

I thought it could be useful to share my experience installing Ubuntu Linux on my rooted TV thanks to SamyGo. With some help it could become a very good set up for our TV's ^^.

What is this?

This is a project to install a "full" Ubuntu Linux system that natively runs on your rooted Samsung Smart TV. But since I don't want to get anyone's hopes up, note that the current state of the project has some major drawbacks that makes it difficult to use, namely a very low performance so it might not worth it for you.

What do I need?

- A Samsung Smart TV (LOL). I'll be using my 46ES8000.
- Root access + telnet + ftp. You can get this using SamyGo... or creating your own hook in libSkype.so to get them (jk, don't waste your time like that :P).
- A second computer with telnet and ftp clients. I'll be using my Windows laptop, with PuTTY and winSPC. Since it's a laptop I can work right in front of the TV.
- An USB storage device (HDD, pendrive, etc...) with at least 2 GB. I would recommend a 4 GB one. I'll be using a Kingston Datatravel 3.0 with 64 GB.

Things to consider...

I didn't want this to be a risky project that could potentially kill my TV, that's why we won't be touching the TV's internal memory. This makes the project harder, but at least we are 100% safe ☺.

Also, note that I haven't been able to test all this in a different TV, I guess it should work the same (or even better in newer models), but I can't confirm it.

So let's a-go!

We are going to download Ubuntu Linux into our pendrive and configure a chroot environment that will use the tv's XServer (yes, the SmatHub runs in an XServer) to display our X11 desktop.

First, plug your pendrive in one of the TV's USB port. Then, it's time to format it. We are gonna use XFS, since it's the only fully supported Linux filesystem my TV's kernel support (and since I want this a safe procedure, I wouldn't like to flash a new kernel xD).

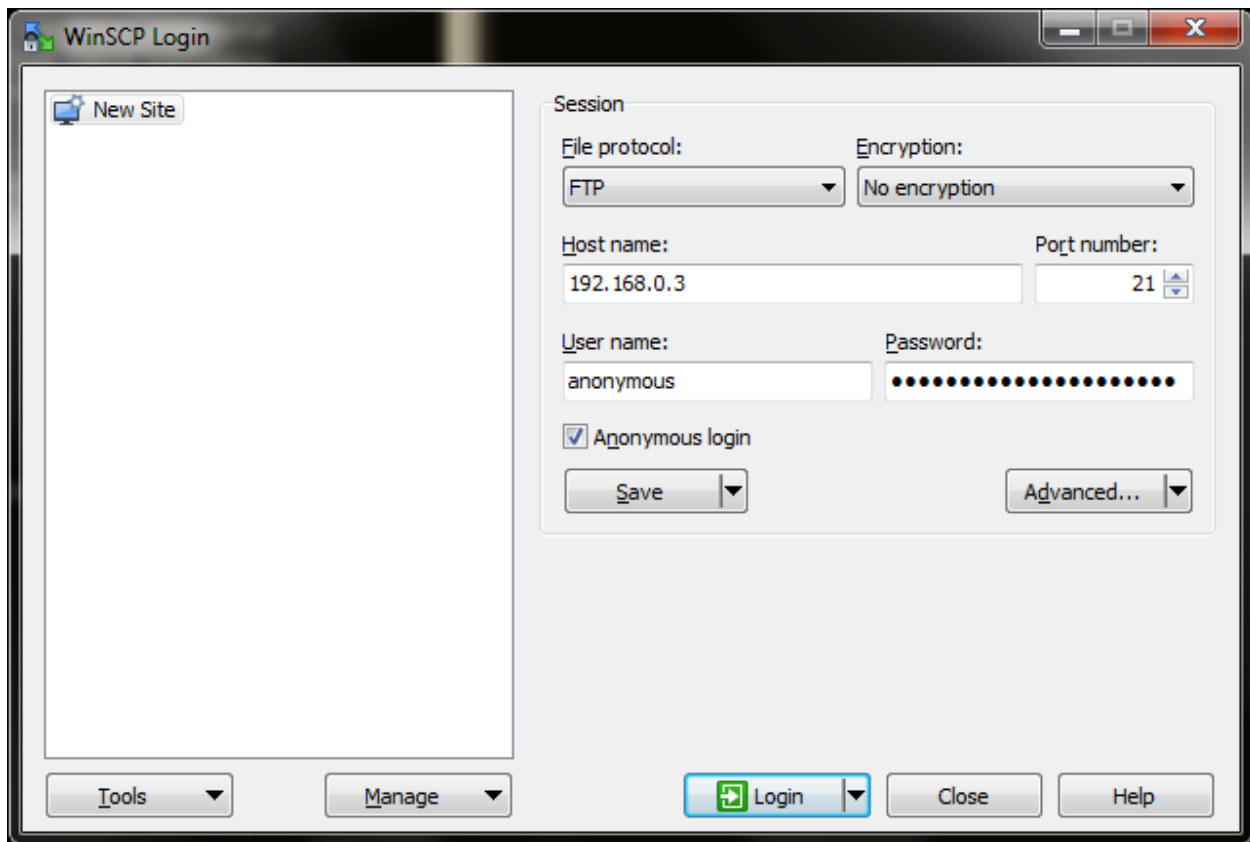
- You can check the pendrive linux device with:
 - o VDLinux#> df -h
 - o (in my case it was /dev/sda1)
- Then unmount it:
 - o VDLinux#> umount /dev/sda1
- Format the device, I'm using the -f option cos I already had a filesystem there:
 - o VDLinux#> mkfs.xfs /dev/sda1 -f
- Mount it where you want your Ubuntu installation. I'm gonna use /mnt/core:
 - o VDLinux#> mkdir /mnt/core/
 - o VDLinux#> mount /dev/sda1/ /mnt/core

