Linux – Worst Kept Secrets

Hidden in plain sight

Linux has a wealth of command line tools. Everybody has their favorite. For every *cat* or *more*, there's probably another ten commands that rarely see the light of day. This month, Steven Goodwin uncovers some hidden gems that don't deserve to remain hidden for much longer. BY STEVEN GOODWIN

Finding information is not difficult. The real problem is knowing *what* to look for in the first place.

apropos

This utility will search through each man page looking for a particular word or phrase. This can be a keyword or regular expression, and so can be quite complex. It will then output each appropriate command, along with its description. So, if you're looking for a command that's related to passwords try as in Listing 1:

The *-e* switch will only match exact words (so in the above example, the phrase passwords would not get matched). As you are probably aware, the number in brackets indicates the section of the man pages where a particular command exists. See Table 1 for details. You can invoke a particular section by using:

```
man chage 1
```

whatis

whatis is a very simple command that produces a useful description of the command.

```
$whatis apropos
apropos (1) - search the manual➥
 page names and descriptions
```

This is the description that appears on the first line of the man page, and the same text that appears with the *apropos* command. *whatis* can also take regular expressions, and is often used to quickly check a command without having to format and display the whole man page.

file

Before reading a text file, with either *cat* or *more*, it is sensible to check that it is, in fact, text. Failure to do so may cause your screen to fill with junk, and require a *reset* command before your shell becomes legible again. This command lets us know in advance what type of file this is. It is an improvement over the use of file extensions (which may be wrong, or non-existent), as it reads a portion of the file and uses its contents to determine the type. For example, a text file will be one that has carriage returns in it, and a gzip file will begin with a specific header with the bytes \037 and \213.

file determines the type by using a file of magic numbers stored at */usr/share/misc/magic* and */usr/share/misc/magic.mgc*, the latter being a pre-compiled version of the original magic file, used for speed purposes. The magic file indicates which bytes need to appear in which position to indicate a particular file type. The file description can be as simple or complex as needed. MP3s, for example can indicate their bit rate and their recording frequency.

```
$ file Vexations.mp3
Vexations.mp3: MPEG 1.0 layer➥
 3 audio stream data, 48 kBit/s➥
, 44.1 kHz, jstereo
```

Anyone with superuser privileges can recompile a *magic.mgc* file with:

```
$ file -C
```

which

Linux commands are spread across different paths. My system has 2242 commands in seven different directories! (Press the *tab* key twice and let tab completion show you how many are available, followed by *echo $PATH*). Determining *which* command is executed from *which* directory is the job of a command called, er, *which*!

```
$ which file
/usr/bin/file
```

If an install, or compile, is causing problems, or you can not see why a particular command is not getting called, then *which* will indicate the command that is getting called in its place. This is helpful in finding out when older (or newer) versions of programs are being used erroneously. It can also be used with *file* to determine if the command about to be run is a program, symlink, or script.

```
$ file `which which`
/usr/bin/which: Bourne-Again ➥
shell script text executable
```

A Saucerful of Secrets

With everything in Linux being treated as a file, it's no surprise that a large number of commands exist to handle them.

Table 1: Man page sections

Section	Description
1	Executable programs or shell commands
2	System calls (functions provided by the kernel)
3	Library calls (functions within system libraries)
4	Special files (usually found in /dev)
5	File formats and conventions eg /etc/passwd
6	Games
7	Macro packages and conventions eg man(7), groff(7)
8	System administration commands
9	Kernel routines [Non standard]

Listing 1: apropos

```
01 $ apropos password
02 afppasswd (1)         - netatalk password maintenance utility
03 chage (1)             - change user password expiry information
04 chpasswd (8)          - update password file in batch
05 crypt (3)             - password and data encryption
06 ... and so on ...
```

tr

tr is short for translate, and substitutes one set of characters with another. It can also remove characters or squeeze multiple occurrences of them together to be replaced by a single instance. In all cases, input is taken from stdin, and fed to stdout. Translations are often used to demonstrate rot13 processing, or converting text into lowercase. In these cases, both sets of characters must be of equal size.

```
# we can use the A-Z shorthand ➥
rather than the alphabet in full
$ tr [A-Z] [a-z]  < afile   ➥
> a_lower_case_file
$ tr [a-zA-Z] [n-za-mN-ZA-M] ➥
< afile > a_rot13_file
```

However, *tr* has many more uses than that. It can be used to validate potential filenames by changing non-alphanumeric characters into a filesystem friendly underscore. This requires the -c (complement) option, meaning 'anything except the following'. As we have no way of knowing how many letters and numbers might exist in the complementary set, we can only replace with a single character.

```
$ uname -v | tr -c [a-zA-Z0-9] _
_44_SMP_Sun_Dec_28_19_07_54_➥
GMT_2003_
```

There is also a snappy shortcut for alphanumeric characters called *:alnum:*, which can be used in place of the rather verbose *[a-zA-Z0-9]*.

Another feature of *tr* lets us delete individual characters. This gives an easy and quick way of removing spurious carriage return codes from text files that have originated from within the Windows universe.

```
tr -d "\r" < windows.txt > ➥
linux.txt
```

Referring again to our filename fixer above, we could also have decided to delete the problematic characters, instead of changing them. Giving us,

```
$ uname -v | tr -c -d [:alnum:]
44SMPSunDec28190754GMT2003
```

Squeezing is a technique to compress several repeated characters into one. We could use this to replace multiple blank lines with a single carriage return.

```
$ tr -s '\n' <many_blank_lines ➥
>few_blank_lines
```

Note that when we redirect both input and output, the names must differ. This is because the redirection truncates the destination file to zero before the command executes. To replace the original file, you could include this command in a small script that uses *mktemp* to generate a temporary, and unique, filename.

cut

cut is a program that is used to extract columns of data from the input. A popular use of this command is to extract information from commands like *ls* and *ps* which are naturally tabulated. By default, the columns are considered separate by the presence of a tab. However, this can be changed to any arbitrary character or symbol with the -d switch. Each column is termed a *field*, and you can output individual fields, or groups therefore, using *-f*.

```
$ ls -l | tail +2 | tr -s ' ' |➥
 cut -d ' ' -f 3 | sort | uniq
root
steev
```

The above example uses a little extra massaging, firstly to remove the initial line from *ls* (*tail +2*, see Table 2) and then to squeeze the spaces between each field into one (*tr -s*). From here *cut* simply outputs field three (the user name) of each file in the directory. Since there are likely to be a lot of duplicates, we *sort* the names into alphabetical order and then use *uniq*, which acts like *tr*'s squash, but acts upon successively identical lines.

cut can output multiple fields by applying the *-f* option with a comma (which gives only the columns requested), or a hyphen (which shows a range of columns).

```
$ # Give the time & date ➥
of every file
$ ls -l | tail +2 | tr -s ' ' |➥
 cut -d ' ' -f 6-8
```

Table 2: Secret Arguments

Arguement	Description
unzip -a	Converts all text files to use Linux-style carriage returns when decompressing
echo -n	Used on its own will produce zero length files (normal echo adds a carriage return)
cat -n	Adds numbers before each line in the file. Very useful when printing program listings. To number non-blank lines use -b
tail +2	Outputs the whole file, starting from the second line. Useful for removing the headings on programs like *ls* and *ps*
grep -v	Instead of returning every line that matches, -v causes grep to match every line that doesn't. *grep -v grep* will therefore ignore the grep process(es) itself which can be useful when searching the process list
ls -1	Lists each file name on its own line, without the 'total' header