LINUX VOICE TUTORIAL

LINUX 101: POWER UP YOUR SHELL

MIKE SAUNDERS

Get a more versatile, featureful and colourful command line interface with our guide to shell basics.

WHY DO THIS?

- Make life at the shell prompt easier and faster.
- Resume sessions after losing a connection.
- Stop pushing around that fiddly rodent!

As a Linux user, you're probably familiar with the shell (aka command line). You may pop up the occasional terminal now and then for some essential jobs that you can't do at the GUI, or perhaps you live in a tiling window manager environment and the shell is your main way of interacting with your Linux box.

In either case, you're probably using the stock *Bash* configuration that came with your distro – and while it's powerful enough for most jobs, it could still be a lot better. In this tutorial we'll show you how to pimp up your shell to make it more informative, useful and pleasant to work in. We'll customise the prompt to make it provide better feedback than the defaults, and we'll show you how to manage sessions and run multiple programs together with the incredibly cool tmux tool. And for a bit of eye candy, we'll look at colour schemes as well. So, onwards!

1 MAKE YOUR PROMPT SING

Most distributions ship with very plain prompts – they show a bit of information, and generally get you by, but the prompt can do so much more. Take the default prompt on a Debian 7 installation, for instance:

```
mike@somebox:~$
```

This shows the user, hostname, current directory and account type symbol (if you switch to root, the **$** changes to **#**). But where is this information stored? The answer is in the **PS1** environment variable. If you enter **echo $PS1** you'll see this at the end of the text string that appears:

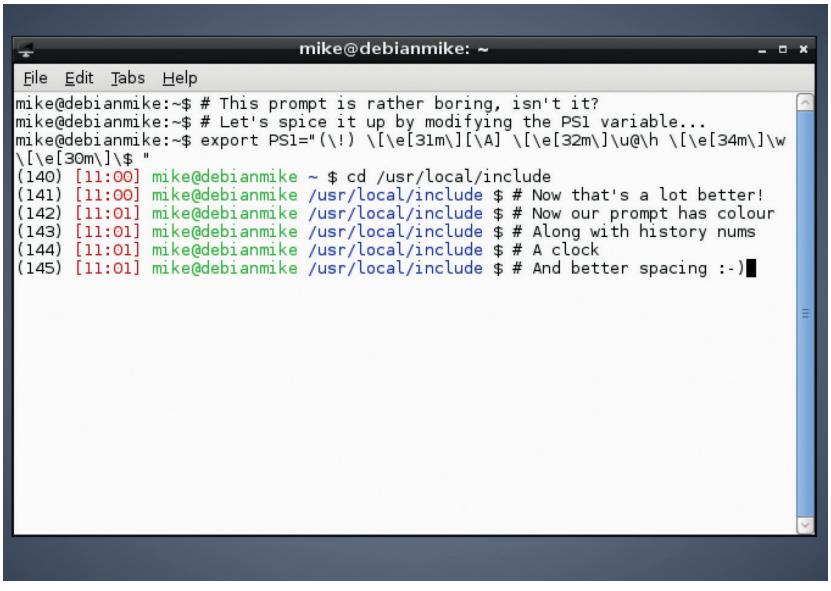
```
\u@\h:\w\$
```

This looks a bit ugly, and at first glance you might start screaming, assuming it to be a dreaded regular expression, but we're not going to fry our brains with the complexity of those. No, the slashes here are escape sequences, telling the prompt to do special things. The **\u** part, for instance, tells the prompt to show the username, while **\w** means the working directory.

Here's a list of things you can use in the prompt:

- **\d** The current date.
- **\h** The hostname.
- **\n** A newline character.
- **\A** The current time (HH:MM).
- **\u** The current user.
- **\w** (lowercase) The whole working directory.
- **\W** (uppercase) The basename of the working directory.
- **\$** A prompt symbol that changes to # for root.
- **\!** The shell history number of this command.

To clarify the difference in the **\w** and **\W** options: with the former, you'll see the whole path for the directory in which you're working (eg **/usr/local/bin**), whereas for the latter it will just show the **bin** part.



Here's our souped-up prompt on steroids. It's a bit long for this small terminal window, but you can tweak it to your liking.