Awesome, we have our storage media set up, now it's time to download a Linux filesystem. Any one should do it, but I used Ubuntu Core for convenience. Since my TV is an ARM device with no hard float support, I'm gonna use the last armel release available, which is Ubuntu 12.4 Precise Pangolin. You can get it from here:

<http://old-releases.ubuntu.com/releases/ubuntu-core/releases/12.04/release/ubuntu-core-12.04-core-armel.tar.gz>

Now send the downloaded archive to our pendrive using the ftp client:

- Log anonymously in the TV's ftp server:



- Go to /mnt/core and drop the tar archive there:

/mnt/core				
Name	Ext	...	Changed	Rights
..				
ubuntu-core-12.04-core-ar...		..	1/1/2014 12:42 AM	rw-r--r--

- Let's decompress the filesystem archive, then we can delete it:
 - o VDLinux#> cd /mnt/core
 - o VDLinux#> tar -xzf ubuntu-core-12.04-core-armel.tar.gz
 - o VDLinux#> rm ubuntu-core-12.04-core-armel.tar.gz

The result is a full Linux filesystem in our pendrive! Now it's time to set it up.

Usually, when configuring a chroot environment, one would use the **mount -bind** to share some resources between the host filesystem (in our case provided by SamyGo) and the guest filesystem (the one we just decompressed).

However, the `-bind` option simply does not work in my TV, I don't know why but since the binaries seem to be OK, I can only think the binding feature is missing at kernel level (and we don't want to flash a new kernel so...), we can get around this simply copy the needed resources from the host filesystem. That's what we're gonna do to get network support under Ubuntu:

- To get network support:
 - `VDLinux#> cp /etc/resolv.conf /mnt/core/etc/resolv.conf`
 - `VDLinux#> cp /etc/hosts /mnt/core/etc/hosts`
 - `VDLinux#> cp /etc/HOSTNAME /mnt/core/etc/hostname`

At this point we can enter Ubuntu typing:

- `VDLinux#> chroot /mnt/core /bin/bash`

Welcome to Ubuntu! One interesting thing to avoid future errors is to set the host name:

- `root@(none):/# hostname localhost`

Note that the hostname won't be update until the next time you log in, for the time being it'll stay as (none), but that's fine 😊.

To get more features in the Ubuntu chrooted environment, let's mount some virtual filesystems: these are shared with the host filesystem:

- `root@(none):/# mount -t sysfs sysfs /sys`
- `root@(none):/# mount -t proc proc /proc`
- `root@(none):/# mount -t usbfs none /proc/bus/usb`
- `root@(none):/# mount -t devpts devpts /dev/pts`

Just like that, let's set some environment variables we're gonna need:

- `export LD_LIBRARY_PATH=""`
- `export HOME="/root"`
- `export TMPDIR="/tmp"`
- `export FONTCONFIG_FILE="/etc/fonts/fonts.conf"`
- `export FONTCONFIG_PATH="/etc/fonts"`
- `export PANGO_RC_FILE="/usr/lib/arm-linux-gnueabi/pango/pangorc"`

Since we don't have bind support, we cannot have the host `/dev` directory, but just like we did with those network configuration files, we can reproduce device nodes in our chroot.

I'm just gonna create the framebuffer for now, but note that in the future, to get support for all the TV devices, we should create all device nodes present in the original `/dev` directory. For this, we are gonna need major and minor device numbers, the following link may help you:

<https://www.kernel.org/doc/Documentation/devices.txt>

In general, you can just check these straight from the original /dev using:

- VDLinux#> ls -Rla /dev

Even though we could create a script to make all those node automatically, the framebuffer is enough for now:

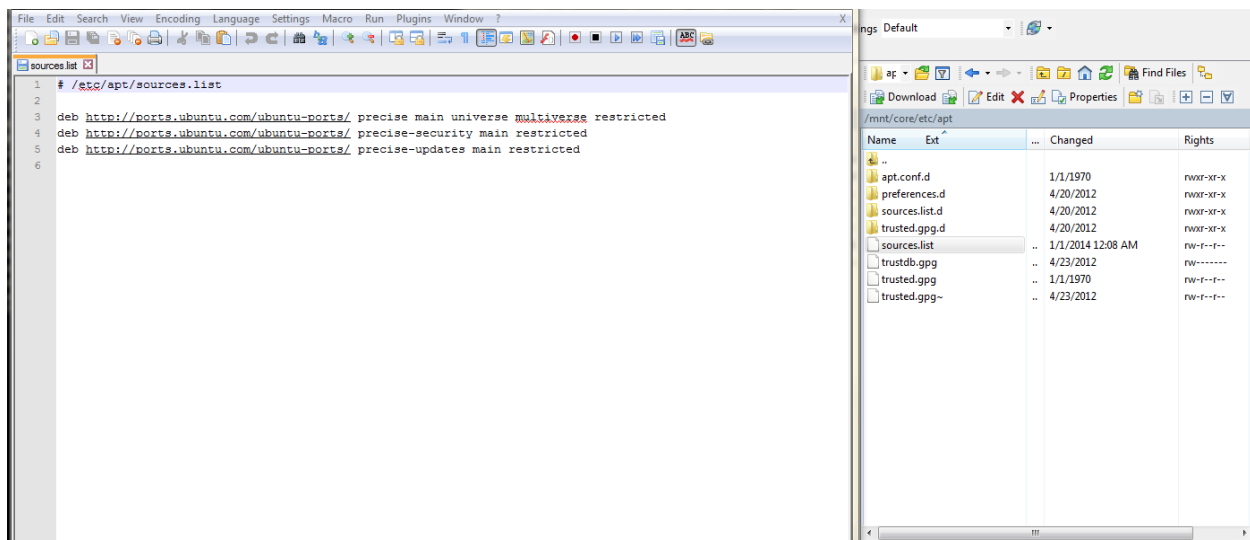
- root@(none):/# mknod /dev/fbo c 29 0

Another interesting one could be the mouse (we're not gonna use it for now, though):

