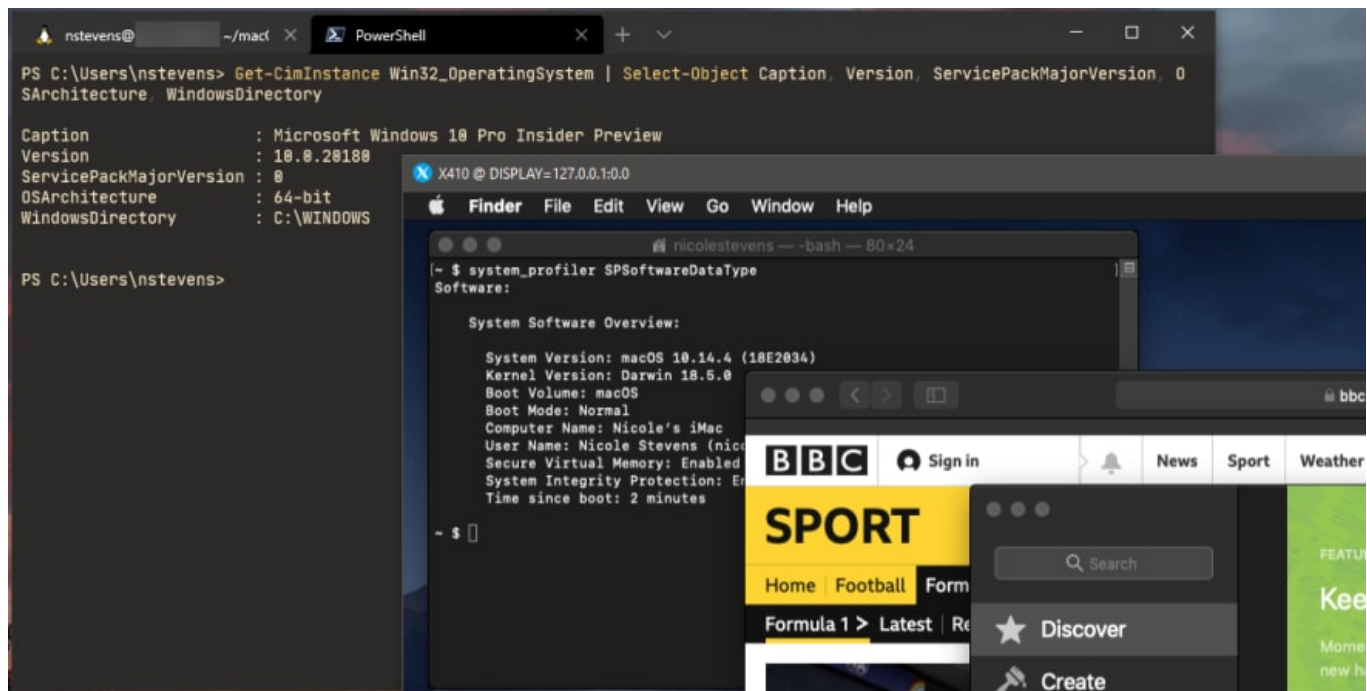


Running MacOS on Windows 10 with WSL2, KVM and QEMU



I needed to record a demo on a Mac, I don't own a Mac and was contemplating borrowing one from a friend. Then I realised, I finally had an excuse to give this a go! I've been itching to since I read - [Accelerated KVM guests on WSL 2](#), an awesome write-up on how to run accelerated [KVM](#) guests on WSL2 from [@unixterminal](#)

This is a walkthrough of how I used the excellent work of [@unixterminal](#) and [@FoxletFox](#) and got my 3 year old XPS Intel i7 to run MacOS on Windows 10! Without their writeups and scripting this post probably wouldn't exist.

I am still stunned how good the performance is having run through this! I've tried a couple of other Linux distro's too, seriously slick.