Get customising

Now, how do you go about changing the prompt? You need to modify the contents of the **PS1** environment variable. Try this:

```
export PS1="I am \u and it is \A \$"
```

Now your prompt will look something like:

```
I am mike and it is 11:26 $
```

From here you can experiment with the other escape sequences shown above to create the prompt of your dreams. But wait a second – when you log out, all of your hard work will be lost, because the value of the **PS1** environment variable is reset each time you start a terminal. The simplest way to fix this is to open the **.bashrc** configuration file (in your home directory) and add the complete export command to the bottom. This **.bashrc** file will be read by *Bash* every time you start a new shell session, so your beefed-up

prompt will always appear. You can also spruce up your prompt with extra colour. This is a bit tricky at first, as you have to use some rather odd-looking escape sequences, but the results can be great. Add this to a point in your **PS1** string and it will change the text to red:

```
\[\e[31m\]
```

You can change 31 here to other numbers for different colours:

- **30** Black
- **32** Green
- **33** Yellow
- **34** Blue
- **35** Magenta
- **36** Cyan
- **37** White

So, let's finish off this section by creating the mother of all prompts, using the escape sequences and colours we've already looked at. Take a deep breath, flex your fingers, and then type this beast:

```
export PS1="(\!) \[\e[31m\][\A] \[\e[32m\]\u@\h \[\e[34m\]\w \
[\e[30m\]\$ "
```

This provides a *Bash* command history number, current time, and colours for the user/hostname combination and working directory. If you're feeling especially ambitious, you can change the background colours as well as the foreground ones, for really striking combinations. The ever useful Arch wiki has a full list of colour codes: **http://tinyurl.com/3gvz4ec**.

Shell essentials

If you're totally new to Linux and have just picked up this magazine for the first time, you might find the tutorial a bit heavy going. So here are the basics to get you familiar with the shell. It's usually found as *Terminal*, *XTerm* or *Konsole* in your menus, and when you start it the most useful commands are:

ls (list files); **cp one.txt two.txt** (copy file); **rm file.txt** (remove file); **mv old.txt new.txt** (move or rename); **cd /some/directory** (change directory); **cd ..** (change to directory above); **./program** (run program in current directory); **ls > list.txt** (redirect output to a file).

Almost every command has a manual page explaining options (eg **man ls** – press Q to quit the viewer). There you can learn about command options, so you can see that **ls -la** shows a detailed list including hidden files. Use the up and down cursor keys to cycle through previous commands, and use Tab after entering part of a file or directory name to auto-complete it.

2 TMUX: A WINDOW MANAGER FOR YOUR SHELL

A window manager inside a text mode environment – it sounds crazy, right? Well, do you remember when web browsers first implemented tabbed browsing? It was a major step forward in usability at the time, and reduced clutter in desktop taskbars and window lists enormously. Instead of having taskbar or pager icons for every single site you had open, you just had the one button for your browser, and then the ability to switch sites inside the browser itself. It made an awful lot of sense.

If you end up running several terminals at the same time, a similar situation occurs; you might find it annoying to keep jumping between them, and finding the right one in your taskbar or window list each time. With a text-mode window manager you can not only run multiple shell sessions simultaneously inside the same terminal window, but you can even arrange them side-by-side.

And there's another benefit too: detaching and reattaching. The best way to see how this works is to try it yourself. In a terminal window, enter **screen** (it's installed by default on most distros, or will be available in your package repositories). Some welcome text appears – just hit Enter to dismiss it. Now run an interactive text mode program, such as **nano**, and close the terminal window.