- root@(none):/# mkdir /dev/input
- root@(none):/# mknod /dev/input/mice c 13 63

The TV's framebuffer is pretty interesting: by default, it's a 32 bit colorbuffer that uses the alpha channel not only for opacity, but also as a flag to show the TV picture. This is what the TV uses for PIP. We could exploit this to capture TV screens, or record video straight from the telnet session, as well as injecting images in our TV. But, as I said, it's better if we leave that to the XServer.

Let's continue with the Ubuntu set up. It's now time to install some new software! By default, Ubuntu Core includes just a few package repositories. Since those may not be enough, we can go and edit the file **/etc/apt/sources.list** to add some extra repositories. To make things faster, I use WinSCP for this. My sources.list file looks like this after edition:



Make sure you save the changes. Now we can update our package lists in Ubuntu, this will also work as an internet access test :P:

- root@(none):/# apt-get update

Now we can install some interesting package, the coolest one will be the desktop itself, but let's go step by step.

With this two lines you'll avoid future upstart-related errors:

- `dpkg-divert --local --rename --add /sbin/initctl`
- `ln -s /bin/true /sbin/initctl`

Let's install two supporting packages:

- `root@(none):/# apt-get install dialog apt-utils`

Let's install some desktop-related packages, this will take some time:

- `root@(none):/# apt-get install vim.tiny sudo ssh net-tools ethtool wireless-tools network-manager iputils-ping rsyslog alsa-utils`

Finally, it's time to install our desktop environment, here you can choose between a large variety and flavors, I've tested both the original Gnome (now Unity), and the Lightweight X11 Desktop Environment.

I can tell Unity works better in my case so let's go for it:


- `root@(none):/# apt-get install ubuntu-desktop 2> /error.log`

The installation of the desktop may take more than an hour, it will install all gnome tools (text editor, graphic terminal, calculators, some game...), drivers, make some configurations, etc... This is why I redirect the error stream to a file, so I can check if something went wrong more easily.


Remember to delete the error log if everything went right:

- `root@(none):/# rm /error.log`

Now that we have our desktop installed, we are gonna make a quick change so the Pango graphic library works alright. For this we need the original Pango modules and configuration files from the original filesystem. Those are present in `/mtd_exe/Runtime/pango`:

/mtd_exe/Runtime/pango				
Name	Ext	...	Changed	Rights
				
1.6.0			9/11/2013	rwXr-Xr-X
pango.modules		..	9/11/2013	rwXr-Xr-X
pangorc		..	9/11/2013	rwXr-Xr-X
pangox.aliases		..	9/11/2013	rwXr-Xr-X

So let's copy everything in our chroot's Pango directory, which is `/usr/lib/arm-linux-gnueabi/pango`:

/mnt/core/usr/lib/arm-linux-gnueabi/pango				
Name	Ext	...	Changed	Rights
				
1.6.0			1/1/2014 12:15 AM	rwXr-Xr-X
pango.modules		..	1/1/1970	rw-r--r--
pangorc		..	1/1/1970	rw-r--r--
pangox.aliases		..	1/1/1970	rw-r--r--

However, two of those files, **pango.modules**, and **pangorc** include references using absolute file paths from the original filesystem, we need to edit them so the paths match our chroot's directory structure. Here is how my file look after edition, you can just use the automatic replacement feature of your text editor :P:

Pangorc

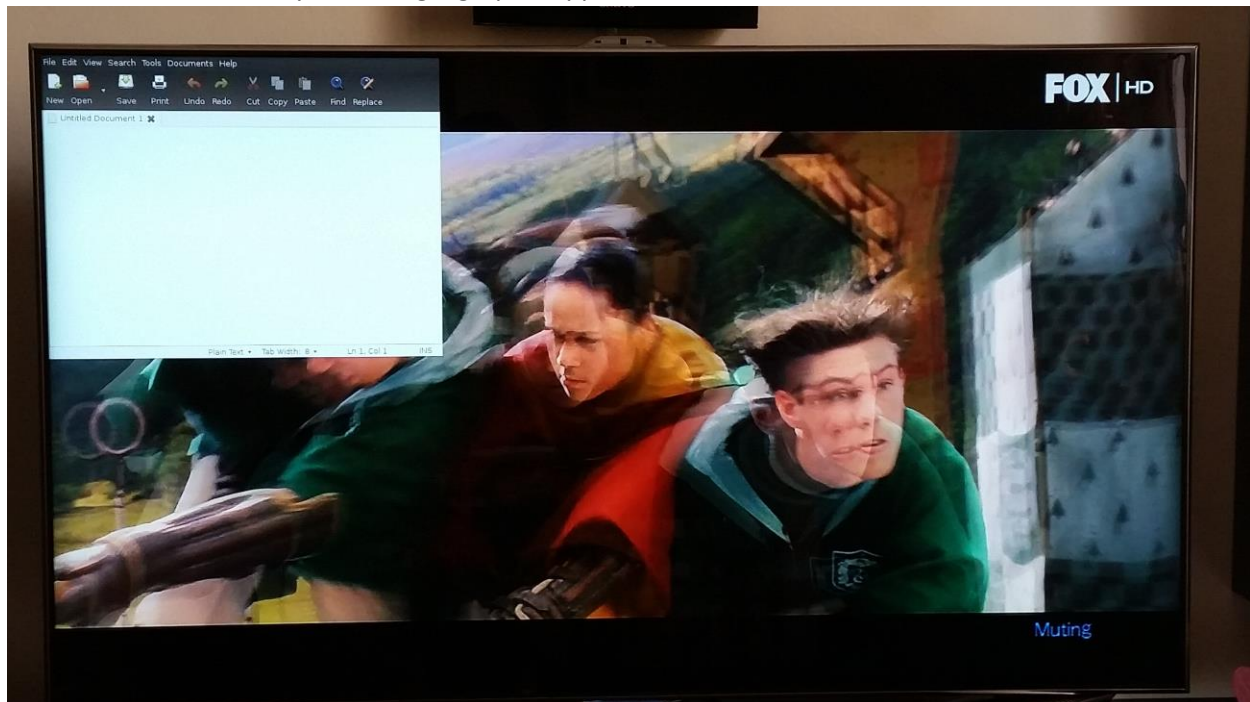
```
1 #
2 # pangorc file for uninstalled operation.
3 # We set the path as ../modules, such that it works from any of
4 # top level build subdirs.
5 #
6
7 [Pango]
8 ModuleFiles = /usr/lib/arm-linux-gnueabi/pango/pango.modules
9
```

Pango.modules

```
1 # Pango Modules file
2 # Automatically generated file, do not edit
3 #
4 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-arabic-fc.so ArabicScriptEngineFc PangoEngineShape PangoRenderFc arabic:*
5 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-arabic-lang.so ArabicScriptEngineLang PangoEngineLang PangoRenderNone arabic:*
6 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-basic-fc.so BasicScriptEngineFc PangoEngineShape PangoRenderFc armenian:* cherokee:* coptic:* cuxillic:* deseret:* ethiopic:*
7 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-hangul-fc.so HangulScriptEngineFc PangoEngineShape PangoRenderFc hangul:*
8 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-hebrew-fc.so HebrewScriptEngineFc PangoEngineShape PangoRenderFc hebrew:*
9 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-fc.so devaScriptEngineFc PangoEngineShape PangoRenderFc devanagari:*
10 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-fc.so bengScriptEngineFc PangoEngineShape PangoRenderFc bengali:*
11 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-fc.so guruScriptEngineFc PangoEngineShape PangoRenderFc gurmukhi:*
12 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-fc.so gujrScriptEngineFc PangoEngineShape PangoRenderFc gujarati:*
13 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-fc.so oryaScriptEngineFc PangoEngineShape PangoRenderFc oia:*
14 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-fc.so tam1ScriptEngineFc PangoEngineShape PangoRenderFc tamil:*
15 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-fc.so teluScriptEngineFc PangoEngineShape PangoRenderFc telugu:*
16 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-fc.so kndaScriptEngineFc PangoEngineShape PangoRenderFc kannada:*
17 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-fc.so mlymScriptEngineFc PangoEngineShape PangoRenderFc malayalam:*
18 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-fc.so sinnScriptEngineFc PangoEngineShape PangoRenderFc sinhala:*
19 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-lang.so devaIndicScriptEngineLang PangoEngineLang PangoRenderNone devanagari:*
20 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-lang.so bengIndicScriptEngineLang PangoEngineLang PangoRenderNone bengali:*
21 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-lang.so guruIndicScriptEngineLang PangoEngineLang PangoRenderNone gurmukhi:*
22 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-lang.so gujrIndicScriptEngineLang PangoEngineLang PangoRenderNone gujarati:*
23 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-lang.so oryaIndicScriptEngineLang PangoEngineLang PangoRenderNone oia:*
24 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-lang.so tam1IndicScriptEngineLang PangoEngineLang PangoRenderNone tamil:*
25 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-lang.so teluIndicScriptEngineLang PangoEngineLang PangoRenderNone telugu:*
26 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-lang.so kndaIndicScriptEngineLang PangoEngineLang PangoRenderNone kannada:*
27 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-lang.so mlymIndicScriptEngineLang PangoEngineLang PangoRenderNone malayalam:*
28 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-indic-lang.so sinnIndicScriptEngineLang PangoEngineLang PangoRenderNone sinhala:*
29 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-khmer-fc.so KhmerScriptEngineFc PangoEngineShape PangoRenderFc khmer:*
30 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-syriac-fc.so SyriacScriptEngineFc PangoEngineShape PangoRenderFc syriac:*
31 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-thai-fc.so ThaiScriptEngineFc PangoEngineShape PangoRenderFc thai:* lao:*
32 /usr/lib/arm-linux-gnueabi/pango/1.6.0/modules/pango-tibetan-fc.so TibetanScriptEngineFc PangoEngineShape PangoRenderFc tibetan:*
```

I know, I know, it says that being machine generated, we shouldn't edit it, but, we don't really have much choice XD.

And that's it ladies and gentlemen, we have a basic graphic set up of Ubuntu, that will use the TV's XServer, we can test it by launching a graphic application from Gnome such as... **Gedit!**



- o Yeah, Harry potter was on TV, never mind xD.