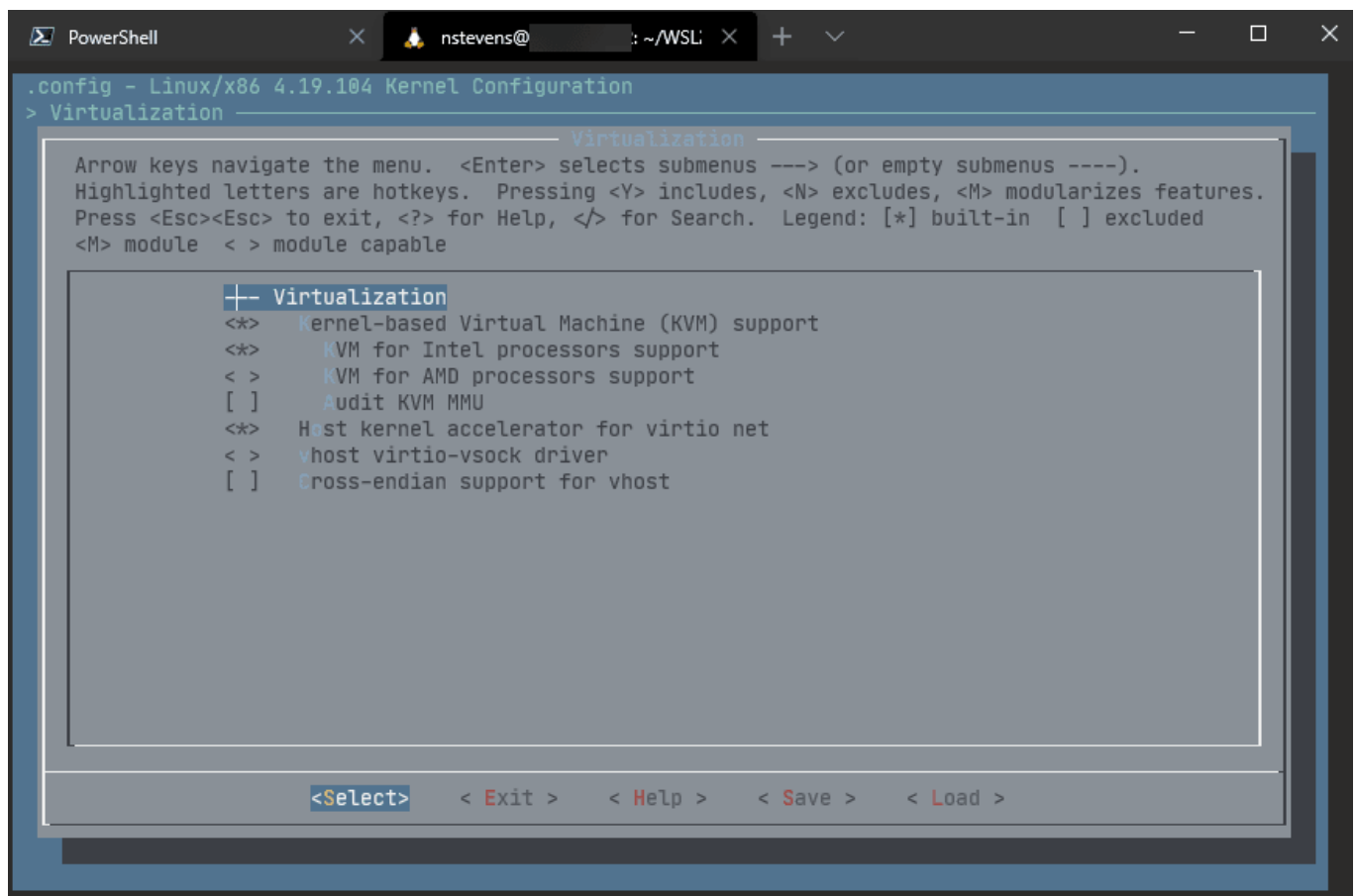
Before starting check out the requirements from the top of the ["Accelerated KVM guests on WSL2"](#) post. One thing that's not obvious is your CPU needs to support virtualisation. Unless you're running an Intel CPU from the early 2000s or even earlier you should be OK. If your CPU does support virtualization it might not be enabled in your BIOS, it's worth a check before you start. If you're already running a hypervisor, it's likely this is good to go.

You'll also need to be running a [windows insider](#) build of Windows 10. The insiders Fast ring which is mentioned in the pre-requisites is now the Dev channel. One of the features required didn't come into Windows 10 until build 19619 and the [nested virtualisation support for AMD](#) didn't come in until build 19636 either, so that's a must if you are on AMD too. The [windows insider](#) page currently lists 19042.423 as the highest build available in Beta and 19041.423 as Release Preview, so for the moment this will not work correctly without using the Dev channel.

