$driver
```

There are generally no spaces left between the arguments and the switches, but that's up to you. Be sure to replace "$interface" with your interface name (usually wlan0), "$file" with the path to the configuration file we created in the last step, and "$driver" with the driver for your device (usually wext will work, but other drivers are listed in the man page). Once the command starts running, switch back to a non-occupied tty screen and run:

```
sudo dhclient $interface
```

Be sure to replace "$interface" with the actual interface name.

Once you're connected, give the ping request another shot. If it works, you're all set. You can then cd to /var/log/ or wherever you need to go, and check the logs using cat. Once you've decided on a search term for the Google search, open up elinks with the following command:

```
elinks
```

By default, elinks will ask you straight away for a URL, which will generally be google.com. Once it loads, use the arrow keys to highlight the search box (displayed with underscores), if it isn't already selected. Then, hit enter to allow input, and type your search term, hit enter again to begin the search. Use the arrow keys to select links, and enter to follow them. If you stumble across a file you need to download, simply highlight the link to the file, hit Esc, head to Links, and choose "download link" (or hit "d"). You can check on downloads

by either hitting Esc, going to Tools, and then choosing downloads, or hitting “D” (shift+d). Once you're done searching, you can close elinks with “q” or Esc (to bring up the menu), then choose File and Exit.

Hopefully this quick guide (and it really was quick - there are many more things I could have covered here) will help anyone who is following my virtualization series, and anyone who runs into a tty screen. Next month, I'll be covering the things required for installing an OS via the command-line, and managing partitions from the command-line. If anyone has questions, or requests for a more in-depth explanation of anything covered in this article, they can reach me at: lswest34@gmail.com. Please be sure to place the word “C&C” or “FCM” in the subject line, so I don't overlook it.

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

U^3 (U-Cubed) - 28th August 2010

On 28th August, MadLab is hosting U^3 (U-Cubed) - an Ubuntu and Upstream UnWorkshop day in collaboration with HacMan, ManLUG and Manchester Free Software. The day is inspired by the Ubuntu Global Jam event which is being held on the same weekend.

It's an opportunity to help show users how to get involved in wider distribution work - both in the projects they're using already (maybe Ubuntu, maybe just some specific Free & Open Source applications) but also in the upstream projects, such as Debian, Gnome and others, and is inspired by the Ubuntu Global Jam events being held on the same day.

We're hoping to find support on the day from people experienced in Ubuntu, but also people that are involved in more than just Ubuntu, so we're reaching out to anyone in the North West UK region to see if people are prepared to help out - even if Ubuntu isn't the Linux distribution you normally would use, so, if you're interested and available between 11am and 9pm and can get to Manchester, or even if you can just be around for part of the day, please get in touch with me, or better yet, go to http://u-cubed.eventbrite.com to reserve a ticket.

Because we've got limited space, we can only allocate 60 tickets, and to make it fair for everyone, we're giving people access to these tickets at 1PM on Thursday 12th August (giving people a chance to find out about the event), but if you're able to come along to help out with technical information or guidance, please let me know and I'll make sure that you're invited when the flood gates open!

I really hope you'll be able to make it on the day, and help to make the day great!

Les Pounder
One of the U^3 Organisers

COMMAND & CONQUER

Written by Lucas Westermann

This month, I decided to cover both the necessary tools for CLI-based installers (fdisk, mkfs, and so forth), as well as useful tools for finding files on a hard drive - in case you decide to find configuration files on the newly installed system, or your old system that you'd like to carry over. About 2 years ago, I first installed Arch Linux, and in doing so I learned a lot about the command-line interface. Most of it I've found extremely useful. In order to make my Virtualization series a bit easier for my readers, I've been trying to expand on certain ideas within this series as well, in an attempt to bring it all together.

Fdisk:

I use fdisk primarily for listing my partitions on an installed system, and to do so, simply type:

```
sudo fdisk -l
```

It should result in output similar to that shown above right.

```
Disk /dev/sda: 320.1 GB, 320072933376 bytes
255 heads, 63 sectors/track, 38913 cylinders, total 625142448 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x76692ca8

Device Boot      Start         End      Blocks   Id  System
/dev/sda1            2048    30716279    15357116   1c  Hidden W95 FAT32 (LBA)
/dev/sda2   *    30716280   186996599    78140160    7  HPFS/NTFS
/dev/sda3       186996600   625137344   219070372+   f  W95 Ext'd (LBA)
/dev/sda5       186996663   543109454   178056396    7  HPFS/NTFS
/dev/sda6       543109518   570452084    13671283+  83  Linux
/dev/sda7       570452148   625137344    27342598+  83  Linux
```

As you can see, the top half is dedicated to information regarding the hard drive itself, and the second half is dedicated to partitions and information about them.

If you wanted to edit the partition/hard disk, you could enter the following command:

```
sudo fdisk /dev/sda
```

Make sure to replace "/dev/sda" with whatever disk you actually want to edit. Upon doing so, you'll be greeted with a prompt that reads "Command (m for help):". If you enter "m", you get a list of possible commands.

```
Command action
   a   toggle a bootable flag
   b   edit bsd disklabel
   c   toggle the dos
compatibility flag
   d   delete a partition
   l   list known partition
types
   m   print this menu
   n   add a new partition
   o   create a new empty DOS
partition table
   p   print the partition
table
   q   quit without saving
changes
   s   create a new empty Sun
disklabel
   t   change a partition's
system id
   u   change display/entry
units
   v   verify the partition
table
   w   write table to disk and
exit
   x   extra functionality
(experts only)
```

As you can tell, the instructions are generally pretty straightforward. If you enter a command, you'll be prompted for more information as you progress. Please note that when creating a partition ("n" command), you'll be prompted about what cylinder to start from, where the default is generally what you want (as long

as you don't want to leave free space between partitions). Also, it will ask you where you want the partition to end. You can enter a cylinder, or "+1024M" for 1GB later, and so forth. It also accepts bytes and kilobytes as input, though megabytes should be what most people will need. Once you create your partitions, use "a" to make the primary partition bootable, and "t" to change the partition's format ("system id"). When choosing the partition type, it will ask for a hex code for the format, and not the name. A list of the codes can be pulled up with "L". An example is below:

```
Hex code (type L to list codes): 82
```

As you can see, that is the prompt where you can enter "L", and 82 is the hex code for Linux swap. Fdisk is also capable of giving you information about possible issues with current setups, and these are generally clear and, at the very least, give you enough information to easily be able to google for a solution. Once you get comfortable with fdisk, you can also do everything you'll need to do via command-line arguments, instead of having to go through one command at a time.

Mkfs:

You may ask yourself why you'd need "make filesystem" (mkfs) when you can just use fdisk for it all. It's really quite simple – most people don't want to use fdisk for everything since it can be more complicated than necessary. If the partition is created, and you simply want to reformat it, mkfs is probably the tool better suited to the job. There are a few ways to use mkfs, listed below.

```
mkfs /dev/sdXY

mkfs.ext2 /dev/sdXY

mkfs -t ext2 /dev/sdXY
```

Each of these commands will format "sdXY" (replace X with the drive letter, and Y with the partition number) as ext2 (the default format for mkfs). The first command works only for ext2 partitions, since it defaults to ext2 if the type isn't specified.

This command also lets you specify blocksize, volume labels, percentage of blocks that are reserved for root, a UUID, and so forth. In order to get a complete list, you can check the manpages of mkfs and mkfs.<type> (replace "<type>" with the actual format). I won't go into further detail today, since I don't have my testing machine for formatting present. If anyone would like a more detailed explanation of mkfs, send me an email and I'll do so.

Find:

How many of you have ever been looking for a file, and find that Nautilus just doesn't cut it? I realize there are certain alternatives for desktop search, but I also know that "find" is an integrated utility in most distributions, making it ideal to learn.

