An introduction to GNU screen

Terminal manager with vt100/ANSI terminal emulation

Malcolm Herbert
mjch@mjch.net
2013-02-27

What **screen** is

- **screen** is a text-only window manager which multiplexes a single terminal between several processes (most often interactive shells)
- Each terminal window provides VT100-like emulation to the applications run under it and **screen** itself supports any terminal defined in **termcap** or **terminfo** for the display
- Sessions may be detached and re-attached without changing the state of the clients
- The history buffer provides cut-and-paste access to allow moving text regions between windows

What **screen** isn't

screen understands text and terminals and that's about it.

- Doesn't know anything about X11 windows
- Doesn't know anything about X11 mice
- Limited support for multi-character languages

Other **screen**-like things

The following commands do similar **screen**-like things:

- **window** (4.3BSD, circa 1993)

  Allows arbitrary text panes, a text-based wm
- **splitvt** (circa 1995)

  Basic two-pane terminal splitter.
- **twin** (Linux, circa 2009)

  Seems to be similar in focus to **window**
- **tmux** (OpenBSD, circa 2009)

  About on par with **screen** for features now

Getting started

From the prompt, simply type

```
screen
```

This will create a new **screen** session, open a new window and then spawn a fresh shell.

At this point, if you exit the shell, **screen** will also close.

Basic **screen** commands

Once in screen, there are many two-key sequences which control **screen** behaviour

create a window:	**ctrl-a c**
kill a window:	**ctrl-a k**
show windows:	**ctrl-a w**
show help list:	**ctrl-a ?**
switch to window *n*:	**ctrl-a *n***

Some commands are available from the shell

Creating another window

There are several ways to create windows:

- from your **screen**-managed shell:

```
screen
```

- similarly, to spawn an application:

```
screen vi foo
```

More basic commands

title a window:	**ctrl-a A *name***
prompt for name:	**ctrl-a ' *name***
select from list:	**ctrl-a ”**
switch to next:	**ctrl-a n**
switch to prev:	**ctrl-a p**
toggle recent:	**ctrl-a ctrl-a**

More basic commands

screen automatically determines the size of your terminal, but sometimes needs help:

Fit to terminal: **ctrl-a F**

The X11 **resize** command is also useful before you start a session:

```
maja[~] 6v>: resize -u
COLUMNS=152;
LINES=47;
export COLUMNS LINES;
```

Extended features

The previous commands are the essential ones which will get you going in the short term - it gets really interesting from here:

- Scrollback buffer
- Cut and paste
- Suspend/resume
- Multi-head capability
- Window monitoring/logging

... and more

Scrollback buffer

screen conveniently keeps session history:

enter history: **ctrl-a [**
exit history: ***ESC***

Once in the buffer, use **vi** key bindings:

movement: **i**, **j**, **k**, **l**, **ctrl-u**, **ctrl-d**, *etc*
search: ***/*** (fwd), **?** (rev), **n** (next)

Cut and paste

The buffer allows cut and paste:

Selecting text:

- Navigate to where you want to start
- Press **enter**
- Navigate to where you want to stop
- Press **enter** again

Screen will then exit the history buffer.

Cut and paste

Having selected a region, you can do this:

paste selection: **ctrl-a]**

... or you can copy the selected text to/from the temporary file `/tmp/screen-exchange`

dump to file: **ctrl-a >**
read from file: **ctrl-a <**
delete file: **ctrl-a =**

Advanced cut and paste

screen lets you join lines of selected text.

Whilst inside the history buffer and before you have finished a selection:

J cycle through:

- Join lines with no space
- Join lines with space
- Join lines with comma
- Leave as multiple lines

Advanced cut and paste

screen also allows you to select an arbitrary region of text which you can then join as above – most useful for constructing long lines of PIDs for **kill**, for example.

Whilst inside history buffer and before you have finished a selection:

set left margin: **c**
set right margin: **C**

Suspend and resume

screen is able to detach from the 'head' terminal and maintain the session whilst disconnected. This is a very very useful feature.

detach: **ctrl-a d**

To reattach, use the following command:

```
screen -r
```

Suspend and resume

Sometimes **screen** has more than one session open and needs help to work out which you want to open again:

```
maja[~] 2>: screen -r
There are several suitable screens on:
        1905.ttyp1.maja (Detached)
        2212.ttyp4.maja (Detached)
Type "screen [-d] -r [pid.]tty.host" to
resume one of them.
```

Suspend and resume

You may have a remote session you want to close and log out – use the 'power detach'

```
screen -D
```

or, from inside **screen** in multi-head mode:

power detach: **ctrl-a D**

Use these to drag sessions to your terminal.

Multi-head mode

In essence, multi-head mode is just being able to 'resume' a session whilst staying logged in on the other terminal:

```
screen -x
```

Both sessions are now completely available from either 'head' terminal ... including viewing the same one at the same time.

See the manual for more option combos.