The insider channel renaming is [described here](#). Dev channel is described as *Ideal for highly technical users. Insiders in the Dev Channel will receive builds that is earliest in a development cycle and will contain the latest work-in-progress code from our engineers. These builds will have rough edges and some instability that could block key activities or require workarounds.*

With this in mind, think twice about running the Dev channel on your main machine, or if you wouldn't consider yourself a highly technical user! If you're OK with that, dive right in :

1. Follow the steps in "[Accelerated KVM guests on WSL2](#)" until you reach the section titled **Important note about building a module in WSL**: Time was fairly tight for getting this up and running, I wasn't going to be tweaking performance and I didn't want to keep loading the KVM for Intel module manually, so:



```
PowerShell
nsteven@: ~/WSL:
.config - Linux/x86 4.19.104 Kernel Configuration
> Virtualization
  Virtualization
  Arrow keys navigate the menu.  <Enter> selects submenus ---- (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.
  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in [ ] excluded
  <M> module  < > module capable

  +- Virtualization
  <*> Kernel-based Virtual Machine (KVM) support
  <*>   KVM for Intel processors support
  < >   KVM for AMD processors support
  [ ]   Audit KVM MMU
  <*> Host kernel accelerator for virtio net
  < > vhost virtio-vsock driver
  [ ] Cross-endian support for vhost

  <Select>  < Exit >  < Help >  < Save >  < Load >
```

KVM for Intel processors was selected as above, note that the AMD selection is below it.

1. Carry on from "Exit the Virtualization" paragraph until you get to this command:

```
kvm-ok
```

Enter fullscreen mode Exit fullscreen mode

If you've followed the steps carefully and hit all the pre-reqs when you execute `kvm-ok` you should see:

```
INFO: /dev/kvm exists
KVM acceleration can be used
```

Enter fullscreen mode Exit fullscreen mode

If not you are likely to see:

```
INFO: Your CPU does not support KVM extensions  
KVM acceleration can NOT be used
```

Enter fullscreen mode Exit fullscreen mode

If you are seeing the above you are definitely running a Windows 10 Dev channel, run back through the steps once more, you might have missed something.

1. Check nested KVM is OK

If you're getting "KVM acceleration can be used" you can proceed with the steps in the post.

The next steps is:

```
cat /sys/module/kvm_intel/parameters/nested
```

Enter fullscreen mode Exit fullscreen mode

When executing this command N was being returned. Having traced back through the steps it was time for a search which yielded an [issue from the WSL github repo](#). So reverting back to the step **Install your kernel in WSL 2 and enable nested KVM** the .wslconfig was changed to:

```
nestedVirtualization=true kernel=C:\\Users\\<username>\\bzImage  
debugConsole=true pageReporting=true kernelCommandLine=intel_iommu=on iommu=pt  
kvm.ignore_msrs=1 kvm-intel.nested=1 kvm-intel.ept=1 kvm-  
intel.emulate_invalid_guest_state=0 kvm-intel.enable_shadow_vmcs=1 kvm-  
intel.enable_apicv=1
```

Enter fullscreen mode Exit fullscreen mode

You'll need to restart WSL once more and check the command once more.

Note the command line args are almost identical to those added to kvm_intel.conf, I haven't investigated why the nested setting isn't picked up from the conf file.

Also note that setting debugConsole opens up the WSL debug console, you can turn this and pageReporting off later by removing them from the .wslconfig file and restarting WSL. I left it in for a while in case I needed to troubleshoot further.

