Dennis Ogbe

Home About Research Blog

Extending your X11 Desktop using VNC

I love multihead setups and I am a thrifty grad student. Today, this combination lead to an interesting and fun Saturday morning project. I figured out how to add an additional monitor to my dual-monitor set-up, which is already a little convoluted.

Let me clear things up: My main setup currently consists of a Thinkpad with its lid closed connected to two monitors (one over VGA, one over DisplayPort). Since my laptop has no third physical display connector, I had to get creative in order to add a third physical monitor to my set-up.

I use a cheap DisplayLink adapter at home for exactly this reason, but unfortunately the official dirvers are buggy and the whole set-up is generally very unreliable. So, with me being the thrifty grad student that I am, I did not want to shell out another $30 for a piece of hardware that doesn't even really work.

What I ended up doing is pretty neat.

The high-level description of my set-up is the following: I have a desktop machine that I was using headless until today. It is now connected to my third monitor and runs the TightVNC client, which is being served by x11vnc. In other words, I'm using xrandr to create a virtual output device on my laptop, I make it available for "remote" viewing using x11vnc, and use the TightVNC client on my desktop machine to view it over the network. My desktop machine and my laptop are connected via a small switch behind my desk, which means that the connection is fast enough to even watch YouTube videos on my networked monitor.

I was heavily inspired by this post on the arch forums and added some tricks of my own to make this set-up comfortable.

For this blog post, I will follow along the lines of the forum post above and describe how this can be replicated. I have most of the code below sitting in a bash script, which I invoke whenever I get to the office in the morning.

Step 1: Create a temporary directory to hold PIDs and logs

This is mostly for debugging purposes and will hold most of the relevant information for my SSH tunnel and the VNC server.

```
TMPD=/tmp/extra_screen
if ! [[ -d $TMPD ]]; then
    mkdir -p $TMPD
fi
```

Step 2: Force XRandR to connect the `VIRTUAL1` device to our X11 desktop

The below part is still a little messy, but it gets the job done for now. I'm using the new monitor in a portrait orientation, so I'm creating a output of dimension `1024x1280`. I'm also telling xrandr to put this output to the right of the output `DP2`.

```
# the local virtual xrandr device
DEVICE="VIRTUAL1"

# get the modeline from
# gtf 1024 1280 60 | sed -n 's/.*Modeline "\([^" ]\+\)"
MODELINE="1024x1280_60.00 110.66 1024 1096 1208 1392 128
NAME="1024x1280_60.00"

# FIXME get rid of error messages
xrandr --delmode "$DEVICE" "${NAME}"
xrandr --rmmode "${NAME}"
xrandr --newmode ${MODELINE}
xrandr --addmode "$DEVICE" "${NAME}"
xrandr --output $DEVICE --mode $NAME --right-of DP2
```

The above commands extend my desktop over a—as of right now—invisible area of `1024x1280` pixels. We could theoretically drag windows in this area. Next, I'm using VNC to make this area available to a client on my desktop machine.

Step 3: Generate a random VNC password and store it on the desktop machine

From now on, any reference to `$REMOTE` should be replaced with SSH username and hostname, e.g. `foo@bar.baz`.

```
PW=$(openssl rand -hex 50 | vncpasswd -f | tee $TMPD/vnc
scp $TMPD/vncpw $REMOTE:~/.ava_vncpw
```

Step 4: Create a reverse SSH tunnel to the desktop machine

Since this is the 21st century, we want to encrypt any traffic that we send through a network. Here, we set up a reverse SSH Tunnel, which forwards port 5900 on my laptop to port 5900 on my desktop machine through an encrypted pipe. I'm also using the nohup command to send the process in the background and have it persist even after I close my terminal.

```
PORT=5900
nohup ssh -2tnNv -R $PORT:localhost:$PORT $REMOTE > $TMP
echo -n $! > $TMPD/tunnel_pid # write the pid of the ssh
```

Step 5: Launch `x11vnc`, clipped to the invisible region

x11vnc has a bunch of configuration options and I spent a significant amount of time reading tweaking its configuration to get the best performance.

```
CLIP=$(xrandr | grep "^$DEVICE.*$" | grep -o '[0-9]*x[0-
nohup x11vnc -clip $CLIP -noxinerama -noxrandr \
      -repeat -localhost -nevershared -forever \
      -rfbauth $TMPD/vncpw \
      -nowf -noncache -wait 1 -defer 1 > $TMPD/x11vnc_lo
echo -n $! > $TMPD/x11vnc_pid # write the pid of x11vnc
```

- `-clip` defines the area which is exported. This coincides with the area of the output created using the mode line above

- I had to set `-noxinerama` to get rid of some nasty letterboxing
- Without setting `-repeat`, I couldn't enter a letter more than once by long-pressing a key on the keyboard. That was annoying
- `-localhost` lets `x11vnc` only accept connections from `localhost`, which is fine since we're using a SSH tunnel.
- `-forever` makes `x11vnc` not quit after the first client quits
- without the `-noncache` switch, `x11vnc` exported a much larger area filled with mostly black pixels

Step 4: Be lazy and write a script to kill `x11vnc` and the SSH tunnel

This saves a few keystrokes during debugging.

```
KILL=$TMPD/kill.sh
echo "#!/bin/sh" > $KILL
echo -n "kill $(cat $TMPD/tunnel_pid) && " >> $KILL
echo -n "kill $(cat $TMPD/x11vnc_pid) && " >> $KILL
echo "echo \"Killed x11vnc and SSH.\"" >> $KILL
chmod +x $KILL
```

Step 5: Start a VNC client on the desktop machine

Since I'm not doing anything else on this machine other than running the VNC client, I have an infinite loop in my `.xinitrc` file which attempts to restart the client if it is not running.

```
# rotate the display
xrandr --output DVI-0 --rotate left
# disable screen sleep
xset s off
xset -dpms
# unclutter removes the local mouse pointer
unclutter &
force reloading of the vnc viewer to simulate plug & pla
while true; do
    if ! ps aux | grep -v grep | grep -q vncviewer; then
        vncviewer localhost:0 -passwd ~/.ava_vncpw -view
    else
      sleep 0.5
    fi
done
```

Step 6: Celebrate!

This was fun. I hope this inspires someone to do their own thrifty grad student multihead set-up.

RSS