Monitoring modes

screen can listen in on your sessions and raise an alert when it detects silence and/or activity – this can be useful if you don't want to watch a log but do want to know if it stops:

monitor activity: **ctrl-a M**
monitor silence: **ctrl-a _**

These options toggle on/off and both may be active at the same time.

Screen colon prompt

All **screen** commands are available from the colon prompt, even if not bound to keys.

colon prompt: **ctrl-a :**

Useful commands:

visual bell on: `vbell on`
visual bell off: `vbell off`
new shell env: `setenv BLAH foo`

Multiple windows

screen can display multiple windows at once by splitting the terminal horizontally into multiple regions. Each individual window is completely independent of the others.

split screen:	**ctrl-a S**
focus on next:	**ctrl-a *TAB***
recombine:	**ctrl-a Q**

Sadly, detach/resume forces a recombine.

Multiple windows

Window regions can be resized arbitrarily or set so that all share equally. From the colon prompt, resize the window in focus:

resize absolute:	`resize` *n*
resize relative:	`resize +`*n*
	`resize -`*n*
equal sizes:	`resize =`

Configuring screen

screen is configured from `$HOME/.screenrc` by default but you can include other files for better management of more complex configuration.

An example directory layout:

```
.screenrc -> .screen/default/client

.screen/default/client
.screen/default/logging
.screen/default/meta
.screen/default/pconsole
.screen/eeny/client
.screen/eeny/pconsole
.screen/exchange
.screen/logs/
```

Configuring screen

For example, `~/.screenrc` contains:

```
# default-client config

source $HOME/.screen/default/common

#password xxxxxxxxx
```

Common screen configuration

~/.screen/default/common contains:

```
startup_message off
autodetach on

vbell off
vbell_msg "BELL"
bell "Bell in main window %"

defscrollback 10000
defutf8 on

zombie kc
nethack on
bufferfile $HOME/.screen/exchange
deflogin off
defautonuke on

fit
```

Common screen configuration

`~/.screen/default/common` (cont.):

```
bind ! select 11
bind @ select 12
bind \# select 13
bind $ select 14
bind % select 15
bind \136 select 16
bind & select 17
bind * select 18
bind ( select 19
bind ) select 10
```

Screen shell-script management

~/bin/screen-client:

```
#!/bin/sh

unset STY

host=`uname -n | sed -e 's/\..*//'`
level=client
name=${host}/${level}
sess=${host}-${level}
config=$HOME/.screen

if [! -r ${config}/${name}]; then name=default/${level}; fi

exist=`screen -ls 2>&1 | cut -f2 | sed -ne "\#$sess#p" |
head -1`

if [ ! -z "$exist" ] ; then
  exec screen -U -m -xr ${exist} -c ${config}/${name} "$@"
else
  exec screen -U -m -S ${sess} -c ${config}/${name} "$@"
fi
```

Per-host config

The previous scripts allow for per-host configuration by default.

For example:

`~/.screen/eeny/client:`

```
source $HOME/.screen/default/client

screen -t eeny ssh eeny.virtual
screen -t meeny ssh meeny.virtual
screen -t miny ssh miny.virtual
screen -t mo ssh mo.virtual
```

Configuring screen

Some people like having a fixed status line visible at the bottom of the terminal all the time:

```
hardstatus on
hardstatus lastline
hardstatus string "%w"
```

There are many possible format strings, see the manual for more.

Shell magic

Like many vt100 emulators, **screen** has escape sequences to allow the application to change titles of windows:

Title string: *ESC*k*string**ESC*\ *CR*

eg, **tcsh**:
```
alias precmd \
        'echo -n "^[k$host^[\ ^M"'
```

Logging sessions

Like **script**, **screen** can allow you to capture your complete session for later review.

Apart from logging keys and responses, **screen** can optionally add date stamps and other items.

In these examples, session logs will be written to `~/.screen/logs/`

Configuring for logging

~/.screen/default/logging:

```
# default-logging config

source $HOME/.screen/default/client

backtick 9 1 1 date '+%Y%m%d%H%M%S'

logfile $HOME/.screen/logs/%9`-%n-%t.log
logfile flush 5

logtstamp on
logtstamp after 120
logtstamp string "[[%Y-%m-%d %0c:%s %n %t]]"

deflog on
```

Configuring for logging

~/bin/screen-logging:

```
#!/bin/sh

unset STY

host=`uname -n | sed -e 's/\..*//'`
level=logging
name=${host}/${level}
sess=${host}-${level}
config=$HOME/.screen

if [! -r ${config}/${name}]; then name=default/${level}; fi

exist=`screen -ls 2>&1 | cut -f2 | sed -ne "\#$sess#p" |
head -1`

if [ ! -z "$exist" ] ; then
  exec screen -U -m -xr ${exist} -c ${config}/${name} "$@"
else
  exec screen -U -m -S ${sess} -c ${config}/${name} "$@"
fi
```

Cluster sessions

Although **screen** is good at managing single sessions, with larger groups of hosts it can be very useful to have parallel sessions open to run the same commands in sync.