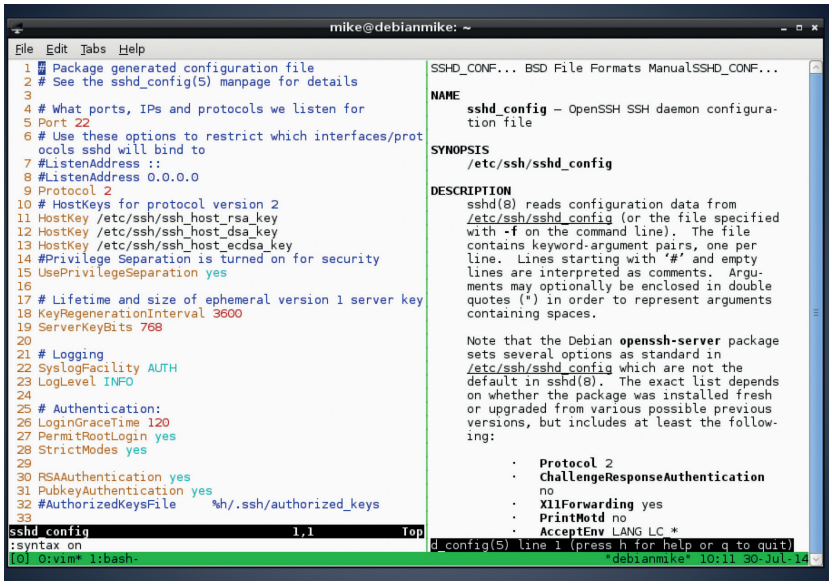
In a normal shell session, the act of closing the window would terminate every process running inside it – so your *Nano* editing session would be a goner. But not with *screen*. Open a new terminal and enter:

```
screen -r
```

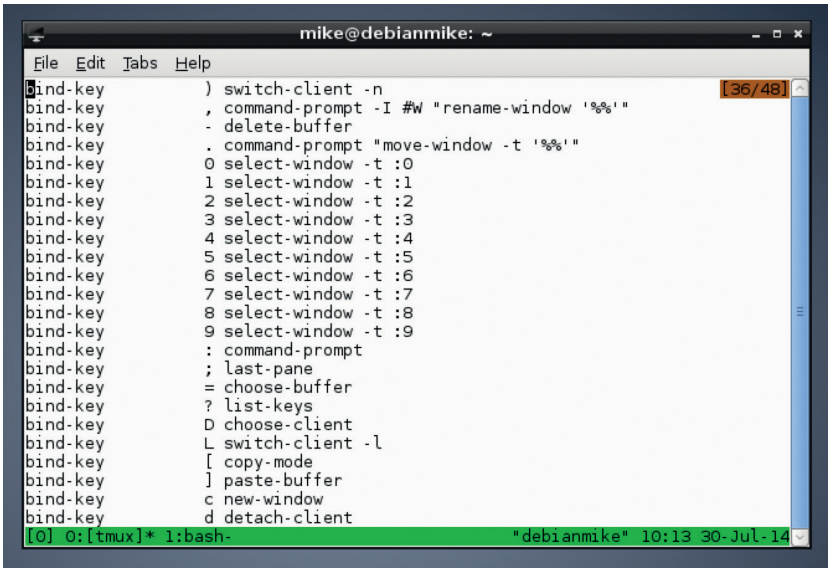
And *voilà*: the *Nano* session you started before is back!

When you originally ran **screen**, it created a new shell session that was independent and not tied to a specific terminal window, so it could be detached and reattached (hence the **-r** option) later.

This is especially useful if you're using SSH to connect to another machine, doing some work, and don't want a flaky connection to ruin all your progress. If you do your work inside a **screen** session and your connection goes down (or your laptop battery dies, or your computer explodes), you can simply reconnect/recharge/buy a new computer, then SSH back in to the remote box, run **screen -r** to reattach and carry on from where you left off.



Here's **tmux** with two panes open: the left has Vim editing a configuration file, while the right shows a manual page.



In **tmux**, hit Ctrl+B followed by ? to get a list of the default key bindings.

Now, we've been talking about GNU *screen* here, but the title of this section mentions *tmux*. Essentially, *tmux* (terminal multiplexer) is like a beefed up version of *screen* with lots of useful extra features, so we're going to focus on it here. Some distros include *tmux* by default; in others it's usually just an **apt-get**, **yum install** or **pacman -S** command away.

Multiplexing magic

Once you have it installed, enter **tmux** to start it. You'll notice right away that there's a green line of information along the bottom. This is very much like a taskbar from a traditional window manager: there's a list of running programs, the hostname of the machine, a clock and the date. Now run a program, eg *Nano* again, and hit Ctrl+B followed by C. This creates a new window inside the *tmux* session, and you can see this in the taskbar at the bottom:

"Tmux enables you to have multiple programs running inside a single terminal window."

```
0:nano- 1:bash*
```

Each window has a number, and the currently displayed program is marked with an asterisk symbol. Ctrl+B is the standard way of interacting with *tmux*, so if you hit that key combo followed by a window number, you'll switch to that window. You can also use Ctrl+B followed by N and P to switch to the next and previous windows respectively – or use Ctrl+B followed by L to switch between the two most recently used windows (a bit like the classic Alt+Tab behaviour on the desktop). To get a window list, use Ctrl+B followed by W.