Now that we know we can connect to the TV's XServer, let's launch a full-fledged desktop environment. You can just close gedit using CTRL-C. However, running the desktop as root will give you some warnings (some apps will even give you errors, such as the Chromium web browser). So let's create an user:

```
root@(none):/# adduser sky
Adding user `sky' ...
Adding new group `sky' (1001) ...
Adding new user `sky' (1001) with group `sky' ...
Creating home directory `/home/sky' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for sky
Enter the new value, or press ENTER for the default
    Full Name []: SkyBladeCloud
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n] Y
root@(none):/# █
```

In order to not limit ourselves using the desktop environment, let's add the new user to the sudoers:

- o root@(none):/# adduser sky sudo

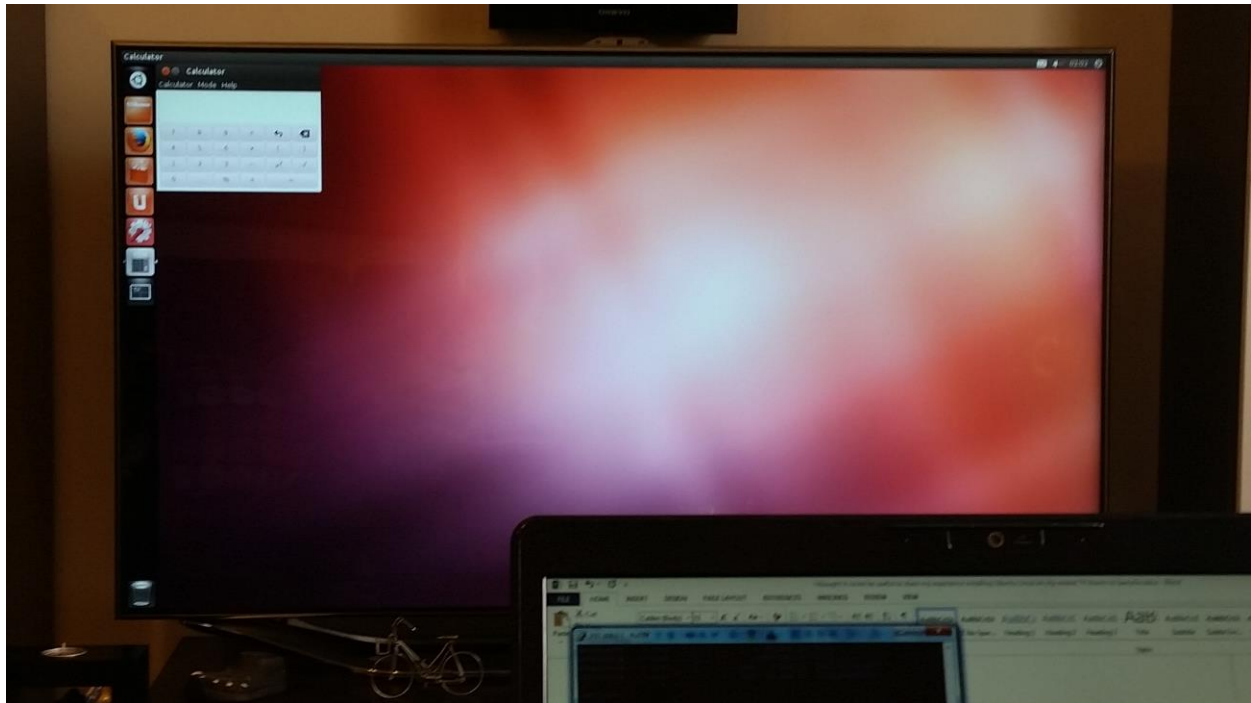
And now, let's start our graphic session:

- `root@(none):/# su sky`
- `sky@localhost:/$ gnome-session`

You may get a lot of warnings, errors, etc... Just ignore them and after a pair of minutes, this is what you should be getting in your TV:



You can go ahead and open another telnet session, enter the chroot, set the environment variables again, log in your account, and start opening graphic applications that perfectly work on the XServer and are now integrated in the desktop, for example, the calculator:



Another interesting one in the SmartHub itself, now integrated in the desktop, you can simply hit the SmartHub button in the remote and it will show up, within an X Window named "browser":



Note that it is still fullscreen, but it does not fit now, since we have the Unity desktop panels, also the PIP frame is a little bit off-place due to those panels ^^.

So... How do we directly interact with our new Desktop? This is no easy task, one does not simply connect a mouse and use it. This is because the TV's XServer does not have mouse support, instead, the TV mouse is hardcoded to work inside the SmartHub and its widgets, so if you connect a mouse, it will only work there, and ignore the input over the new desktop elements. This is regardless of `/dev/input/mice`, which actually works as expected.

How can we get around this? Well, there are several alternatives here:

- Configure the XServer (using the `Xorg.conf` present in `/mtd_exe/Runtime/bin`) to add mouse support. However, since we don't have bind mounts, and the `/mtd_exe` partition is read-only we cannot edit the XServer configuration file without touching the TV's internal memory.
- Use XdoTool: a package that allows us to use a virtual mouse (and keyboard) from the command line. In theory we can write a script to connect the data from `/dev/input/mice` to XdoTool. But since this takes time and I'm alone here, we'll just go with the third option.
- Control the TV over VNC. Yay! The fastest way. The con here is that we need a computer to control the desktop remotely; but that's just fine for some testing ^^

