Launching bash scripts at startup

Jump to bottom

Somahtr edited this page on Jul 25, 2013 · 1 revision

How to write a simple bash script

To write a simple bash script just open a new file in your favourite text editor, e.g.

```
nano simple_bash.sh
```

and write as a first line the heading

```
#! /bin/bash
```

This indicates that everything written from now on will be bash commands. All the commands in a shell script must be valid bash instructions (like launching applications, 'exit', 'logout', 'apt-get install', 'cd', and so on). You might in general want to make shell/bash scripts executable and add root rights. This can be done by typing

```
chmod a+x simple_bash.sh    //Make it executable
chmod 777 simple_bash.sh    //Give it root rights (saves you to write sudo every
time)
```

This is pretty much all there is to know about what is needed to make shell scripts work. The syntax of a single shell script may of course be very complex, but we won't go into much detail here.

Run programs at startup on the Pi

There are multiple ways of running programs at startup on the Raspberry Pi (or in any other Linux environment). The three most straightforward options are

- using update-rc.d command: this will allow you to launch any command BEFORE login. Note that if an invalid bash command is launched such that it does not give a valid exit error (I'm not sure which commands might do this, but they apparently exist) the whole booting process might crash and never even arrive at the login screen. Which means you will have to format and reinstall everything. So be careful what you make your startup do!

- including new instructions in ~/.bashrc: this will launch new instructions every time an interactive shell is opened. Not particularly dangerous, and should be easy to fix by logging in with a different account and modifying it from there. However, just a word of advice: do NOT write any command that requires opening a terminal to execute, as this will give you an endless cascade of terminals opening and running .bashrc, and so on and so on.

- including new instructions in ~/.profile: in my opinion this is the best method to launch applications you might be interested in using from startup, as anything included here will be launched from the first login shell ONLY (no cascading), immediately after you have logged in to your account.

update-rc.d

udpate-rc.d is a powerful command, but it needs to be used with caution. Now, assuming you know what you are doing, the way to include shell scripts in your startup 'line of fire' is:

- make your script executable

- copy your script to the /etc/init.d/ folder with

  ```
  sudo cp /path_to_script/script.sh /etc/init.d/
  ```

- invoke update-rc.d with

  ```
  sudo update-rc.d script.sh defaults
  ```

This will add your script as the last thing to be run before login. There are ways of tweaking around with when exactly during startup you might want to launch your script (see man update-rc.d 8), but since in general you will probably need everything else that is launched by default to be operational already, we won't bother about that here.

Should you want to remove the link to your shell script from the startup list, you will have to

- remove the shell script from the init.d folder FIRST

  ```
  sudo rm /etc/init.d/script.sh
  ```

- and invoke

  ```
  sudo update-rc.d script.sh remove
  ```

.bashrc and .profile

First, I would suggest to read the descriptions of these files from their commented headings. You will for example find out that, should a ~/.bash_profile or a ~/.bash_login file exist, ~/.profile will not be executed. These two .bash_* files don't seem to be present by default, so we will use the .profile file. However, should any of those be present, you can just add your instructions in those instead.

Editing these files is very simple, just open them with a text editor and, from the last line forwards, just write what you want to launch after login. For example, in .profile:

```
# ~/.profile: executed by the command interpreter for login shells.
.
.
.

#Execute my script
sh /home/pi/script.sh
```

or similar. The same is true for .bashrc, only remember that this will be launched every time an interactive shell session is launched.

Example: embed a terminal in the desktop!

First things first: we will need to install two programs:

- xfce4-terminal is a terminal that has a different preferences file from the default terminal (so that all the settings you make for this terminal will not modify the appearance/behaviour of your normal terminal)
- wmctrl is a window management utility that allows to modify window characteristics from terminal.

As usual, just run

```
sudo apt-get install xfce4-terminal wmctrl
```

and you will have the programs. Then open a session of the xfce4-terminal and modify the settings to:

- **General > Title > Dinamically-set title** to 'Replaces initial title'
- **General > Scrolling > Scrollbar is**: to 'Disabled'
- **Appearance > Background > Transparent background** and slide the bar all the way to the left (value 0.00)

You can fidget around with the other settings if you want, but these will make your terminal look nice.

Then write a shellscript (named for example **.embedded_desktop.sh**) with the following content:

```
#! /bin/bash

export DISPLAY=:0
xfce4-terminal --title=descon1 --hide-menubar --hide-borders --hide-toolbars &
xfce4-terminal --title=descon2 --hide-menubar --hide-borders --hide-toolbars &
sleep 5 &&
wmctrl -F -r descon1 -e 0,40,50,1200,870 && wmctrl -F -r descon1 -b
add,sticky,below && wmctrl -F -r descon1 -b add,skip_pager,skip_taskbar
wmctrl -F -r descon2 -t 1 && wmctrl -F -r descon2 -e 0,40,50,1200,870 && wmctrl
-F -r descon2 -b add,sticky,below && wmctrl -F -r descon2 -b
add,skip_pager,skip_taskbar
```

You can adjust the size and position of the terminal to suit your needs. These are given by the first invocation of wmctrl. Note also that openbox used in Raspbian does NOT support sticky configuration, so this shell script creates a different terminal per desktop (sticky just means that your terminal will appear in all desktops). If you are working under a graphical environment that does support the 'stickyness' feature, then remove the second terminal and the associated wmctrl instructions. For more information on how wmctrl works give a quick read to

```
man wmctrl
```

as it contains quite a few other options that you may want to try. You can also change the name of the window if you wish, although it doesn't really matter as it does not appear anywhere.

Now add the following lines at the end of the ~/.profile file

```
.
.
.
# Start graphic session automatically (and wait for it to start)
startx &
sleep 15 &&

# Execute script for embedded terminals
sh -x /home/pi/.embedded_terminal.sh
```

Note that this will start the graphical environment immediately after login. It is necessary to do so, as the xfce4-terminal requires a graphical environment to run. The sleep commands are just to give the graphical enviroment time to load. Now, after reboot, you should have terminals embedded in your desktop!

Find a Page…

Clone this wiki locally

```
https://github.com/OpenLabTools/OpenLabTools.wiki.git
```