Oracle Scene TOP TIP – by Tim Onions, TOdC Limited

For this edition's tip I thought I would stray away from my usual comfortable database and/or SQL related topics and talk about something a little less familiar to me. My reasoning is simple, after many years of being nothing more than a knowledgeable UNIX user (the system administrators were there to do all the real UNIX stuff) I am now finding more and more need to get my hands dirty – and all because of Linux. I am not alone in this; I hear of organisations moving off Windows to Linux, because of the so called cost and stability benefits in doing so, all the time.

RLWRAP – making life easier when using SQL*Plus on Unix/Linux

The first thing that struck me, when I started using Linux directly (rather than simply using windows client tools to access a Linux hosted database), was relief that SQL*Plus was exactly the same (I'm a command-line DBA, not a GUI DBA). So for five minutes I was happy. Happy that is until my first typing error and I wanted to use the cursor keys to recall a statement and go back within the line via the left and right arrow keys. Instead of the cursor moving nicely along I got a line of ^[[S^[[C^[[A^[[D^[[D – which coincidentally spells out pretty accurately some of the expletives found coming from my mouth. Dropping out of SQL*Plus I try a similar edit of a command line at the host command prompt – and it works perfectly (this is Linux RHEL 4 by the way). Now I cannot live for long in a world without the arrow keys (those of you unfortunate enough to have me on instant messenger will know why). Enter rlwrap - the answer to this rookie's nightmare. Rlwrap is a wrapper process for Linux and Unix, using GPL libraries, that reads line input, interprets it and then sends it on to a command or process – hence the strange name, **r**ead **l**ine **wrap**per! So with rlwrap installed (a quick google will locate a score of places to get it from together with all the details required to get it going, although for simplicity I would recommend a pre-built copy rather than a tar or an rpm) and an alias in place (such as `alias sqlplus='rlwrap -i -c -f rlwrap.sqlplus.dict sqlplus'`) SQL*Plus will behave properly when any of the arrow keys are used.

It does not stop there however, just when I thought I was happy I was about to become happier still. Linux has the nice feature of filename completion when you press the tab key – so if I wanted to show the contents of the TimOnionsBirthdayPresentList2006_familyandfriendsonly file I could save my fingers (and frustration at getting that hideously long file name wrong 5 times) by just entering: `cat ./TimO` and then pressing the TAB key. Provided there were no other files starting with the four characters TimO, Linux would fill in the rest. Rlwrap can do the same kind of thing. Rlwrap will do that for you from within SQL*plus simply by adding the `-c` switch to the command line.

There are more goodies too. If you supply it with a dictionary (a word here that means a simple file containing a list of terms), then it will also use that dictionary to auto-complete words when the TAB key is pressed. So as we are discussing SQL*Plus if your dictionary contains a list of the most common terms you use in SQL*Plus (e.g. SELECT, INSERT, DELETE, UPDATE, EXIT, BEGIN, DECLARE, MERGE…) then you can save time and keystrokes using this feature. I guess in reality you would define a dictionary with more useful auto-complete terms. The get rlwrap to use the dictionary you simply use it with the `-f` switch followed by the name of the file that contains your dictionary. The `-i` switch makes the dictionary look-up case insensitive too. One of the distributions I found even came with a pre-built SQL*Plus dictionary file.

Rlwrap is a generic tool not limited to just SQL*Plus. So you can use it for any Linux application that can be wrapped (meaning to me any Oracle tool that I cannot type accurately into – RMAN; LSNRCTL;ASMCMD…). It also has numerous further options and features which, so far, I have yet to discover the true usefulness of. The online help (obtained via `rlwrap --help` shows the following):

```
[DCPT1+tonions@pt1db01 ~]$ rlwrap --help
Usage: rlwrap [options] command ...

Options:
  -a[password:]              --always-readline[=password:]
  -b <chars>                 --break_chars=<chars>
  -c                         --complete-filenames
  -C <name|N>                --command-name=<name|N>
  -f <completion list>       --file=<completion list>
  -h                         --help
  -H <file>                  --history-filename=<file>
  -i                         --case-insensitive
  -l <file>                  --logfile=<file>
  -n                         --no-warnings
  -P <input>                 --pre-given=<input>
  -m[newline substitute]     --multi-line[=newline substitute]
  -r                         --remember
  -v                         --version
  -s <N>                     --histsize=<N> (negative: readonly)
```

The “man” page (“man” as it seems to exist only on the web and does not getting added to your install’s man) gives further details still:

−a, −−always-readline [<*password_prompt*>**]**

Always use readline, regardless of *command*’s terminal settings. Use this option you want to use rlwrap with *command*s that already use readline, e.g. to get rlwrap’s history and completion. With this option, rlwrap will echo (and save) passwords; giving *command*’s password prompt as an argument will prevent this. The argument is optional; if given, it has to directly follow the option (−aPassword: or −−always-readline=Password:).

−b, −−break-chars <*list_of_characters*>

Consider the specified characters word-breaking (whitespace is

always word-breaking). This determines what is considered a "word", both when completing and when building a completion word list from files specified by **−f** options following (not preceding!) it. Default list (){}[],+-=&^%$#@"";|\ Unless -c is specified, / and . (period) are added to the default list.