```
find /home/ -name "*~"
```

This command will search through the /home/ folder (and any subdirectories, such as other users) for "tilde files" (backups of files from their previous edit). I find this useful for scripts where you want to automate cleaning up certain files. As you can probably figure out, "-name" tells find to only display files that contain what

> **[...] ever been looking for a file, and find that Nautilus just doesn't cut it?**

is enclosed in the quotes. In this case, I tell it to show me anything that ends in a tilde (the asterisk is a wildcard, which means anything). You can also specify any path you'd like instead of "/home/". Please note that, if you plan to search a folder that you have no read permissions for, or simply want to search your entire drive, you will need to run the find command as root with "sudo". Otherwise, you'll get permission denied errors. If you want to use it in a script, and don't want to access the folders you don't have permissions for, you'll have to grep -v it (inverse grep matching). I find this an excellent resource if you're willing to spend a while longer for the search (it can be extremely slow when searching large folders), but expect up-to-date and exact matches. Otherwise, for quick "where is this file" searches, I use "locate".

Locate:

This is a program that uses a database of files that's indexed, in order to quickly find a result. This database is updated regularly, but you should get into the habit of forcing an update before searching for a more recent file. You can do so with the following command:

```
sudo updatedb
```

This can take a few minutes to complete, but it shouldn't need to be done every time - only if you're looking for something you'd practically just downloaded/installed. Once the database is updated, you can run a search with the following command:

```
locate "*~"
```

As you may have noticed, since locate searches a database, you will receive search results in all folders of the hard drive, even ones you wouldn't have permission for using find, since updatedb is run as root. If you find you have too many results, you can use grep, head, or tail to reduce the search results. There are yet more commands you can use to find files, but I will only cover "where" and "whereis":

Where/Whereis:

These commands are intended to help you quickly find binaries of programs, and configuration folders. For example, if you install skype and run these commands, you see the following:

```
where skype
```

```
output: /usr/bin/skype
```

```
whereis skype
```

```
output: skype:
/usr/bin/skype.real
/usr/bin/skype
/usr/bin/skype.bak2
/usr/share/skype
```

As you can see, it returns results that are linked to a program's binary. If you try running where or whereis on a folder, you'll get 0 results. It's intended for a quick search for configuration files of a program, or the location of a binary. It also has the added bonus of making you sound like a caveman/hulk.

I hope that you've found these explanations clear and that you'll try some of these tips the next time you're looking for a file or wanting to reformat a drive. As always, if you have any questions, commands, or suggestions, you can reach me at lswest34@gmail.com. Also, make sure to put "C&C" or "FCM" in the subject line so I don't overlook it.

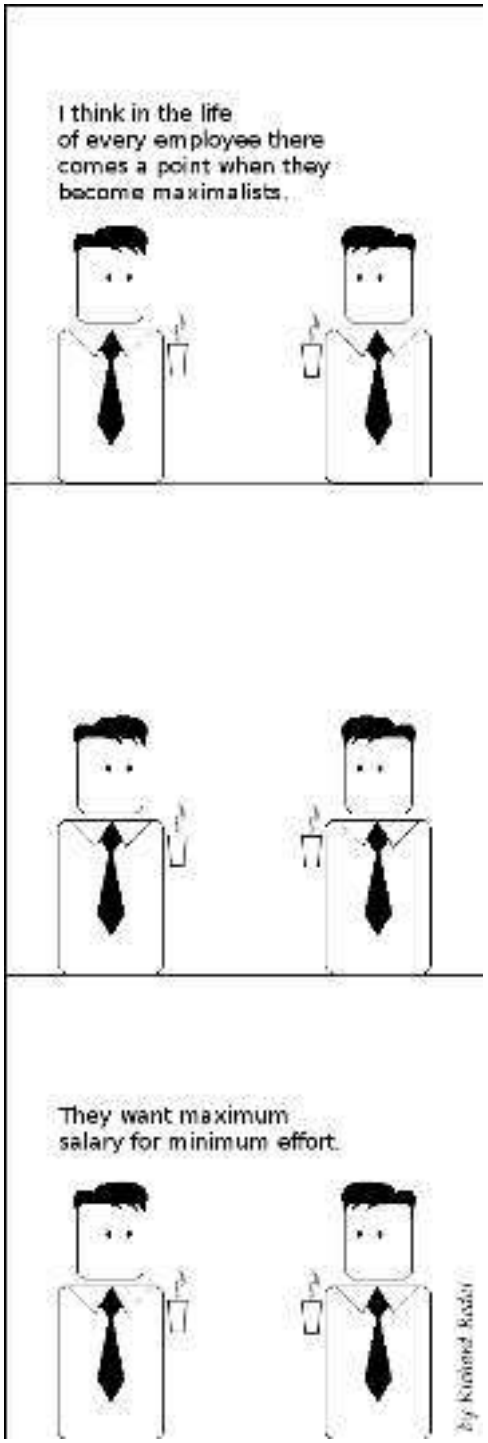
Further Reading:

http://www.linfo.org/mkfs.html (useful mkfs resource)

http://tldp.org/HOWTO/Partition/fdisk_partitioning.html (useful fdisk resource)

Manpages for all commands are also an excellent resource to start with.

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.



COMMAND & CONQUER

Written by Lucas Westermann

Before I start this month's article, I feel I should apologize for my usage of "cat," as a reader felt it deserved an award for the "worst usage of cat" (or words to that effect). I know that "tail" accepts files as well, and, as such, cat was superfluous, but I wanted to give an example of what cat does – not how best one should use it! If I hadn't done it as I had, I'd have needed another example command, which I thought was unnecessary. I apologize if I confused (or offended) anyone in doing so.

A year ago, I started to learn Japanese (mainly for a challenge), and the first real hurdle I faced was figuring out how to use an input-method system to allow for Japanese typing. Since I'd never heard of iBus, I decided on SCIM (Smart Common Input Method). These days, Ubuntu ships with iBus, which is similar to SCIM, though easier to configure.

I imagine that many readers of Full Circle need to use Japanese or Chinese input. Here is how you go about enabling the extended input.

Step 1

Open your Language Support window (found under System > Administration > Language Support in Ubuntu 10.10).

Step 2

Choose the "Install/Remove Languages" button in the main window (below)

Step 3

Scroll through the list, and check the box at the end of the line of the language you'd like (in my case, it was Japanese – see Fig. 2). After selecting the language, and closing the window, the language packs required by the chosen language (fonts, dictionaries, locales, etc.) will be installed.

Step 4

Once you're back in the main Language Support window, you'll need to choose "iBus" from the drop-down menu next to "Keyboard input method system".

Step 5

Log out and back in, in order to enable the extra systems and daemons required by iBus (you could do it by hand, but it could cause issues).

Step 6

Open your iBus preferences by going to System > Preferences > Keyboard Input Methods. Once it's opened, go to "Input Method" (see Figs. 3 and 4, below and next page), and choose your language from the drop-down box (Japanese > Anthy, or a method chosen from the Chinese menu). I can't recommend a method from the Chinese section, since I don't use Chinese input, but it's easily enough changed if you decide the method you're using isn't suitable. Once you select one, you must hit "add" to actually accept that Input

Method. The one at the top of the list is your “default” language (the one that is turned on/off when you enable or disable iBus).

Step 7

To use iBus, you'll need to simply hit Ctrl + Space to turn it on. Then, if you're typing Japanese, for example, you will type in rōmaji (ローマ字 i.e. latin alphabet), but the symbols will appear as you type the phonetic pronunciation. To generate the Kanji (漢字) for a word, simply hit space after you type it (so that にっぽんご becomes 日本語). If there are multiple options, just hit space again and it will show you a list of possible Kanji.

That should be everything you need to start typing away in Japanese. I hope some of you find it useful, and, as usual, if you have any questions, suggestions, or requests, send me an email at lswest34@gmail.com. Be sure to include “Command & Conquer” or “C&C” in the subject line, so I don't overlook it.

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

Full Circle Podcast

In this episode, UDS, Unity and no big guns...

In episode #13:

* **Review**: Issue #42 of Full Circle Magazine
* **News**: Ubuntu Developers Summit (UDS), Unity interface
* **Opinion**: Introducing "Ubuntu the Movie," (it's not an ad, apparently...)
* **Gaming**: The game sweeping the world: Minecraft and Ed previews new indie puzzle game 'And Yet It Moves'

File Sizes:
OGG 29.3Mb
mp3 23.8Mb

Runtime: 54min 40seconds
Released: 06th Nov. 2010

http://fullcirclemagazine.org/

COMMAND & CONQUER

Written by Lucas Westermann

Lately, I've seen a large influx of intriguing Conky setups in the Arch Linux forums, which gave me the idea of sharing a few tips and tricks I use for all my setups. I will be covering only one specific trick this month, but I will be giving you some extra things to work with. First off, for those of you who don't know what Conky is, it's a text-based system monitor, which can be displayed on your desktop, piped into dzen (popular in some tiling window managers), or have it floated as a panel on its own.

By default, Conky has a lot of options (from displaying the time and date, to memory or hard-disk usage). However, it does not offer a way to display the number of updates available for your machine – which is understandable since there are so many packaging formats and systems out there. So here's where another feature of Conky comes into play – being able to execute custom scripts and have their output displayed in Conky itself. There are two variations of this command – one with a refresh interval (what we'll need for the update checker), and a single-run execute (useful if the script itself refreshes). All these settings are controlled from the file .Conkyrc within your home folder. To start you off, here's a basic .Conkyrc I use (I've stripped it of all custom scripts – since I don't plan on sharing them all): http://fullcirclemagazine.pastebin.com/jMDg9kzG

> **I've taken the liberty of commenting all the options for Conky...**

As you can see, I've taken the liberty of commenting all the options for Conky, and you may also notice that Updates: displays nothing at the moment. I'll be explaining how I came up with the script for that, and showing you how to implement it in just a second. First, I want to preface it by saying that the example command isn't the shortest (you can do the same with sed and some regular expressions), but it's the most readable example that I can think of. If you want to practice some regex, feel free to replace grep and cut with sed. As for the script, here it is:

```
#!/bin/bash

updateChecker=`apt-get -s upgrade|grep upgraded,|cut --delimiter=" " -f1`

echo "$updateChecker";
```

Now, how I came up with the command (what's in "updateChecker") is quite simple. I ran:

```
apt-get -s upgrade
```

and looked at the output, then found the line with the number of updates and found a unique word within that line, and then re-ran the command while piping it into "|grep upgraded,". Once I was certain it gave me the right line, I simply took that line, cut it into fields (delimited, meaning separated, by spaces), and displayed the first field ("-f1"), since that was the number I wanted. I then re-ran the whole command, made sure it returned the number properly, copied it to the bash script, and wrote the echo line to return it. If you wanted to have it as a less Conky-specific script, you could just add "Updates:" (without the quotes) to the echo line (before $updateChecker), and delete the word "Updates:" from the .Conkyrc file. As for implementing it in Conky, all you need to do is adjust the update line to this:

```
${font DejaVuSans:bold:size=8}Updates:$font ${execi 300 /path/to/script}
```

Where, of course, /path/to/script is the actual path, and the script is set to executable. To make the script executable:

```
chmod +x /path/to/script
```

My last check is always running the script from the terminal to make sure it works right, but in this case you can safely skip that step (Conky will let you see if it's

working right, after all!).

Hopefully, this has shown some new users that the command-line isn't just ugly black and white text, but can add something to your graphical setup as well. For any of you who are interested in a Bash script that removes extra kernels (while leaving the 2 most recent), have a look below at the "further scripting" section. If you found this useful and/or interesting, I'd be happy to share some more scripts in the next few months. And as always, if you have any corrections, questions, or suggestions, you can send me an email at lswest34@gmail.com. Keep in mind that "C&C" or "FCM" should appear in the subject line, so I don't overlook it.

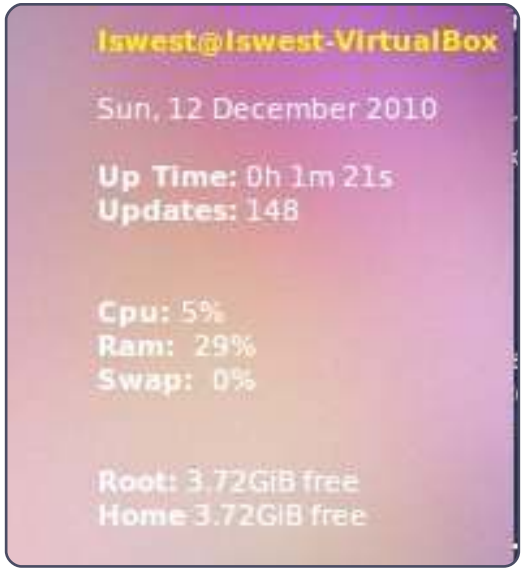
Further Scripting

If anyone is wondering why I would have a kernel manager, it was written for an Ubuntu/Windows dual-boot computer, where Windows was the default grub option, meaning any kernel updates screwed up defaults. Instead of teaching the new user how to update Grub2, I simply set up a symbolic link on the desktop for the script that they needed to run when the list got longer.

My "kernel manager" - for anyone who might find it interesting (can also be adapted into a Conky script to display number of installed kernels): http://fullcirclemagazine.pastebin.com/0JzTHjJ1

The program is pretty well commented, but here's the gist of what it does:

- Checks the folders in /usr/src and counts how many linux kernel folders there are.
- It then stores the actual names in another variable.
- If there are 4 (or fewer) folders, echo "nothing to do.", and exit.
- Otherwise ($folders > 4) display how many kernels need to be deleted, show which ones exactly are being deleted (for security's sake).
- Once the user has read this, ask if it should continue.
- If the input is "y" then remove the kernels, and wait 3 seconds to ensure all output is done.
- Wait until the user hits enter to quit (otherwise any error messages get lost).
- If none of the if-statements apply, let the user know there was an error.



Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

Full Circle Podcast

In this episode, Narwhals, Wayland and… I almost forgot, Amnesia!

In episode #14:

* **Review**: Issue #43 of FCM
* **News**: Ubuntu 11.04 Alpha1, Wayland, System 76, Android 2.3, Fee-paying Games in Ubuntu Software Centre, Flash 10.2 Beta.
* **Gaming**: Humble Indie Bundle #2 and Amnesia

File Sizes:
OGG 42.5Mb
mp3 34.9Mb

Runtime: 1hr 18min 58seconds
Released: 19th Dec. 2010

http://fullcirclemagazine.org/

COMMAND & CONQUER

Written by Lucas Westermann

Conky - Part 2

I'm going to use this month to cover a few extra things you can do with Conky for any who wants a slightly more GUI-like system monitor. Before I start though, if anyone is wondering why I set up a script for the updates instead of embedding the code within the actual .conkyrc file, you can check out my response in the Letters section of this issue.

If you have ever seen a screenshot of my desktop (http://lswest.deviantart.com/), you'll notice I have a very noticeable "Music" section in my main Conky. If you're wondering what I'm talking about, look at my newest screenshot (shown right) here: http://lswest.deviantart.com/#/d360pfy. Basically, what I do within that script is read the song information from the MPD (Music Player Daemon), and then pull an image from albumart.org using the artist and album name. If I can't get the script to fetch it automatically, I can download it and save it in my .covers folder with the correct naming scheme, and it will be used. As for displaying it in Conky, I have my script save a copy of the current song as /tmp/cover, and then have Conky simply display that image (updated every 15 seconds). This is partially contained within an if statement, to avoid having empty information displayed when MPD is not running. I have put together a basic .conkyrc file for my script here: http://lswestfcm.pastebin.com/rrCS0hDt, and the script is here: http://lswestfcm.pastebin.com/iX7Y7W3v

And now, time for some explaining. The very first line of the .conkyrc ("imlib_cache_size 0") tells Conky to not cache images (necessary since you want it updated). The other lines are all pretty much the same as last month, with the exception of the MPD stuff, which simply tells Conky where it can find my daemon. This is required only if you're using MPD for music. If you're using something else, you'll have to adjust the Python script to read the album and artist information from whatever system you are using, and replace any MPD-specific calls in the conkyrc file. The other sections should be fairly self-explanatory, but this is what happens after the TEXT header:

I force it to use UTF-8 in order to display any special characters that may crop up