The check for nested KVM now returned a Y as required:

```
cat /sys/module/kvm_intel/parameters/nested  
Y
```

Enter fullscreen mode Exit fullscreen mode

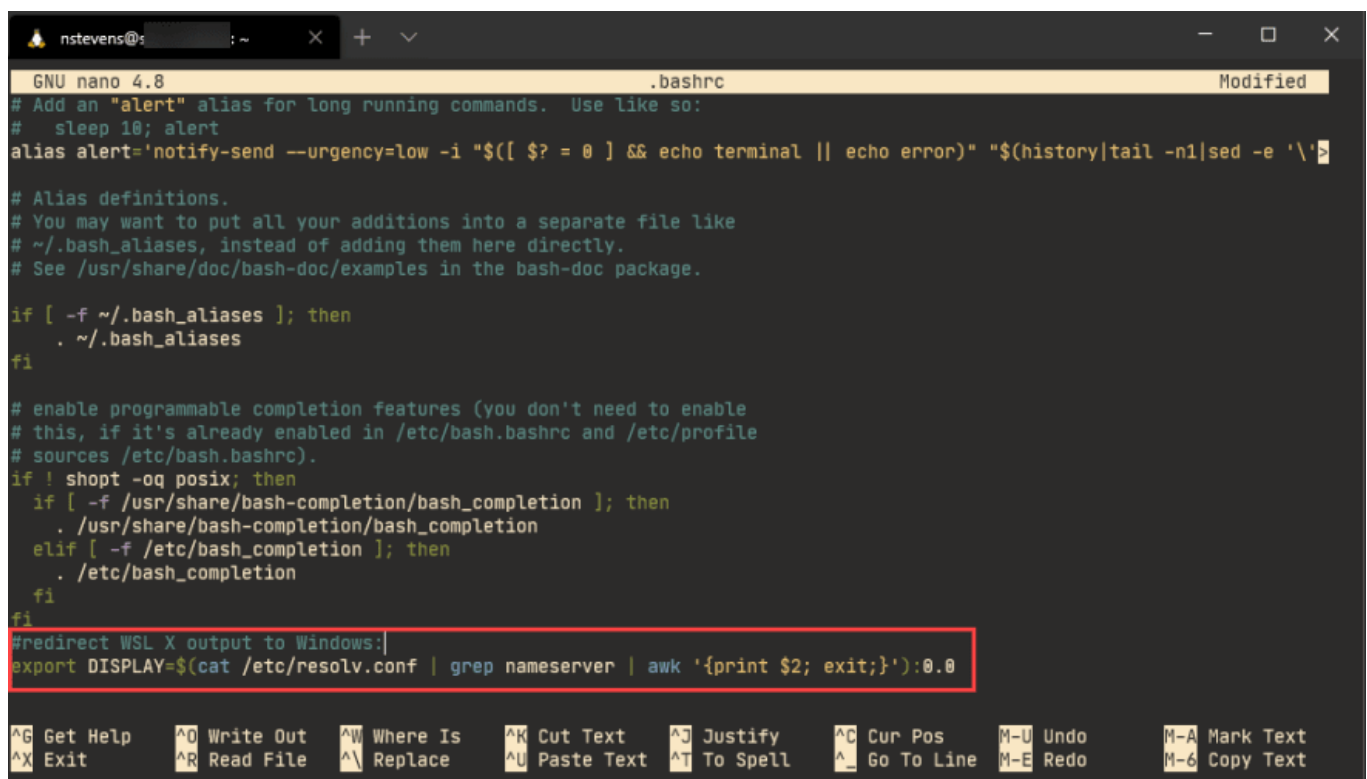
1. Config X for WSL2.

Having run a far few X servers in the past I just wanted something that just worked without a lot of setup. Therefore, I went for X410 which is £9ish in the Microsoft store, but as I'm planning to use this until GUI support comes out later in the year, it was worth the price for the easy setup.

Install as normal in the store then add the environment variable in the steps to your .bashrc:

```
cd ~  
nano .bashrc
```

Enter fullscreen mode Exit fullscreen mode



```
GNU nano 4.8 .bashrc Modified  
# Add an "alert" alias for long running commands. Use like so:  
# sleep 10; alert  
alias alert='notify-send --urgency=low -i "${?} = 0" && echo terminal || echo error)' "${history|tail -n1|sed -e '\>  
  
# Alias definitions.  
# You may want to put all your additions into a separate file like  
# ~/.bash_aliases, instead of adding them here directly.  
# See /usr/share/doc/bash-doc/examples in the bash-doc package.  
  
if [ -f ~/.bash_aliases ]; then  
    . ~/.bash_aliases  
fi  
  
# enable programmable completion features (you don't need to enable  
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile  
# sources /etc/bash.bashrc).  
if ! shopt -oq posix; then  
    if [ -f /usr/share/bash-completion/bash_completion ]; then  
        . /usr/share/bash-completion/bash_completion  
    elif [ -f /etc/bash_completion ]; then  
        . /etc/bash_completion  
    fi  
fi  
#redirect WSL X output to Windows:  
export DISPLAY=$(cat /etc/resolv.conf | grep nameserver | awk '{print $2; exit;}'):0.0
```