−c, −−complete-filenames

Complete filenames (filename completion is always case-sensitive, even with the −i option) When doing this, rlwrap keeps track of *command*s working directory.

−C, −−command-name <*command_name*>|<*N*>

Use *command_name* instead of *command* to find the names of history and completion files, and to initialise readline (as specified in ~/.inputrc). A numeric argument N means: use the Nth argument counting backwards from the end of the argument list

−D, −−history-no-dupes *n*

How agressively to weed out duplicate entries from the input history. If *n* = **0**, all inputs are kept in the history list, if *n* = **1** (this is the default) consecutive duplicates are dropped from the list, while *n* = **2** will make **rlwrap** drop all previous occurrences of the current input from the list.

−f, −−file *file*

Put all words from *file* on the completion word list. This option can be given more than once, and *adds* to the default completion list in $RLWRAP_HOME or /usr/local/share/rlwrap.

−F, −−history-format *format*

Append *format* to each history entry, replacing the following printf-style conversion specifiers: **%D** by *command*s working directory, **%P** by the current prompt, **%C** by the *command* name, and all remaining format specifiers recognised by **strftime (3)**

format should start with one or more non-space characters, which are used to recognise and strip off the appended *format* from recalled history items.

−h, −−help

Print a short help message.

−H, −−history-filename *file*

Read command history from *file* (and write it back there if −−histsize >= 0)

−i, −−case-insensitive

Ignore case when completing (filename completion remains case-sensitive). This option has to come before any -f options.

−l, −−logfile *file*

Append *command*'s output (including echo'ed user input) to *file* (creating *file* when it doesn't exist).

−n, −−no-warnings

Don't print warnings.

−m, −−multi-line [<*newline_substitute*>**]**

Enable multi-line input using a "newline substitute" character sequence (" \ ", [space-backslash-space] by default). Newline substitutes are translated to newlines before sending the input to *command*. With this option, you can call an external editor $RLWRAP_EDITOR on the (expanded) current input with the *rlwrap_call_editor* key (CTRL-^ by default) The argument is optional; if given, it has to directly follow the option (−m';;' or −−multi-line=';;').

−P, −−pre-given *text*

Start rlwrap with *text* in edit buffer (this will automatically set the −−always-readline option). Use this for default user input in "one-shot" situations like the following replacement for the *read* shell command:

```
    rlwrap -H past_dinners -f toppings -P "pizza"\
      sh -c 'echo -n "please order: "; head -n 1 >
dinner'
    prepare $(dinner)
```

−r, −−remember

Put all words seen on in- and output on the completion list.

−s, −−histsize <*N*>

Limit the history list to N entries, truncating the history file (default: 300). A negative size -N means the same as N, but treats the history file as read-only.

−v, −−version

Print rlwrap version.

SPECIAL KEYS

Control + O

Accept current line, but don't put it in the history list. This action has a **readline** command name *rlwrap_accept_line_and_forget*

Control + ^

Use an external editor to edit the current input. This action has a **readline** command name *rlwrap_call_editor*

rlwraps special keys can be re-bound by including a line like the following in **~/.inputrc**:

"\M−\C−m":rlwrap_accept_line_and_forget # ESC-ENTER

cf. the **readline(3)** manpage for more about re-binding keys

ENVIRONMENT

RLWRAP_HOME: directory to keep the history and completion files.

RLWRAP_EDITOR (or else **EDITOR**, or else **VISUAL**): editor to use for multi-line input. Example:

```
export RLWRAP_EDITOR="emacs -nw"
export RLWRAP_EDITOR="vi +%L"
```

The last example is the default; %L and %C are replaced by line and column numbers corresponding to the cursor position in **rlwrap**

SIGNALS

A number of signals are forwarded to *command*: HUP INT QUIT USR1 USR2 TERM and (by way of resizing *command*s terminal) WINCH. Some care is taken to handle TSTP (usually a result of a CTRL-Z from the terminal) sensibly.

When *command* is killed by a signal, **rlwrap** will clean up, reset its signal handlers an then commit suicide by sending the same signal to itself. Thus, your shell will see the same exit status it would have seen without **rlwrap**.

REDIRECTION

When the standard input is not a terminal, **rlwrap** will check, and then ignore, all options and simply execute *command*. When stdout (or stderr) is not a terminal, rlwrap will re-open it to /dev/tty (the users terminal) after it has

started *command*

The upshot of this is that **rlwrap** *command* behaves more or less like *command* when redirecting.

EXIT STATUS

non-zero after an error, or else *command*'s exit status.

FILES

If $RLWRAP_HOME is not set, **rlwrap** uses hidden files in the users home directory

$RLWRAP_HOME/*command*_history, ~/.*command*_history

History for *command*

$RLWRAP_HOME/*command*_completions, ~/.*command*_completions

Per-user completion word list for *command*. **rlwrap** never writes into this list, but one can combine **−l** and **−f** options to to simulate the effect of a **−r** option that works across invocations.

