A Simple Guide to Screen

Simon Symeonidis

Fri Apr 18 2014

Introduction

Screen is awesome. I've been using it for a while now. Basically it allows you to have a similar experience as that of a graphical desktop, where you can minimize different sessions, but in terminal mode.

Screen is pretty useful too if you need to run different sessions with a specific context, and want to resume monitoring it (manually) later on. This is a pretty nice way to group tasks involving different jobs as well, especially on a server setting.

For example if for some reason you want to run different *minecraft* processes with different contexts on the same server, you can have different screens to be used to show logging information and other stuff, for each. For example, this is what the setup would look like:

```
5960.database-diagnostics (Detached)
5941.minecraft-jail (Detached)
5893.minecraft-with-mods  (Detached)
5877.minecraft-simple (Detached)
```

Each line on the above denotes a particular session. Each session can have more screens attached to them. So in essence you could totally have something like this as a setting for one of the screens:

```
minecraft-jail
  - logging
  - misc
  - scripted-tasks
  - server
```

It is also possible to nest more complex screens inside the lower levels as to have something like this as well:

```
minecraft-jail
  - logging
    - critical
    - medium
    - low
  - misc
  - scripted-tasks
  - server
```

Most of the time you probably won't need to do something like that however.

Forenote

We'll be using the notation usually found in Linux man-pages for the keyboard shortcuts. That is, we note `ctrl+a` as `C-a`. If you combine *C-a*, and a key must follow without having the control key combined, we denote this as *C-a g* where `g` is the next key (the actual 'g' key) that needs to be pressed.

Getting Started

To run a screen you need to type in the following:

```
$ screen
```

This would bring you to a session you can detach yourself from. You might notice that the screen flashes instead of beeping on weird keystrokes. To switch from flashing to beeping (which hopefully is disabled), you can simply press `C-a, C-g`.

Detaching

Execute the command `screen`, if you haven't already.

```
$ screen
```

Now say we're connected to a server, and we'd like to leave something running in the background, in which we'd eventually like to get back to, and interact with. For the sake of this example, let it be this simple bash line:

```
$ for (( ;; )) do sleep 3s; echo "ZzzzzZZZzzzz ..."; done
```

We can enter the following sequence: `C-a, C-d`, and detach from the session. We're back at the terminal where we started from!

Naming Screens

So far you have started `screen` by typing in the command. If you have many screens without a name, you're eventually going to have something like this to sort out:

```
$ screen -ls
There are screens on:

2303.pts-3.aeolus (Detached)
2196.pts-3.aeolus (Detached)
2185.pts-3.aeolus (Detached)
2172.pts-3.aeolus (Detached)
2164.pts-3.aeolus (Detached)
```

If you wanted to reattach to something, you would need to specify its id. The following command would attach to the first screen:

```
$ screen -r 2303
```

And this would attach you to the second screen...

```
$ screen -r 2196
```

... Rocket Science!

If you can remember numbers, that's cool. But not everyone is like you for the better or worse. There is always the option (which I highly recommend, you monster) of using `session names`. You simply need to use the `-S` flag.

```
$ screen -S derpy-screen
```

More than One Window

Screens are useful. But the feature which will raise your life expectancy in a software environment is the fact that you can have multiple windows in a screen. You heard me right.

You can either create these via the command line, or manually. Usually you'll be doing this manually, unless you want to automate something - we will talk about this later.

Enter a screen session:

```
$ screen -S awesome
```

Once you're inside the new screen, perform these keystrokes:

C-a c

This will create a new window. So now, you should actually have two windows inside a screen. To view your windows you will need to perform another set of keystrokes:

C-a "

(Yes, double inverted commas). You can use the arrow keys, or the 'j', 'k' keys in order to choose a particular window. Press 'enter' when you have made up your mind.

Naming Windows

Eventually, you will need to name your windows. If you want to name a window you are currently in, you just need to enter the following keystrokes:

C-a A

This will open a small edit box at the end of the terminal where you can enter the desired name (eg: bin, src, test, logging, windows 95 etc).

The Screen that *is* and *is not*

In some cases, the bad and ugly happens, and a screen session is disrupted from some chaos. For example your coworker tripping over the wires of your computer, and severing the connection between you and the server. Though funny, this can sometimes mark the screen as `Attached` when you're trying to reconnect, and block you. If you're sure that something has probably gone wrong, you need to 'wipe' the screen. This is achieved by supplying the `-D` flag.

```
$ screen -D screen-name-or-id
```

After the successful `wipe` you can type the following command over your coworker's cries of anguish, to get back to work; much more important anyway:

```
$ screen -r screen-name-or-id
```

Spying On People

One of the nice features of screen is seen when using the `-x` flag (notice small 'x'). If someone is working on a current screen, and you want to spy on then you can totally use this option.

For example let's have Amy and Jon. Amy wants to find out what Jon's favorite linux command is so she can get him one for his birthday. Jon needs to first be in a screen session. Jon types in the following:

```
$ screen -S jons-session
```

Amy can now spy on Jon:

```
$ screen -x jons-session
```

And now, Amy can see everything.

Jokes aside, a nice use for this is could be if you've got a bunch of people you want to show a few things on the command line; for example like a tutorial. Just be careful.

Making Screens via Script

Finally you might end up finding screen useful, and want to exploit its use a little more. What I personally ended up needing is automating sessions on startup. We had a really weird server setup at a place that I worked, that would decide to reboot without notice, and randomly. So any session that was running would not be resumed.

So what you'd usually want is to have some script that would create these sessions for you automatically. The usual use case is to start some screen with a task, and detach it in th background. This may simply be achieved in the following manner:

```
$ screen -dmS myscreen htop
```

That will create a screen, with the name `myname` and run htop in it. The screen is also detached and you won't be seeing anything else on the command line after executing that.

Finally, the next question arises - what if I want to make a detached screen with multiple, named, tabs?

No problem. You just need to run these commands:

```
$ screen -dmS myscreen
$ screen -S myscreen -X screen -t checkls ls
$ screen -S myscreen -X screen -t monitorsys htop
# If you just want a screen with no job
$ screen -S myscreen -X screen -t spare
# And finally if you want to reattach to that screen
$ screen -r myscreen
```

On a side note, I found that many times when I was working with a RubyGem library, I would constantly create a screen with the same windows. For example I would have a window *src* where I would be doing all my coding. I would have another window called *spec* where I would be writing my tests. Another one would be *run* if I needed to run against some file and see how the software was behaving.

The repeated work got annoying really fast, and I came up with a bash script that somewhat made my life a little easier. Feel free to use and modify it.

```
#!/usr/bin/env bash
# Standard screen stuff that I do too many times

if [ -z "$1" ]
then
  echo "Usage: "
  echo "  mk.screen <scree-session-name>"
  exit
fi

screen -dmS $1
screen -S $1 -X screen -t src
screen -S $1 -X screen -t test
screen -S $1 -X screen -t tasks
screen -S $1 -X screen -t run
screen -S $1 -X screen -t bin
# You can totally start up sessions with commands; for example this one will
# start htop and you can monitor things like that :D
# screen -S $1 -X screen -t monitor -X htop
screen -r $1
```