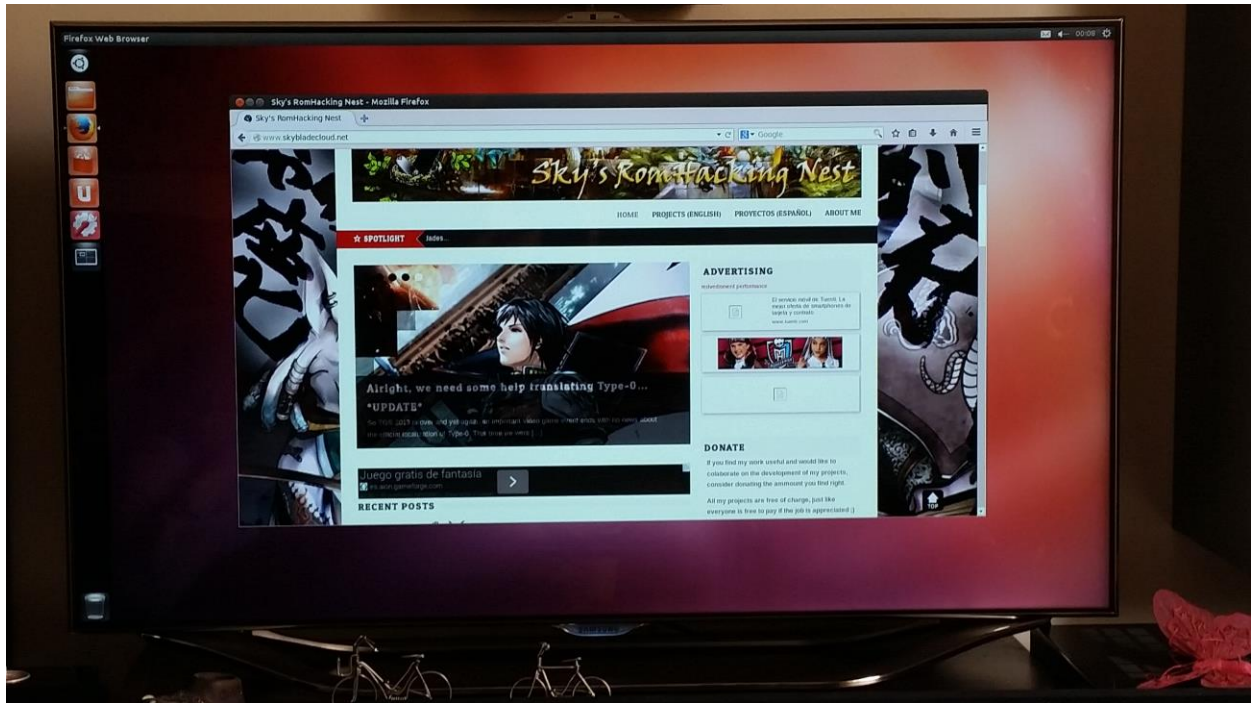
So let's install a VNC server, I'm using sudo, since I have a session as a non-super-user:

- `sky@localhost:/$ sudo apt-get install x1vnc`

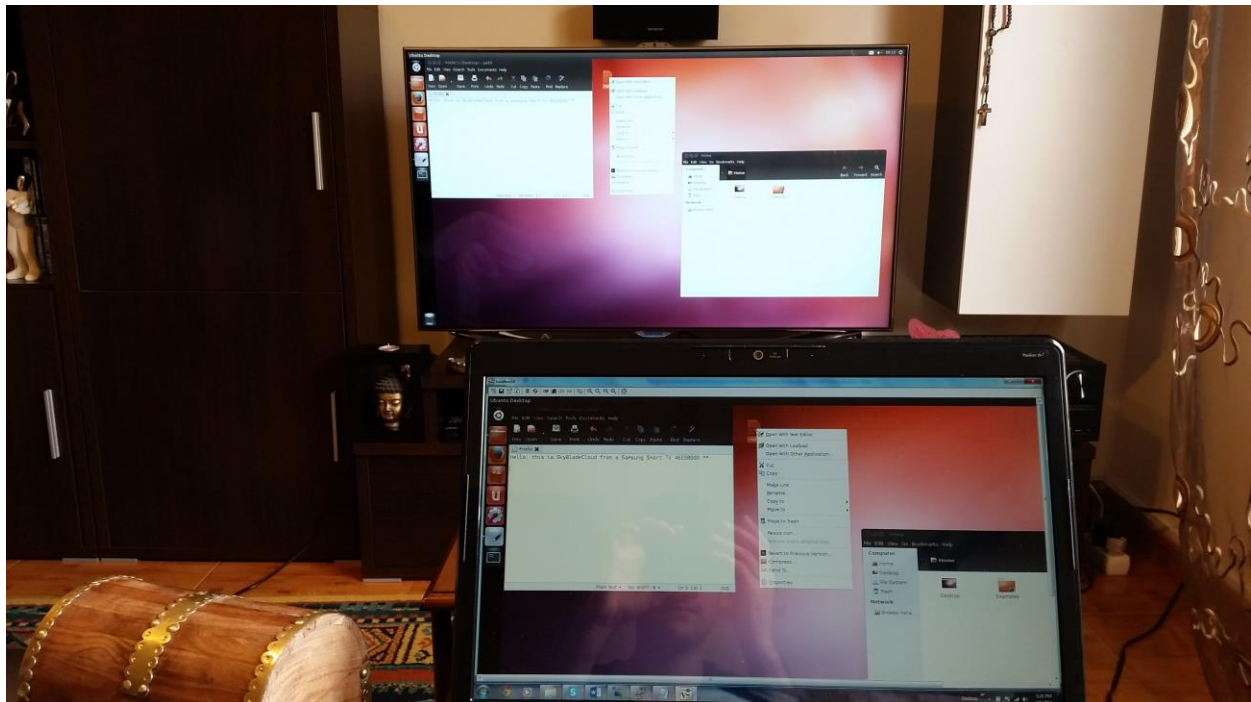
To start the VNC, you need to be root so...