/usr/local/share/rlwrap/*command*

System-wide completion word list for *command*. This file is only consulted if the per-user completion word list is not found.

$INPUTRC, ~/.inputrc

Individual **readline** initialisation file (See **readline** (3) for its format). **rlwrap** sets its *application name* to *command*, enabling different behaviours for different commands. One could e.g. put the following lines in **~/.inputrc**:

```
$if coqtop
    set show-all-if-ambiguous On
$endif
```

making **rlwrap** show all completions whenever it runs *coqtop*

The home site of rlwrap is http://utopia.knoware.nl/~hlub/uck/rlwrap/.

SCREEN – leave your sessions working whilst you detach yourself from work

Having used Linux for a few weeks I found myself needing to kick off a script, leave it running and periodically check on it. I messed about with nohup and the & suffix but they did not really do what I wanted – a session I could easily dip in and out of which continued as per normal whether I was there or not. Sort of like how Windows data center servers work when you connect via VNC, PC Anywhere or Netsupport or any third part remote control product (and definitely NOT like how Microsoft remote desktop works). The answer came via another GPL utility - the screen command. When you start a screen session it is like connecting to a new terminal window, you use it in exactly that way. However, when you are done with it you detach from it, yet it remains and continues doing whatever you left it doing – in my case a long and tedious SQL script. Once you have detached you can shutdown your PC if you like (I assume, like me, you use a terminal editor like putty to access the server). Go home if you want - the screen session is still alive and running. When you get home fire you your laptop, connect your broadband to your company VPN, and re-attach to screen. When you do so you will once again be connected to the session, will be able to see whatever output it has created since you were last on and carry on using it as if you had never left.

From my perspective screen has two major benefits:

1. When running detached it has no concept of a network connection to your PC. So if the VPN goes down (and they do all too frequently and at the most awkward of times) screen cares not and completes its tasks (you normal terminal session would die and Linux and Oracle would sort out your dead connection by rolling back any uncommitted work etc). Screen allows you to kick off that large job, detach and walk away knowing it will not suffer from a network glitch.

2. I can easily set up a series of scripts to run and leave them running whilst I get to do something else (eg. sleep). I can check back in at any time I like to see how the scripts are getting on. I can create as many screen sessions as I need – for example to run scripts in parallel – I name each screen session so I can easily identify which one is doing what. If necessary I can set-up my screen sessions to be multi-user so that other DBAs can check my screen sessions for me or they can be connected to the same screen session (from anywhere in the world network permitting) and I can demonstrate what I am doing.

So long as screen is installed on your system using it is simplicity itself. To create a new screen session enter:

```
screen –S SessionName –t "Window title"
```

Substituting `SessionName` and `"Window title"` with something meaningful to you.

To leave the screen session and return to your original window, but have screen continue processing, key in `Ctrl-a d` – you screen session will detach.

To re-attach to an existing screen session enter:

```
screen -r
```

if you have only one screen session created, or:

```
screen -r SessionName
```

If you have more than one screen session and you know the name of the session you want to attach to.

Forgotten if you have screen sessions running or what you named them? Not a problem, enter:

```
screen -ls
```

This will list out all screen sessions that you own with a process id and the session name. You can then connect to whichever session you want to by picking the session name (which hopefully means something to you).

Left screen active on your work desktop but want to view it from your laptop at home – attach with the `-x` option (multi-user), or force the office computer to relinquish control with the `-D` option.

Even without these extra features I find it very useful to use screen simply because I can leave the session connected for weeks on end and have the command history and environment available to me. I can pick up right from where I left off.

Mini-tip – Oracle DBA Toolbar for Internet Explorer.

Recently Oracle joined the ranks of Google and Yahoo and released a toolbar for IE (a Mozilla/Firefox version is forthcoming they say). You probably already have all your favourite Oracle sites bookmarked but if you are relatively new to the Oracle world then installing this is going to be a good start to finding you way to Oracle's web presence. Even if you do have your own bookmarks there is no harm in having this available – IE allows you to chose not to display it all the time in any case, via the View->Toolbars menu option.

I installed it and for some reason my laptop got confused and thinks the toolbar option called Oracle is the Google toolbar and the toolbar option named Google is the Oracle toolbar! I am pretty sure this is just my aging Toshiba playing games with me but I would like to hear if anybody else experiences the same issue.

The Oracle toolbar can be downloaded and installed from:

http://www.oracle.com/technology/toolbar/install/index.html

Tim Onions is an independent database consultant with over 15 years of experience with Oracle databases, with over 6 years experience of building and supporting high performance 7x24 mission critical systems. He specializes in the architecture, application design and database design of high performance systems, as well as tuning, optimisation techniques and technical management. Over the last 24 months he has led a number of Oracle data migration projects moving from Oracle8i and 9i to Oracle10g RAC on Windows 2003 and Linux platforms. Tim can be contacted via Tim.Onions@TOdC.co.uk. Further articles, hints, tips and scripts are available for download from the website http://www.TOdC.co.uk.