```
Write my "MUSIC" header

Display the status

Start of If statement
      · Run python script every 2 seconds
      · Display /tmp/cover at a position (100,330) (format of (x,y) coordinates) with a size of 50px by 50px, updated every 15 seconds
      · Display artist
      · Display title
      · Display a bar with the time passed and total time on either end.
End of If statement.
```

> **I force it to use UTF-8 in order to display any special characters that may crop up...**

I realize this is a short article, and possibly not too relevant (I'm unsure of how many Ubuntu users run MPD), but I find it extremely useful, and good practice for anyone who wants to learn a bit more about Conky. Also, if you haven't tried MPD out before, I highly recommend it. Since it's a daemon, it saves its state on shutdown, so you can resume your music instantly after a reboot, or continue a song from where you stopped it. It has a number of front-ends, some of which I covered in an earlier article (page 25 of FCM#32). For a small example of what else you can do with Conky, I present to you this link: https://bbs.archlinux.org/viewtopic.php?pid=875306#p875306. It's a post from the Arch Linux forums displaying a Conky that runs GUI elements from lua scripts. I believe Conky has lua support enabled by default now, but it's possible the version in the Ubuntu repositories has this disabled, so keep that in mind if you decide to try the configurations.

As always, I hope at least a few have found this useful, and I'd like to hear your opinions on MPD front-ends, uses for Conky, any feedback you might have or any ideas for articles you might have! You can reach me at lswest34@gmail.com and please put "C&C" or "Full Circle Magazine" in the subject line, so I don't overlook it! Also, I'd much prefer it if any emails were in English or German – I had one in French the other day and it was rather difficult to decipher. On that note, the reader was pointing out the existence of "file" - which is a program that displays information about a file's type (i.e. JPEG, MP3, etc.).

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

Full Circle Podcast

In episode #15: Brainstorms, FUD and Media Players

* **Review**: FCM#44.
* **News**: Brainstorm ideas, Software Centre ratings, Fuduntu, Unity, Android, and more!
* **Gaming**: Humble Indie Bundle 2, Mass Effect, FreeCiv, and Dropbox.

File Sizes:
OGG - 46.9Mb
mp3 - 40.4Mb

Runtime: 1hr 24min 34sec
Released: 13th Jan. 2011

http://fullcirclemagazine.org/

To-Do List

This issue I intend to cover two topics: creating a to-do list using Conky and Bash, and introducing the basics of Zenity.

Both of these were requests. I'll be covering the way I use Conky with my To-Do list, and I'll be suggesting a few other additions one could make. However, I won't go into any of the additions in great detail unless I have an influx of requests for it.

To-Do List

The way I do a To-Do list is simply by creating a symbolic link in my Dropbox folder to a directory I called *Reminders*. Within the directory, I have a bunch of files sorted into topics (university, FCM, work, and personal), and then in Conky I call a Python script (which I wrote) that goes to each file, and prints each line with a "-" before it. This can be done in any language one would like. Shown above is my Python script for anyone who is interested.

```
#!/usr/bin/env python
import os

home=os.path.expanduser("~")

for root, dirs, files in
os.walk(os.path.join(home,"Reminders")):
    for infile in [f for f in files]:
        if(infile.endswith("~")!=True):
            fh=open(os.path.abspath(os.path.join(root,infile)))
            for line in fh:
                print("- "+line, end=' ')
            fh.close()
```

createToDo.sh:

```
#!/bin/bash
file=~/ToDo.txt
toDo=~/toDo.txt
if [[ -e $toDo ]]; then
    `rm "$toDo"`
fi
while read line; do
    date=`date -d"$(echo "$line"|sed 's/\(.\) -.*$/\1/g')" +%s`;
    echo "$(echo "$line"|sed -e s/".*-"/"$date -"/g)" >> "$toDo";
done < "$file"

if [[ -e "$toDo" ]]; then
    temp=`sort -n "$toDo"`
    echo "$temp" > "$toDo"
fi
```

Be sure to change "Reminders" into "<path from home directory>" for it to work - the variable home takes care of the "/home/$USERNAME/" part, so specify only the portion after that. I also tell it to ignore any hidden backup files - files that end with a tilde (~). I realize that my method is extremely low-tech, but it works. If you want to add due-dates in, simply type them into the file while you're adding items to it. Once you start using dates though, it becomes difficult to sort the items properly. For that, I've written two scripts. The first one is shown right, the other is shown at the top of the next page.

What these two scripts do is quite simple. The createToDo.sh script takes all the items within a file (I put it into the $file variable), removes the date (in the format: Month Day HH:MM), and replaces it with unix times (seconds since the unix epoch, aka 1st of Jan 1970 00:00), which is then written into a file (a file that is deleted at the beginning of each run of the script, to avoid duplicates). Once it has done this, it takes the file and sorts it from smallest to largest number (nearest date to "latest" date, i.e. first thing due to last thing due).

printToDo.sh:

```
#!/bin/bash
toDo=~/toDo.txt
while read line; do
    if [[ "$line" != "" ]]; then
        date=`date -d@"$(echo "$line"|sed -e s/"-[^-]*$"//g)" +"%a %b %d %H:%M"`
        echo "$(echo "$line"|sed -e s/".*-"/"$date -"/g)";
    fi
done < "$toDo"
```

The printToDo.sh script simply takes each line from within the newly-created toDo.txt file, replaces the epoch time with a normal date, and prints it out. The first script could become an hourly cronjob (or every few minutes, if you prefer), and the second one could be run from Conky as a normal bash script. I haven't done too much testing for these scripts, but they do certainly work. I'm not sure if there is an easier method, but I think these scripts might teach/introduce you to more.

If you improve the scripts, I'd like to hear about it, and I'd be more than happy to list a few solutions in next month's C&C.

Zenity

For those of you who don't know Zenity, it's a command-line tool intended to create dialog boxes (graphical elements). Since I know that many users are shy of the command-line when first starting off, this might be useful for anyone who is trying to help a beginner along. I plan to cover the extreme basics as an introduction to Zenity, and I will write an in-depth tutorial for next month.

Zenity is capable of creating text-entry windows, calendar dialogs, info windows, progress dialogs, notification icons, list dialogs, save dialogs, checklists, error windows, and so forth. A few examples are as follows:

```
<command>|  tee  > (zenity --progress --pulsate) >file
```

This command runs a progress bar as long as tee is reading in input from STDIN, and then saves it in a file. In this case, you have to pipe the output of any command to tee.

```
zenity --question --text "Question?"; echo $?
```

This command creates a dialog box with a question and an OK/Cancel button. Echo $? returns 0 if the user hits OK, and 1 if the user hits cancel (useful for user-intervention). This is the exact syntax for warnings as well - simply replace "--question" with "--warning".

```
<command>|zenity --text-info --width <size in pixels>
```

This command takes the output of whatever command is piped to it, and prints it within a textbox within the dialog box.

```
input=$(zenity --entry --text "How are you?" --entry-text "enter text here");
echo $input
```

This Zenity command creates a text-input window, and returns the entry within the variable $input (hence the echo statement).