Traditional tools such as **cssh** require X11 to run and don't allow nearly as many features, whilst **puppet** et al are non-interactive.

Thankfully, **screen** and **pconsole** can be used to multiplex keystrokes to multiple windows whilst keeping **screen** features.

Configuring for clusters

~/.screen/default/pconsole:

```
# default/pconsole config

source $HOME/.screen/default/pconsole-head

source $HOME/.screen/default/pconsole-tail
```

Configuring for clusters

~/.screen/default/pconsole-head:

```
# default/pconsole config

source $HOME/.screen/default/logging

deflogin on
deflog off

screen -t pconsole sh -c "sleep 3; $HOME/bin/pconsole-
attach.ksh"

deflog on
```

Configuring for clusters

~/.screen/default/pconsole-tail:

```
select 0
split
resize 1
focus bottom
select 1
focus top
```

Configuring for clusters

~/bin/pconsole-attach.ksh:

```
#!/bin/ksh -x

read "pause?Hit enter to start pconsole: "

self=`tty | sed -e 's#/dev/##'`
parent=`who | grep $self | sed -e 's/:S.[0-9].*//' -e
's/.*://'`

devs=""
who | sed -ne "\@$parent:S.[0-9]*@p" |&
while read -p user tty mon mday time from ; do
  if [ "$tty" != "$self" ] ; then
    devs="$devs /dev/$tty"
  fi
done

sudo pconsole $devs
```

Configuring for clusters

~/bin/screen-pconsole:

```
#!/bin/sh

unset STY

host=`uname -n | sed -e 's/\..*//'`
level=pconsole
name=${host}/${level}
sess=${host}-${level}
config=$HOME/.screen

if [! -r ${config}/${name}]; then name=default/${level}; fi

exist=`screen -ls 2>&1 | cut -f2 | sed -ne "\#$sess#p" |
head -1`

if [ ! -z "$exist" ] ; then
  exec screen -U -m -xr ${exist} -c ${config}/${name} "$@"
else
  exec screen -U -m -S ${sess} -c ${config}/${name} "$@"
fi
```

Per-cluster config

Similarly for the per-host config, we can set up a **screen** config to open all the cluster nodes at once. For example:

~/.screen/eeny/pconsole:

```
source $HOME/.screen/default/pconsole-head

screen -t eeny ssh eeny.virtual
screen -t meeny ssh meeny.virtual
screen -t miny ssh miny.virtual
screen -t mo ssh mo.virtual

source $HOME/.screen/default/pconsole-tail
```

Meta-screen sessions

Naturally, with more than one configuration or several sets of isolated sessions on various remote hosts you would want to manage them in turn with **screen** itself.

This would allow cut and paste between remote **screen** sessions using a meta-session. Multi-head ability allows swapping sets of **screen** sessions between two terminal windows, for example ...

Configuring meta-screen

By default the 'escape sequence' is **ctrl-a**, but can be changed to **ctrl-q** with:

```
escape ^Qq
```

Line discipline could ruin your day – use **stty** to turn off special action from your shell:

```
stty start ^-
stty stop ^-
```

Configuring meta-screen

~/.screen/default/meta:

```
# default-meta config

source $HOME/.screen/default/common

#password xxxxxxxxxx

escape ^Qq
```

Meta-screen shell-script

~/bin/screen-client:

```
#!/bin/sh

unset STY

host=`uname -n | sed -e 's/\..*//'`
level=meta
name=${host}/${level}
sess=${host}-${level}
config=$HOME/.screen

if [! -r ${config}/${name}]; then name=default/${level}; fi

exist=`screen -ls 2>&1 | cut -f2 | sed -ne "\#$sess#p" |
head -1`

if [ ! -z "$exist" ] ; then
  exec screen -U -m -xr ${exist} -c ${config}/${name} "$@"
else
  exec screen -U -m -S ${sess} -c ${config}/${name} "$@"
fi
```

... and more

That's really about all the **screen** features that I regularly use, but it also can do:

- Manage sessions/windows with user ACLs
- Start **screen** sessions in the background
- Use a lock program for pretty screen saver
- Grab system console
- Very hairy pipe/fd magic with exec (run **less** on command output after start!?)

Wishlist

There are always things that our favourite tools don't to that we really would like them to – **screen** is no different:

- Recovering commands used to initiate a window
- Configurable digraph table and better multi-lingual support
- Better client-server model (**tmux** has this)
- others?

More information

Seriously, I've only just scratched the surface here – the manual entry is huge (around 30 pages, from memory) and is worth a good look. I regularly re-read it and usually find something useful I didn't already know.

man screen

http://www.gnu.org/software/screen/

Comments, questions?

An introduction to GNU screen

Terminal manager with vt100/ANSI terminal emulation

Malcolm Herbert
mjch@mjch.net
2013-02-27

Git repository of talk materials and configs:
http://deimos.ergonaut.org/repository/talks-screen

License