Once nested KVM is up and running and you've configured WSL to send X output to windows, you are ready to try a distro. If you keep following the post at this point you'll set up an Ubuntu 20.10 daily build. I ran through this as a check to see if everything was working, the steps for this were exactly the same as in the post.

1. Setup for MacOS VM

The [MacOS-Simple-KVM](#) GitHub repo is linked from the original post. It is described as "set up a simple macOS VM in QEMU, accelerated by KVM." If you ran through setting up the Ubuntu 20.10 daily build in step 4 you'll already have seen [QEMU](#) being mentioned. You might be thinking, why are there two types of virtualization technologies? The key part is in how KVM and QEMU differ, which is summed up nicely in this [post](#). KVM uses the CPU virtualization extensions for Intel and AMD, and

QEMU is performing the virtual hardware emulation, or to put it another way, KVM is QEMU's "go faster stripes"!

The steps were followed as listed but with the following tweaks. First clone the git repo in, make sure to use the linux filesystem as it is [faster](#) at the moment on wsl2 than windows (/mnt).

This walkthrough is using Ubuntu so it's the top line to install QEMU, Python and Pip if you haven't already got them:

```
sudo apt-get install qemu-system qemu-utils python3 python3-pip
```

Enter fullscreen mode Exit fullscreen mode

Run jumpstart.sh to fetch the MacOS media of your choice:

```
./jumpstart.sh --mojave
```

Enter fullscreen mode Exit fullscreen mode

I also went with Mojave.

Now create a hard disk, I went with 32GB, I wanted to try and go a little lower but on investigation I found some issues looked where less than 32GB looked to be creating issues for other users:

```
qemu-img create -f qcow2 MacOS.qcow2 32G
```

Enter fullscreen mode Exit fullscreen mode

Note I've changed the name of my disk, by this point I had a few VM's setup so wanted to make it more obvious!

Edit basic.sh to add the VM disk into the VM :

```
-drive id=SystemDisk,if=none,file=MacOS.qcow2 \  
-device ide-hd,bus=sata.4,drive=SystemDisk \
```

Enter fullscreen mode Exit fullscreen mode

Now make sure the X server from step 4 is running as QEMU will try and use graphics mode by default. If it isn't running when you run basic.sh in this next step it'll have for a minute or so then fail:

```
./basic.sh
```