So far, so good: you can now have multiple programs running inside a single terminal window, reducing clutter (especially if you often have multiple SSH logins active on the same remote machine). But what about seeing two programs at the same time?

For this, *tmux* uses "panes". Hit Ctrl+B followed by % and the current window will be split into two sections, one on the left and one on the right. You can switch between them Using Ctrl+B followed by O. This is especially useful if you want to see two things at the same time – eg a manual page in one pane, and an editor with a configuration file in another.

Sometimes you'll want to resize the individual panes, and this is a bit trickier. First you have to hit Ctrl+B followed by : (colon), which turns the *tmux* bar along the bottom into a dark orange colour. You're now in command mode, where you can type in commands to operate *tmux*. Enter **resize-pane -R** to resize the current pane one character to the right, or use **-L** to resize in a leftward direction. These may seem like long commands for a relatively simple operation, but note that the *tmux* command mode (started with the aforementioned colon) has tab completion. So you don't have to type the whole command – just enter "**resi**" and hit Tab to complete. Also note that the *tmux* command mode also has a history, so if you want to repeat the resize operation, hit Ctrl+B followed by colon and then use the up cursor key to retrieve the command that you entered previously.

Finally, let's look at detaching and reattaching – the awesome feature of *screen* we demonstrated earlier. Inside *tmux*, hit Ctrl+B followed by D to detach the current *tmux* session from the terminal window, which leaves everything running in the background. To reattach to the session use **tmux a**. But what happens if you have multiple *tmux* sessions running? Use this command to list them:

```
tmux ls
```

This shows a number for each session; if you want to reattach to session 1, use *tmux a -t 1*. *tmux* is hugely configurable, with the ability to add custom keybindings and change colour schemes, so once you're comfortable with the main features, delve into the manual page to learn more.

Zsh: an alternative shell

Choice is good, but standardisation is also important as well. So it makes sense that almost every mainstream Linux distribution uses the *Bash* shell by default – although there are others. *Bash* provides pretty much everything you need from a shell, including command history, filename completion and lots of scripting ability. It's mature, reliable and well documented – but it's not the only shell in town.

Many advanced users swear by *Zsh*, the Z Shell. This is a replacement for Bash that offers almost all of the same functionality, with some extra features on top. For instance, in *Zsh* you can enter **ls -** and hit Tab to get quick descriptions of the various options available for **ls**. No need to open the manual page!

Zsh sports other great auto-completion features: type **cd /u/lo/bi** and hit Tab, for instance, and the full path of **/usr/local/bin** will appear (providing there aren't other paths containing **u**, **lo** and **bi**). Or try **cd** on its own followed by Tab, and you'll see nicely coloured directory listings – much better than the plain ones used by *Bash*.

Zsh is available in the package repositories of all major distros; install it and enter **zsh** to start it. To change your default shell from *Bash* to *Zsh*, use the **chsh** command. And for more information visit **www.zsh.org**.

Fine-tune your colour scheme

We're not obsessed with eye-candy at Linux Voice, but we do recognise the importance of aesthetics when you're staring at something for several hours every day. Many of us love to tweak our desktops and window managers to perfection, crafting pixel-perfect drop shadows and fiddling with colour schemes until we're 100% happy. (And then fiddling some more out of habit.)

But then we tend to ignore the terminal window. Well, that deserves some love too, and at **http://ciembor.github.io/4bit** you'll find a highly awesome colour scheme designer that can export settings for all of the popular terminal emulators (*XTerm*, *Gnome Terminal*, *Konsole* and *Xfce4 Terminal* are among the apps supported.) Move the sliders until you attain colour scheme nirvana, then click on the Get Scheme button at the top-right of the page.

Similarly, if you spend a lot of time in a text editor such as *Vim* or *Emacs*, it's worth using a well-crafted palette there as well. **Solarized at http://ethanschoonover.com/solarized** is an excellent scheme that's not just pretty, but designed for maximum usability, with plenty of research and testing behind it.