- `sky@localhost:/$ exit`
- `root@localhost:/# x11vnc`

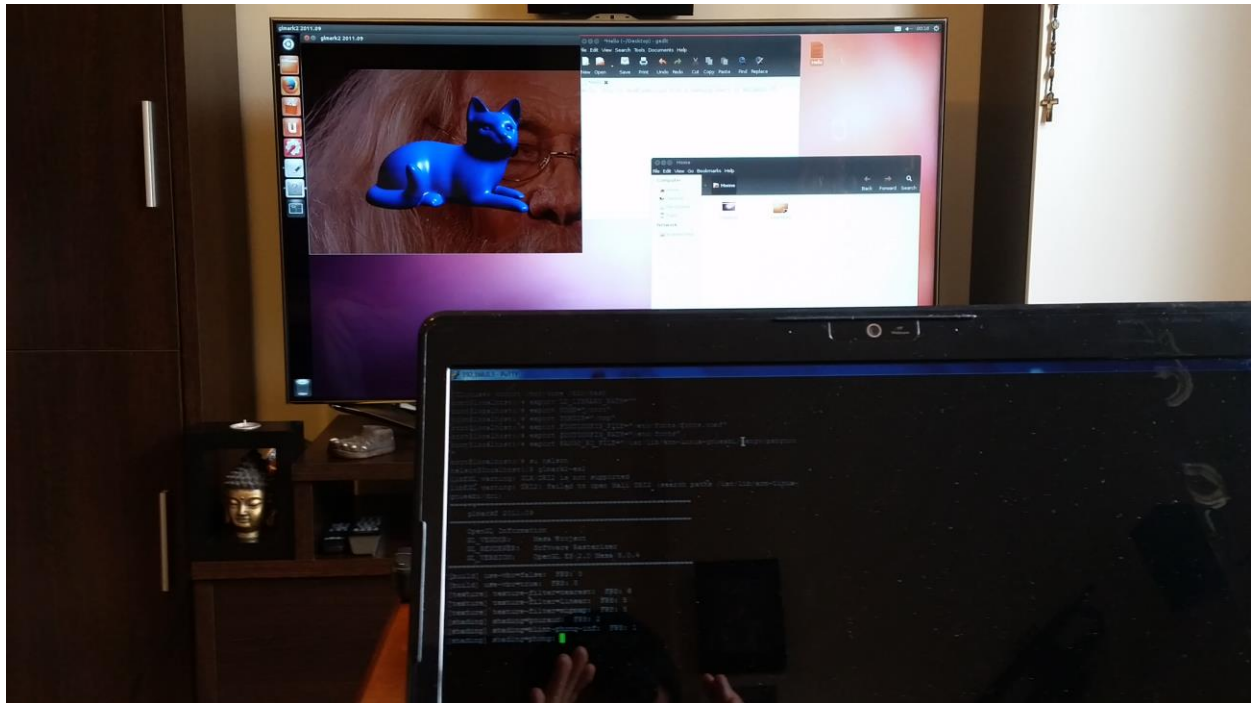
I'll be using Portable TightVNC from my windows laptop to control the desktop over VNC, we can for example launch Firefox and go to my blog ^^



Just like that, we have all Ubuntu Linux tool available and working:



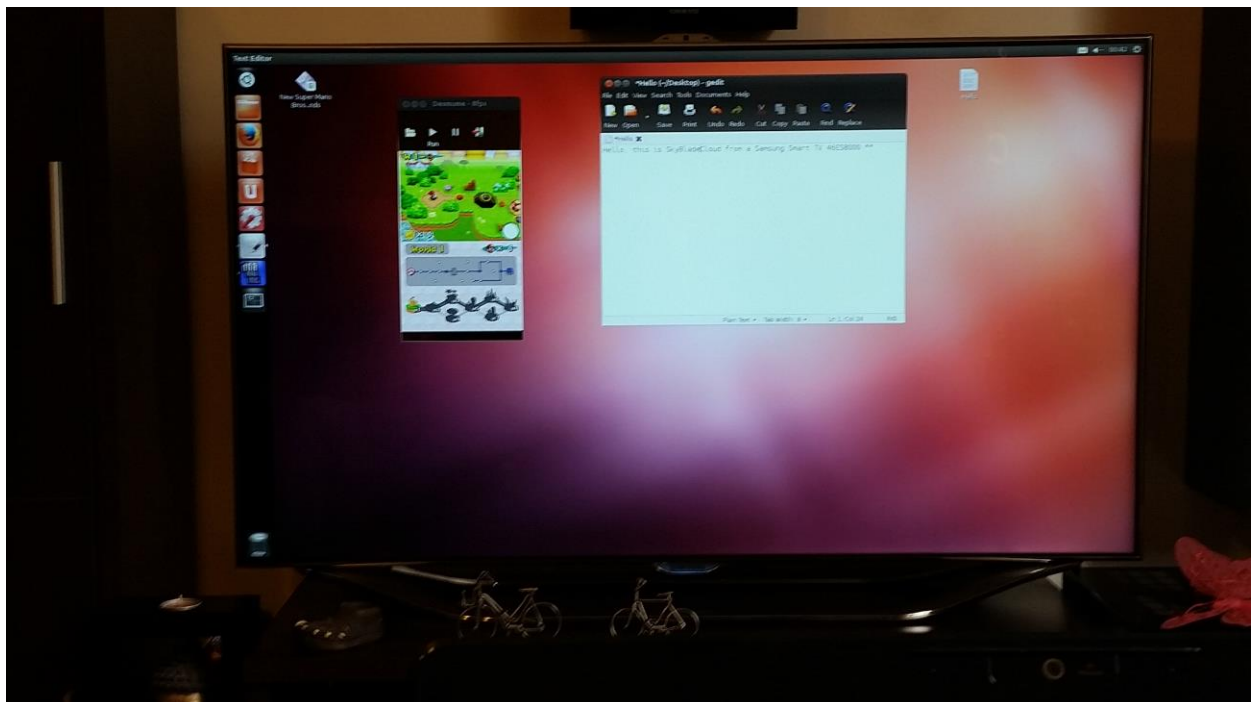
At this point all we need is to configure the rest of devices our TV has. The biggest drawback here is the lack of hardware graphic acceleration, making the whole environment as slow as the SmartHub (and even slower in many applications). In my case, this is partially due to the VNC connection, but as you can see in the next pic, 3D graphic applications run over the MesaGL library, which is a software rendering pipeline, making any game pretty much unplayable:



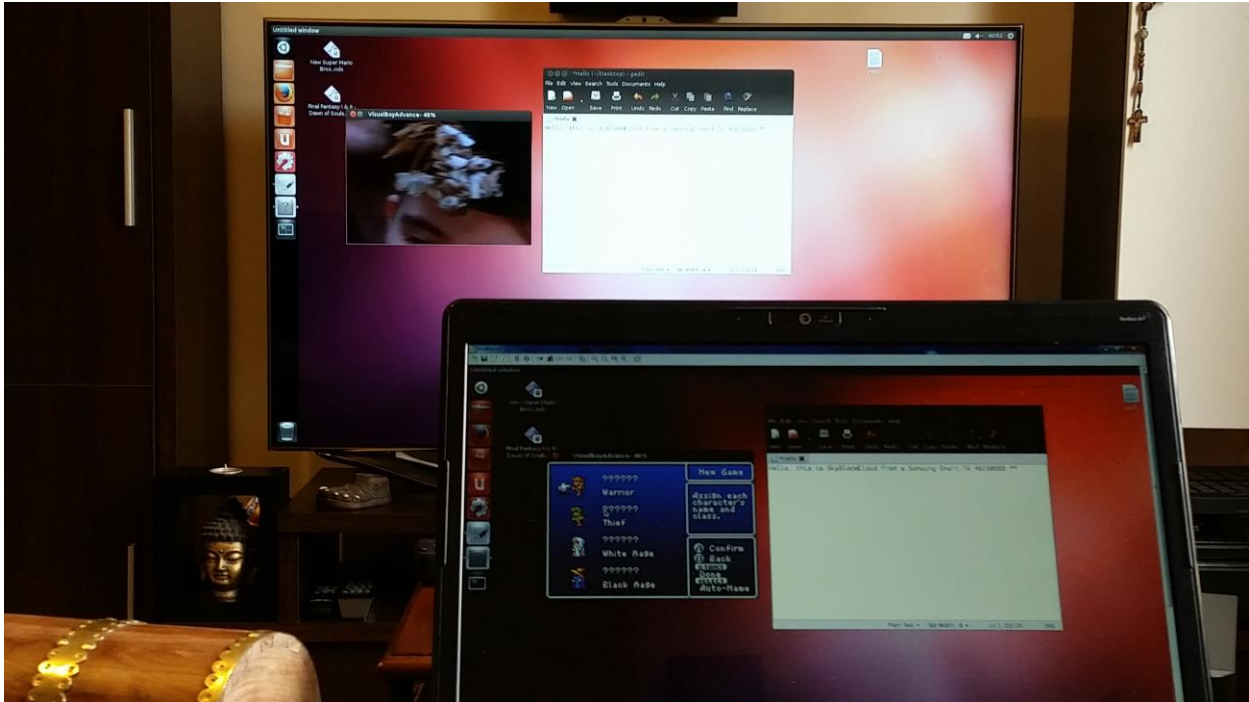
This is glmark2-es2, a 3D benchmark application that, as you can see, uses a transparent background. I started it from command line to see its output: I usually get between 3 and 7 frames per second :/

Still, we can even install emulators and use the TV as a console xD.

- Desmume - Nintendo DS, running New Super Mario Bross (6fps)



- Visual boy Advance – Game Boy Advance, Running Final Fantasy Dawn Of Souls. Unfortunately, it's coded using a SDL surface, which the TV interprets as transparent :/ At least it was getting playable speeds.



The graphic adaptor my TV has is the Mali400 GPU, for some reason, the XServer is not configure to use it (maybe, just like the mouse, the Smathub is hardcoded to directly use it). Moreover, there are no Mali driver available for the armel architecture, you can check the rest here;

http://linux-sunxi.org/Mali_binary_driver

Some other stuff that need extra configuration to work (AKA TODO list)

- The flash player: Currently the browser closes upon attempting to load a YouTube movie :/
- Sound, microphone bluetooth and camera: Maybe adding the corresponding device nodes is enough, but I haven't really tried.
- SSL Certificates: You need to set the right time (since by default, it's set to the Linux Genesis xD) and update them.

And that's it! If you have any question or anything, feel free to post them, or contact me over Skype (my username is "SkyBladeCloud").

~Sky