```
zenity --error --text "An error occurred!"
```

This command creates an error window and puts the text in the window.

There are quite a few more commands that Zenity offers, but this should get any enthusiastic coder prepared for most scripting needs. Next month, I plan to implement some of these commands into a useful little script. If anyone has a request as to what the script should do, feel free to send me an email at lswest34@gmail.com. If you do email me, please put FCM or C&C in the subject line, so that I don't overlook it! Any comments on this article, or requests in general, are always welcome.

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

Conky and Lua

I realize that last month I said I was going to do an article on Zenity within a script. However, I couldn't think of a script that would benefit from Zenity - without getting extremely complicated. Instead, I decided I would go further into Conky, specifically the ability to use lua scripts to draw graphical items to the desktop (in this case, rings, but I'm sure other objects are possible). Before we get started, I'd like to make the disclaimer that I am in no way a lua coder, and there is a good chance that there are easier ways to make the changes I've made, but it's what I came up with.

For those of you who aren't sure what I'm talking about, this screenshot on my DeviantArt profile is an example of what is possible with Conky: http://lswest.deviantart.com/#/d3ay5fb

First and foremost, make sure you have installed Conky (1.7.2 is in the repositories as of version 9.10 Karmic Koala). I will assume that everyone is using version 9.10 or higher. If you're using an older version, launchpad will probably have a PPA for you.

As for the widgets we'll be creating, I'm going for a simple MPD music widget, and a clock (the same widgets visible in the above screenshot). Also, since I felt no need to re-create the wheel (or, in this case, the ring), I will be using the following script as a basis for what we are doing now: http://londonali1010.deviantart.com/art/quot-Rings-quot-Meters-for-Conky-141961783. For those of you who would like the complete scripts (for reference/correction), see the Scripts section at the end of this article.

Pre-coding

Before we get started on the actual script, I ask you to decide if you want it in two separate Conky instances (my choice) or within a single instance. The reason why I use two is quite simply because I have two other Conky instances on my desktop, and merging the widgets into one would have resulted in overlaps between Conky instances. If you wish to use only one, you will need to increase the minimum size, and adjust the x and y values for each widget to place them within the Conky window. The x and y relate to the relative position of Conky. For example, if Conky starts at (400,200) ((x,y)) then a widget with placement (100,85) will actually be at (500,285) on your monitor. Keep this in mind.

Also, to use the lua scripts, you must add in the following to your .conkyrc:

```
lua_load /home/lswest/conky_testing/rings-v1.2.lua

lua_draw_hook_pre ring_stats
```

...where the top line is, of course, the actual absolute path to the lua script, and the name below is the name of your main function (if you write conky_ring_stats, or

Clock

Configuring the clock.

For each ring you want, you will need to configure an entry in the settings table which will look like this:

```
{
        name='time',
        arg='%I',
        max=12,
        bg_colour=0xffffff,
        bg_alpha=0.1,
        fg_colour=0xffffff,
        fg_alpha=0.4,
        x=165, y=170,
        radius=89,
        thickness=7,
        start_angle=0,
        end_angle=360
},
```

ring_stats, it will find the function regardless of which variation you use within the actual script).

The script has comments to clarify the entries, but I'll quickly explain each as well. The name is actually the name of the Conky variable (i.e. ${time }), the args are the arguments (i.e. ${time %I}), and it is parsed by the script in lines 121-131 (on pastebin), within the local function setup_ring. It basically sends the command (after formatting it into ${name args}) to Conky, gets the result, and parses it. Then it's casted into a number, and the deviations (entered into the max variable) of the ring are calculated (so, if you say 360 divisions, then each division is exactly 1° of the ring, or if you have 12, then it's 2*pi/max (in radians)). It's not important if you don't understand this, just keep in mind that to have 12 hours within the ring, you must make 12 divisions. The 4 following variables are simply background and foreground colors, and their alpha (transparency) levels. The x and y variables make up your position vector, radius is the width of the ring, thickness is the size of the line, start_angle is where the circle starts (0°), and end_angle is where it stops (360°), so that we get a complete circle.

For those of you who know the formatting for the date command, you'll know that %I is the format for the hours with leading 0s (so 01...12). The format for a 12-hour clock, without leading 0s, is %l, but it doesn't matter for this clock - I also had it working fine with %H (0...23). The next two rings I made smaller and made the seconds ring 2 pixels thinner. In the end, you should have something along the lines (after the hours ring) of the code shown right.

As you can see, it's fairly straightforward. If you're fine with the seconds ring counting off the seconds, and without the date in the center, you're finished. If you, like me, want the seconds to fill the innermost circle, then you'll need to add the following line before “cairo_arc(cr, xc, yc, ring_r, t_arc-arc_w, t_arc+arc_w)”:

```
if pt['arg'] == '%S' then
cairo_arc(cr, xc, yc,
ring_r, angle_0,
t_arc+arc_w) end
```

What this does is simply start at the angle_0 (12 o'clock on the ring), and extends the line. My first reaction is to put the original line into an else statement, but, it works without it, and it's a little less typing, so we'll forgo good formatting in this case. If you want to place the date within the center of the ring, it's a bit of guess-work for the positioning, but here is what you need to add to your .conkyrc:

```
${goto 115}${voffset
150}${time %A}

${goto 115}${time %b %d %Y}
```

The goto line shifts it over to the right (you can also use ${offset <pixels>}), and voffset is the vertical offset (i.e. pixels moved down from the top of the conky window). What I did was display the day on the top line (${time %A}), and the date on the line below it. If you want to change the way it's displayed, checking the manpage of date will give you the formatting options you need.

MPD Widget

Now before we start this, the widget I describe here works only for MPD (Music Player Daemon), since Conky lacks variables for other music players. I'm sure you could get it working the same with some tricky coding, but I don't think it's worth it as most other music players have a “now-playing” widget of sorts.

The settings_table entry for this one looks like the code shown on the next page, top left.

```
{
      name='time',
      arg='%M',
      max=60,
      bg_colour=0xffffff,
      bg_alpha=0.1,
      fg_colour=0xffffff,
      fg_alpha=0.4,
      x=165, y=170,
      radius=79,
      thickness=7,
      start_angle=0,
      end_angle=360
},
{
      name='time',
      arg='%S',
      max=60,
      bg_colour=0xffffff,
      bg_alpha=0.1,
      fg_colour=0xffffff,
      fg_alpha=0.4,
      x=165, y=170,
      radius=70,
      thickness=5,
      start_angle=0,
      end_angle=360
},
```

```
{
      name='mpd_percent',
      arg='',
      max=100,
      bg_colour=0xffffff,
      bg_alpha=0.1,
      fg_colour=0xffffff,
      fg_alpha=0.4,
      x=70, y=170,
      radius=60,
      thickness=7,
      start_angle=0,
      end_angle=360
},
```

As you can see, we're working with 100 divisions (since it's a percent, it will be a value between 0 and 100). Also, the arg variable is empty, which is important, since leaving it out entirely makes it incompatible with the functions we use later (missing argument). Once you've done this, I also altered the script so that the widget disappears when the music is paused. To do this, you need to make the following changes to the script:

Add this function to the beginning or end of the file:

```
function
conky_my_flag(my_arg)
    flag = my_arg
    return ""
end
```

Then, place the following text from the original script:

```
local
updates=conky_parse('${update
s}')
    update_num=tonumber(updat
es)

    if update_num>5 then
        for i in
pairs(settings_table) do
            setup_rings(cr,se
ttings_table[i])
        end
    end
```

inside of the following if statement:

```
if tonumber(flag) == 1 then
<text from above>
end
cairo_destroy(cr)
```

So that the last 11 or so lines of the file read as shown above right.

What the above changes do is simply to destroy the widget if MPD isn't running, and otherwise to run as normal. The function we created is so we can assign a value to the global variable flag that we use within the if-statement. Now, before this script works, you'll need to add in ${lua my_flag 0} and ${lua my_flag 1} into your .conkyrc so that the function is called and

```
if tonumber(flag) == 1 then
    local updates=conky_parse('${updates}')
    update_num=tonumber(updates)

    if update_num>5 then
        for i in pairs(settings_table) do
            setup_rings(cr,settings_table[i])
        end
    end
end
cairo_destroy(cr)
end
```

```
${lua my_flag 0}
${if_mpd_playing}
${lua my_flag 1}
${execi 2 python /usr/bin/mpd-cover}
${image /tmp/cover -p 40,138 -s 60x60 -u 15}
${if_match "${mpd_status}" == "Paused"}
${offset 137}${voffset 40}${font
DejaVuSans:bold:size=10}Paused
$endif
${if_match "${mpd_status}" == "Playing"}
${offset 137}${voffset 20}${font
DejaVuSans:bold:size=10}${mpd_artist}
${offset 137}${font DejaVuSans:size=9}${scroll 38
${mpd_title}}$font
$endif
$endif
```

the flag variable is set to 0 or 1, depending on if MPD is stopped (0), or not (1). The TEXT section of my .conkyrc looks like the code shown above.

What this does is set the flag variable to 0 when if_mpd_playing is false. Otherwise it gets set to 1.

The rest of the settings display and position the album art, display “Paused” if MPD is paused, or the Artist and Song Title on two lines to the right of the ring if MPD is playing. The ${scroll 38 ${mpd_title}} section causes the title to scroll (so the text moves from right to left) if it's longer

than 38 pixels. You can leave this out, but I put it in there to prevent the text from being longer than my Conky is wide. In order to display the image, you'll need to add the following two settings above the TEXT marker somewhere:

```
imlib_cache_size 0
```

Also, the mpd-cover script is below, in the Scripts section. The mpd-cover script is written for python 2.X, but you can always use the 2to3 program to re-write it for python 3. If you have issues, let me know. Be aware that some symbols can cause problems with the script. I have done very little editing (if any) to it, and it was originally from here: https://bbs.archlinux.org/viewtopic.php?id=112708

Hopefully the majority of you have found this interesting, and, as always, I'm open to requests, suggestions, general feedback, and questions. You can reach me at lswest34@gmail.com, and remember to put C&C or FCM into the subject line, so I don't overlook it. Also, English or German are my preferred languages, because otherwise I will have to rely on Google Translate. If anyone improves the scripts I have listed/used here, feel free to send a copy to me with an explanation of additions/changes, and I will note it at the beginning of the next article for anyone who is interested.

Scripts:

http://pastebin.com/SpC6bcn7
Lua clock ring
http://pastebin.com/iZFdZAeg
Conky mpd
http://pastebin.com/zkVVHkYk
.conkyl_mpd
http://pastebin.com/BDa5MHuR
conkyrc for clock
http://pastebin.com/ZX4pLbta
mpd-cover script

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

Asian Language Support

This month, I felt I would share with you something I just recently learned about. The topics I'll be covering apply only to those readers who either use iBus/SCIM and aren't happy with it, or who have it running and are happy with it - but whose Japanese/Chinese/etc. doesn't appear in a legible font in rxvt-unicode. Also, I'd like to take a moment to announce that next month I hope to do a question and answer session for C&C readers. If you have questions about Linux in general, the command-line, or me as an author, feel free to send your questions to lswest34@gmail.com before the 28th of April. I will be selecting a bunch of questions to answer next month. Requests for articles are also welcome.

As some of you probably know, I wrote an article on iBus in issue #43 of FCM. I hadn't used iBus since I was comfortable with SCIM. However, an update recently disabled SCIM, and so I tried iBus. What really got me was that I couldn't switch between hiragana and katakana easily, so I decided to take a suggestion from a friend of mine and tried out uim. Surprisingly, uim doesn't block my dead keys in rxvt-unicode, and allows easy switching between hiragana and katakana. Below is how I configured it for use.

uim & uim-fep:

From the homepage (http://code.google.com/p/uim/):

"uim's goal is to provide simple, easily extensible and high code-quality input method development platform, and useful input method environment for users of desktop and embedded platforms. See what's uim? for further information."

First, you'll need to install it:

```
sudo apt-get install uim uim-gtk2.0 uim-qt uim-qt3 uim-fep uim-anthy
```

This should cover uim support for terminals, QT applications, and GTK applications using anthy. There are a number of other packages offering applets, different dictionaries, and utilities, that may be of interest to some people.

Once you've installed it, running uim-toolbar-gtk-systray will give you a system tray icon. Right-click on it and choose preferences. Here, I would adjust the list of enabled input languages to only the ones you need, and adjust the global key bindings to your preferences. If you find that the system tray icon is practically invisible, it's because too much information is being displayed in the one "icon" width. To adjust this, open the preferences, and, under "Toolbar", uncheck everything, and set the enabled toolbar buttons per language that you use to just "Input Mode". This will reduce it to one icon - making it readable again. Also, in order to get it working, you'll need to add the following to /etc/profile (or .bashrc, or .zshrc):

```
export XMODIFIERS=@im=uim

export GTK_IM_MODULE="uim"

export QT_IM_MODULE="uim"
```

Once you've set these variables, you should run the following in a terminal:

```
gtk-query-immodules-2.0 > /etc/gtk-2.0/gtk.immodules
```

This will re-create the gtk.immodules file, which specifies to GTK programs which Input Method types are available.

Uim-fep is a Front-End Processor for terminal emulators. Basically, it allows you to type Japanese in a terminal emulator (rxvt-unicode in my case), without relying on uim-xim (which is a bit of a resource hog). In order to get it working, you'll need to add uim-fep to the end of your .bashrc, or your .zshrc, or whatever shell you're using. If you get a warning that uim-fep is already running, you can add "clear" (without the quotes) after it, so that it hides the message. Once it's running, you'll have a line at the end of your terminal that looks something like

this:

Using the global shortcut for uim will result in the icon at the end changing to the input method, and allows you to type Japanese in-line in the terminal.

Rxvt-unicode:

In case you have the problem that your Japanese is nearly unreadable in rxvt-unicode (this may apply to other terminal emulators as well, but I haven't tested it), then you can add the following to your .Xdefaults:

```
URxvt.preeditType:
OnTheSpot,None

URxvt.imLocale: ja_JP.UTF-8

URxvt.font: xft:Anonymous
Pro:size=11:antialias=true:au
tohint=false,xft:IPAGothic:si
ze=11:antialias=true

URxvt.boldFont: xft:Anonymous
Pro:size=11:weight=Bold:antia
lias=true:autohint=false,xft:
IPAGothic:size=11:weight=Bold
:antialias=true
```

This, basically, tells urxvt to expect Japanese input from uim. The fonts are actually a list of two, as you can see. Anonymous Pro is the terminal font I use for everything, but if rxvt-unicode can't find the symbols for something in that font, it will move on to the next one in the list (or a fallback font if there is no such symbol in any font listed). This allows you to have support for multiple languages without compromising the readability of Latin symbols. Also, you may see some people using urxvt.* instead of URxvt.* - which can be problematic if you set the name of your terminal from a shortcut (i.e. urxvt -name ncmpcpp -e ncmpcpp). The first section of these preferences tells the system that the WM_CLASS of the program is that we want to affect, and the lowercase “urxvt” is the first of the list, which is set using the -name argument. If, instead, you use “URxvt”, then it will not change depending on the -name switch. To see what I mean, enter the following command into a terminal, and click on rxvt-unicode.

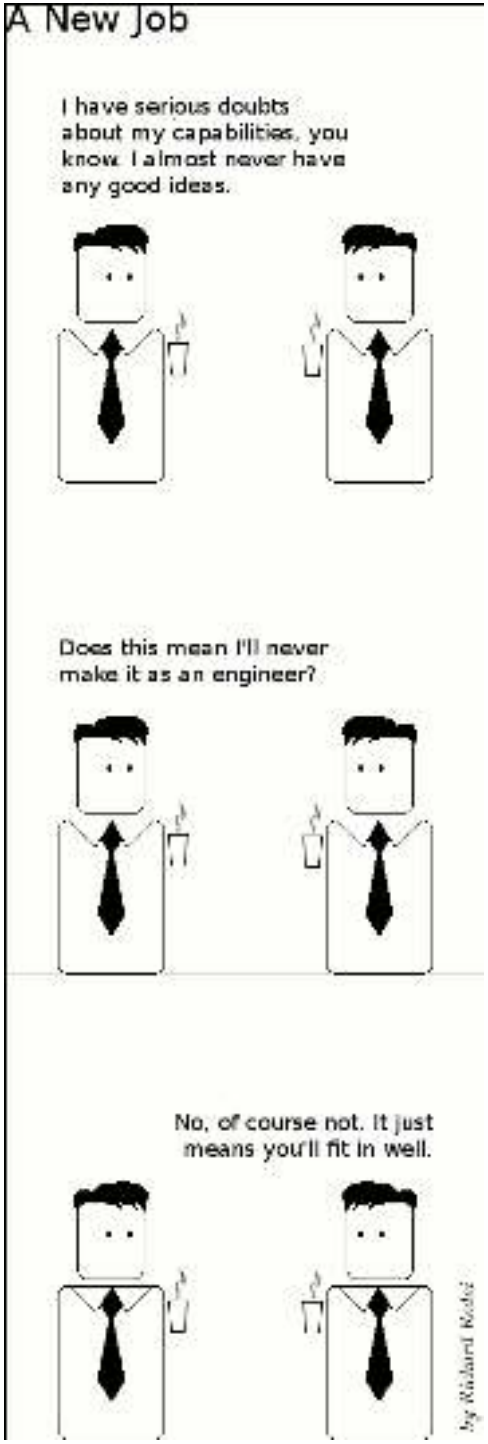
```
xprop|grep "^WM_CLASS"
```

Which gives you something like this:

```
WM_CLASS(STRING) = "urxvt",
"URxvt"
```

Now you should have a fully functional uim setup, and shouldn't have had to compromise any functionality in your terminal either. If you have any suggestions, or requests for articles, feel free to email me at lswest34@gmail.com. Also, don't forget your questions! I will need the questions sent in before the 28th of April!

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.



Graphicsmagick

On April 22nd, reader John Niendorf contacted me to request an article on Graphicsmagick. In response to his email, I plan to cover the basic use-cases of graphicsmagick (henceforth referred to as “gm”), a more advanced use (batch processing), and creation of MIFF files as visual image directories.

If your needs are anything like mine, you'll find that gm is excellent if you want to do a batch conversion, or if you want to quickly create a thumbnail from a large image without opening a graphics app. Before reading the list, please take into account that <something>* means that it can be repeated indefinitely, and anything in square brackets is optional (but useful to know about). So, without further ado, here's a list of commands I find useful, and keep in mind these are basic frameworks:

`gm identify <file>`
lists information on format and size of image, and also displays status of file (incomplete, corrupted, etc).

`gm montage [<options><input>]* <output>`
Combines all the input files into the single output file, with some formatting options (tiling, display image name below image, etc).

`gm mogrify <options> <input file>`
Transforms the file.

`gm convert <options> <input file> <output file>`
Transforms the file (same as mogrify, except that mogrify overwrites the file).

`gm composite <file to change> <base file> [<mask file>] <output file>`
Merges, blends, and masks the files to create a new image.

This is just a very basic outline of the possible commands, and a basic framework of arguments that can be used with it. As for common options:

`-geometry <height>x<width><+/-><x><+/-><y>`
This option specifies the geometry of the image window, including x and y offset. Example: -geometry 1600x1050+10+10.

`-size <height>x<width>`
This option is passed before the input file, and allows jpeg images to be read in as a small size, in order to cut down processing time. Great for batch creation of thumbnails. Example: -size 170x160.

`-thumbnail <height>x<width>`
This option uses preset options to create a thumbnail quickly.

`-resize <height>x<width>`
This option actually scales the image to the supplied size.

`-gaussian <radius>{x<sigma>}`
This option applies a gaussian blur to the image. Sigma refers to the standard deviation. Generally you'll need only the radius option.

`-quality <value>`
This option sets the quality of the output image (for JPG/MIFF/TIFF/PNG). <value> can be an integer between 0 and 100 (where 100 = best quality, lowest level of compression).

> **[Graphicsmagick] is excellent if you want to do a batch conversion...**

`-crop <width>x<height>{+-}<x>{+-}<y>{%}`
This option allows you to crop the image to the size you specify (and supply an offset).

This list of options should be enough to get you started and experimenting. Once you've found a command you like, with suitable options, you may want to apply it to a large section of files within the current directory. In order to do so, you would use a command similar to this (see below for explanation of options):

```
find . -name "*jpg" | xargs -l -i basename "{}" ".jpg" | xargs -l -i gm convert -quality 100% "{}.jpg" "{}.png"
```

Here, find . -name “*jpg”

returns a list of all jpg files in the current directory, which gets passed to xargs, which goes line-by-line ("-l") and removes the suffix(".jpg") from the list ("{}") using basename. Afterwards, the list is passed to xargs again, and it then executes gm convert -quality 100% "{}.jpg" "{}.png", which essentially takes each image and converts it to a png file. The middle-step is necessary to avoid having files called "*.jpg.png" after batch is complete. This trick could also be used for cropping, editing, or resizing a large number of files.

Last, but not least, I'll be covering how to create a visual image directory (a file of thumbnails of the images within a folder). To create the file, use this command:

```
gm convert 'vid:*.jpg'
directory.miff
```

The miff extension stands for the ImageMagick Magick Image File Format. The reason for the format is due to the fact that gm was forked from imagemagick back in 2002. In order to display the file afterwards, simply run the command:

```
gm display directory.miff
```

If you're wondering why this might be useful, imagine having thousands of photos on one PC, and you're looking for a single one. Instead of working on that computer and trying to find the file, you could copy over the miff file and browse it at your leisure, or use it to create a catalogue of thumbnails.

Hopefully you've found the tips in this article helpful, and will continue to put them to good use. If you have any requests or questions, you can reach me at lswest34@gmail.com. Please put the words "Command & Conquer", "C&C", "Full Circle Magazine", or "FCM" in the subject line, so I don't overlook it. Also, please try to write the emails in English or German, since I otherwise have to rely on Google Translate.

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.

Full Circle Podcast

AUDIO MP3 AUDIO OGG

Your Hosts:
Robin Catling
Ed Hewitt
Dave Wilkins
Audio: Victoria Pritchard

Show Notes

00:42	WELCOME and INTRO
01:04	SINCE LAST TIME…
04:55	REVIEW Issue #48 of Full Circle Magazine
12:05	REVIEW: Ubuntu 11.04
57:26	CONTRIBUTE
1.26:23	FEEDBACK
1.27:19	OUTRO AND WRAP

contents ^

Before I begin this month's article, I would like to share two of ***John Niendorf***'s uses for gm (thanks for sharing them!). They are:

```
alias imgresize='gm mogrify -resize 640x480 *.jpg *.JPG'
```

```
alias frameall='gm mogrify -mattecolor yellow -frame 5x5+0+5 *.JPG *.jpg *.jpeg *.png'
```

These aliases could be either pasted into your .bashrc file, or to a dedicated aliases file. The upper command resizes all the jpeg files to 640 x 480, and the second one adds a frame around all jpeg and png files.

Now, on to this month's article. As some of you may know, typing mathematical formulae (in lectures, or classes, or for any other reason) within programs such as OpenOffice or LibreOffice is not the easiest thing in the world. Especially when you start getting into set theory or other advanced mathematical concepts with Greek letters, symbols like "for all", and so forth. For these sorts of things, I highly recommend using LaTeX (see Issue 11 for a basic introduction to LaTeX). In this article, I will be introducing you to some math packages and some useful tips and tricks for formatting mathematical formulae nicely. As for software, I'm quite fond of Texmaker, and the texlive packages offered in the official Ubuntu repositories should include all the packages I refer to here.

Document Preamble

The preamble is all the text included before the \begin{document} line in Latex. This includes document settings, headers, footers, package imports, and package settings. My typical math documents contain the following packages:

tikz (for diagrams/graphs, for which I load the decorations.markings tikz library)
amsmath – offers enhancements to all basic mathematical functionality
amsfonts – offers special math formatting (math calligraphy (\mathcal{}), math block text (\mathbb{}), etc.)
amssymb – offers the ability to display numbered equations, in-line math, etc.
hyperref (when using a table of contents) – Allows the creation of click-able links in TeX documents.

Below is an actual preamble that I use for my Linear Algebra notes (the document section

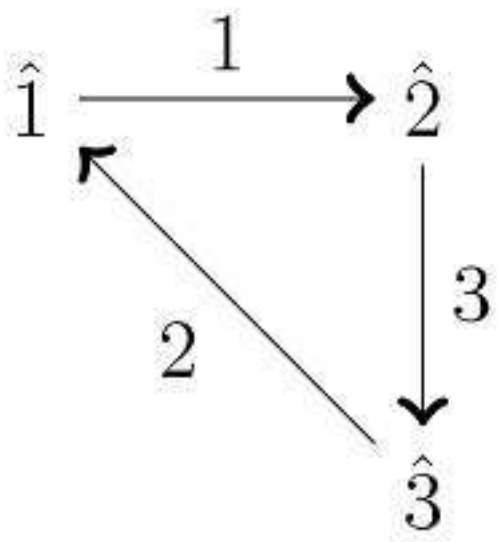
```
\documentclass[12pt,a4paper]{article}
% page counting, header/footer
\usepackage{fancyhdr}
\usepackage{lastpage}
\usepackage[ngerman]{babel}
\usepackage{tikz}
\usetikzlibrary{decorations.markings}
\usepackage{amsmath}
\usepackage{amsfonts}
\usepackage{amssymb}
\usepackage{graphicx}
\usepackage[utf8]{inputenc}
\usepackage{hyperref}
\addtolength{\oddsidemargin}{-.525in}
\addtolength{\evensidemargin}{-.525in}
\addtolength{\textwidth}{1.5in}
\hypersetup{unicode=true,pdfborder={0 0 0 [0 0]},
linkcolor=blue}
\title{Lineare Algebra}
\author{Lucas Westermann}
\pagestyle{fancy}
\fancyhead{}
\fancyfoot{}
\fancyhead[L,L]{Lineare Algebra}
\fancyhead[R,R]{Lucas Westermann}
\fancyfoot[R,R]{Seite \thepage\ von \pageref{LastPage}}
\fancyfoot[L,L]{\hyperlink{contents}{Inhaltsverzeichnis}}
\renewcommand{\headrulewidth}{0.4pt}
\renewcommand{\footrulewidth}{0.4pt}
\setlength{\headheight}{16pt}
```

consists of only two include statements – and the references for utf8 and ngerman are because my lecture is in German).

TikZ is probably the most complicated package to use, so I will cover it first. The following is the code used to create graph A:

```
\begin{tikzpicture}[node
distance=2cm, auto]

\node (1) {$\hat{1}$};

\node (2) [right of = 1]
{$\hat{2}$};

\node (3) [below of = 2]
{$\hat{3}$};

\draw[decoration={markings,ma
rk=at position 1 with
{\arrow[ultra
thick]{>}}},postaction={decor
ate}] (1) to node {1} (2);

\draw[decoration={markings,ma
rk=at position 1 with
{\arrow[ultra
thick]{>}}},postaction={decor
ate}] (2) to node {3} (3);

\draw[decoration={markings,ma
rk=at position 1 with
{\arrow[ultra
thick]{>}}},postaction={decor
ate}] (3) to node {2} (1);

\end{tikzpicture}
```

Graph A



This code creates 3 nodes (named 1, 2 and 3). The information in the braces ("{}") is the label for the node (what is displayed), so leaving it blank results in an empty node. The next three lines "\draw" create the lines between nodes (using the node names – which is in the normal brackets), and labelled again by what's within the braces.

Using the math packages to create and align equations:

```
\begin{align*}

(\mathbb{K}_1^1) & \alpha
+(\beta + \gamma )= (\alpha
+ \beta)+\gamma \\

(\mathbb{K}_1^2) & \alpha +
0 = 0 + \alpha = \alpha \\
(\mathbb{K}_1^3) & \alpha
\cdot -\alpha = -\alpha
\cdot \alpha = 0 \\

(\mathbb{K}_1^4) & \alpha +
\beta = \beta + \alpha

\end{align*}
```

This results in the text shown below.

The align* environment allows you to use tabbing characters ("&") to align the text nicely. This is especially useful when doing a proof, and you want to align the equations at their equals signs. The \mathbb{} results in the blocked K. The "_" and "^" refers to sub and superset. If you have a super/subset that is longer than a single character, you will need to enclose it in braces. The \alpha, \beta, and \gamma refers to the Greek letters. The \cdot is a multiplication sign, the double backslashes denote line breaks, and the rest is self-explanatory.

Other useful commands are things like:

"**\forall**" (the upside-down A symbol)
"**\exists**" (reversed E)
"**\in**" (the curved e-symbol used when referring to sets)
"**\cup**" (union symbol – set theory)
"**\cap**" (intersection symbol – set theory)
"**\mathcal{}**" (makes the letter in braces cursive – used by my professor when referring to a basis – a set of linearly independent

$$(\mathbb{K}_1^1)\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$$
$$(\mathbb{K}_1^2)\alpha + 0 = 0 + \alpha = \alpha$$
$$(\mathbb{K}_1^3)\alpha \cdot -\alpha = -\alpha \cdot \alpha = 0$$
$$(\mathbb{K}_1^4)\alpha + \beta = \beta + \alpha$$

vectors)

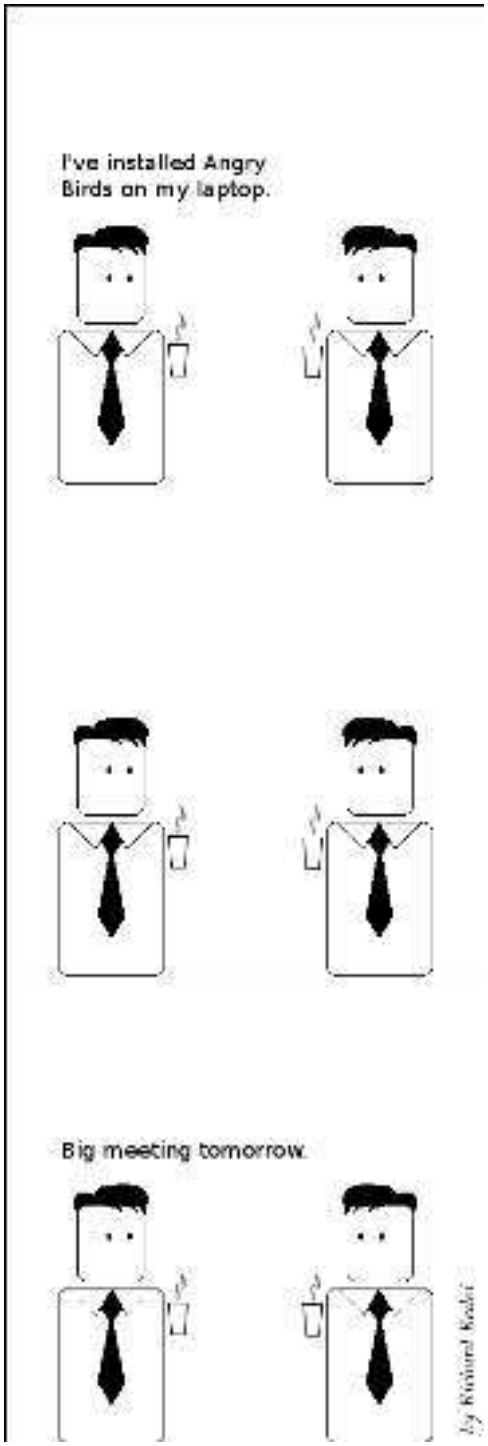
I hope you've found this article useful. I could have covered more examples, but regardless of how many I covered, it would still suit only a small number of use-cases. As such, you should view these as examples of what you can do with LaTeX. See the further reading section for a link to a useful manual. If you've any questions, or requests, email me at lswest34@gmail.com. Please put the words Full Circle Magazine, FCM, or C&C in the subject line, so I don't overlook it.

Further Reading:

http://en.wikibooks.org/wiki/LaTeX – Wikibook covering a large amount of standard uses of LateX. For all other non-standard uses, a quick google search should suffice.

http://sourceforge.net/projects/pgf/ - Included in the zip file (see the Downloads section) is a great manual for many uses of the TikZ package.

Lucas has learned all he knows from repeatedly breaking his system, then having no other option but to discover how to fix it. You can email Lucas at: lswest34@gmail.com.



Server Circle is a new question and answer site run by techies.

Users with any level of experience can ask technical questions for free about anything server related, and receive answers from trusted experts, who are rated by the community.

With time you can earn reputation points, and even financial rewards, by contributing your answers to questions from other people.

http://www.servercircle.com

NOTE: Server Circle is not affiliated with, nor endorsed by, Full Circle magazine.

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.