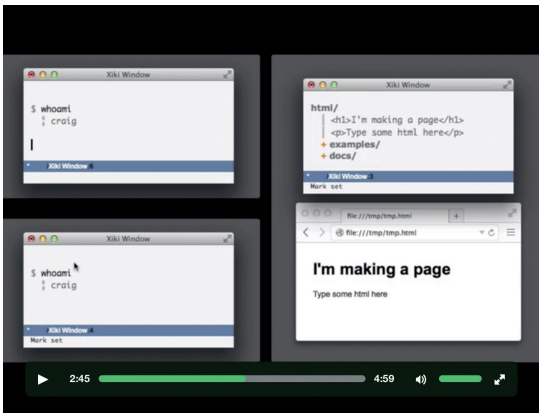
```
#!/bin/bash

cd $ROOT_DIR
DOT_FILES="lastpass weechat ssh Xauthority"
for dotfile in $DOT_FILES; do conform_link "$DATA_DIR/$dotfile" ".$dotfile"; dor

# }}}
# crontab update from file {{{
# TODO: refactor with suffix variables (or common cron values)

case "$PLATFORM" in
        linux)
        #conform_link "$CONF_DIR/shell/zshenv" ".zshenv"
        crontab -l > $ROOT_DIR/tmp/crontab-conflict-arch
        cd $ROOT_DIR/$CONF_DIR/cron
        if [[ "$(diff ~/tmp/crontab-conflict-arch crontab-current-arch)" == ""
            ]];
            then # no difference with current backup
                logger "$LOG_PREFIX: crontab live settings match stored "\
                    "settings; no restore required"
                rm ~/tmp/crontab-conflict-arch
```

The Solarized colour scheme might not look so swish on paper, but it works brilliantly on the screen to reduce eye strain during long coding sessions.

3 THE TERMINALS OF THE FUTURE

You might be wondering why the application that contains your command prompt is called a terminal. Back in the early days of Unix, people tended to work on multi-user machines, with a giant mainframe computer occupying a room somewhere in a building, and people connected to it using screen and keyboard combinations at the end of some wires. These terminal machines were often called "dumb", because they didn't do any important processing themselves – they just displayed whatever was sent down the wire from the mainframe, and sent keyboard presses back to it.

Today, almost all of us do the actual processing on our own machines, so our computers are not terminals in a traditional sense. This is why programs like *XTerm*, *Gnome Terminal*, *Konsole* etc. are called "terminal emulators" – they provide the same facilities as the physical terminals of yesteryear. And indeed, in many respects they haven't moved on much. Sure, we have anti-aliased fonts now, better colours and the ability to click on URLs, but by and large they've been working in the same way for decades.



Xiki aims to be both a more welcoming shell for new users, and a step-up for experienced CLIers.

Some programmers are trying to change this though. *Terminology* (**http://tinyurl.com/osopjv9**), from the team behind the ultra-snazzy *Enlightenment* window manager, aims to bring terminals into the 21st century with features such as inline media display. You can enter **ls** in a directory full of images and see thumbnails, or even play videos from directly inside your terminal. This makes the terminal work a bit more like a file manager, and means that you can quickly check the contents of media files without having to open them in a separate application.

Then there's *Xiki* (**www.xiki.org**), which describes itself as "the command revolution". It's like a cross between a traditional shell, a GUI and a wiki; you can type commands anywhere, store their output as notes for reference later, and create very powerful custom commands. It's hard to describe it in mere words, so the authors have made a video (see the Screencasts section of the *Xiki* site) which shows how much potential it has.

And *Xiki* is definitely not a flash in the pan project that will die of bitrot in a few months. The authors ran a successful Kickstarter campaign to fund its development, netting over $84,000 at the end of July. Yes, you read that correctly – $84K for a terminal emulator. It might be the most unusual crowdfunding campaign since some crazy guys decided to start their own Linux magazine...

Mike Saunders remembers using a mouse once. On the Amiga. Now he just wants kids to get off his damn lawn.

LV PRO TIP

Many command line and text-based programs match their GUI equivalents for feature parity, and are often much faster and more efficient to use. Our recommendations: *Irssi* (IRC client); *Mutt* (mail client); *rTorrent* (BitTorrent); *Ranger* (file manager); *htop* (process monitor). *ELinks* does a decent job for web browsing, given the limitations of the terminal, and it's useful for reading text-heavy websites such as Wikipedia.