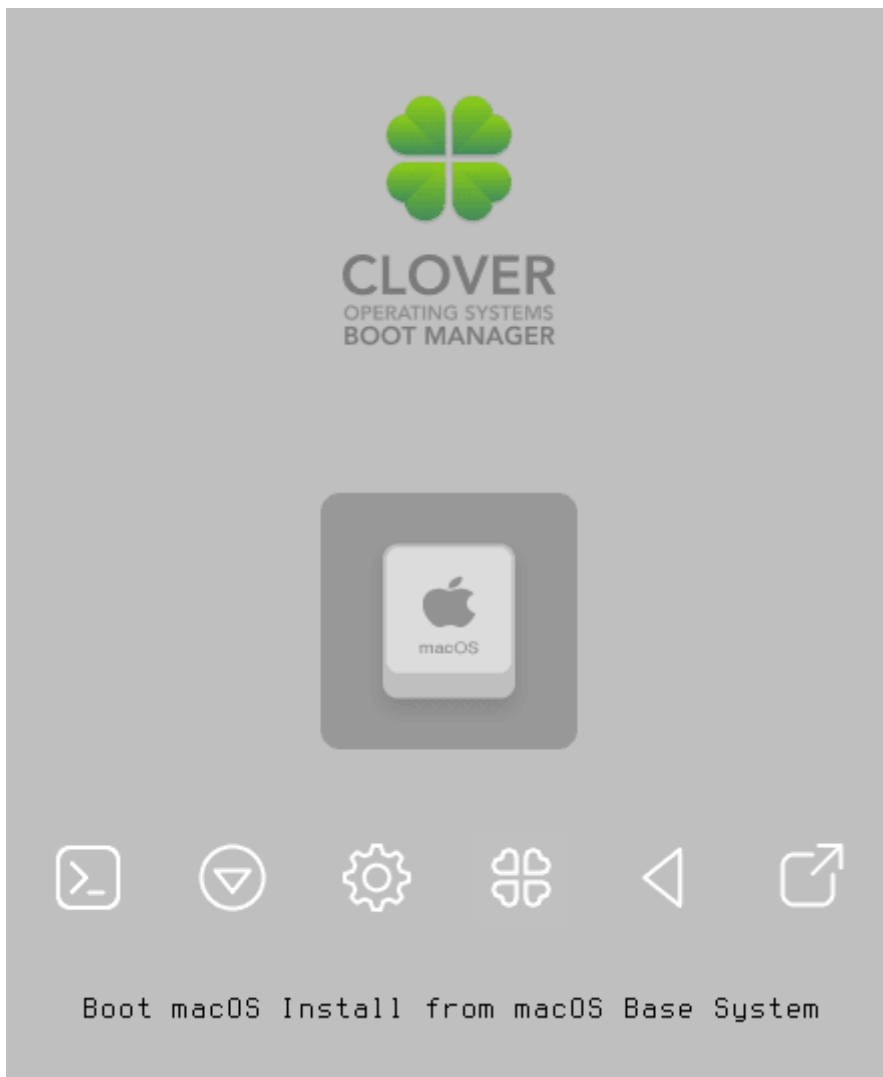
Enter fullscreen mode Exit fullscreen mode

The VM boots but there are some errors:

```
qemu-system-x86_64: warning: host doesn't support requested feature: CPUID.80000007H:EDX.invtsc [bit 8]
qemu-system-x86_64: warning: host doesn't support requested feature: CPUID.80000007H:EDX.invtsc [bit 8]
qemu-system-x86_64: warning: host doesn't support requested feature: CPUID.80000007H:EDX.invtsc [bit 8]
qemu-system-x86_64: warning: host doesn't support requested feature: CPUID.80000007H:EDX.invtsc [bit 8]
ALSA lib confmisc.c:767:(parse_card) cannot find card '0'
ALSA lib conf.c:4693:(_snd_config_evaluate) function snd_func_card_driver returned error: No such file or directory
ALSA lib confmisc.c:392:(snd_func_concat) error evaluating strings
ALSA lib conf.c:4693:(_snd_config_evaluate) function snd_func_concat returned error: No such file or directory
ALSA lib confmisc.c:1246:(snd_func_refer) error evaluating name
ALSA lib conf.c:4693:(_snd_config_evaluate) function snd_func_refer returned error: No such file or directory
ALSA lib conf.c:5181:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2642:(snd_pcm_open_noupdate) Unknown PCM default
alsa: Could not initialize DAC
alsa: Failed to open 'default':
alsa: Reason: No such file or directory
ALSA lib confmisc.c:767:(parse_card) cannot find card '0'
ALSA lib conf.c:4693:(_snd_config_evaluate) function snd_func_card_driver returned error: No such file or directory
ALSA lib confmisc.c:392:(snd_func_concat) error evaluating strings
ALSA lib conf.c:4693:(_snd_config_evaluate) function snd_func_concat returned error: No such file or directory
ALSA lib confmisc.c:1246:(snd_func_refer) error evaluating name
ALSA lib conf.c:4693:(_snd_config_evaluate) function snd_func_refer returned error: No such file or directory
ALSA lib conf.c:5181:(snd_config_expand) Evaluate error: No such file or directory
ALSA lib pcm.c:2642:(snd_pcm_open_noupdate) Unknown PCM default
alsa: Could not initialize DAC
alsa: Failed to open 'default':
alsa: Reason: No such file or directory
audio: Failed to create voice 'dac'
```

The top section boxed in red is caused by the `-cpu` line in [basic.sh](#). The standard settings in the `basic.sh` file are sending instructions around performance enhanced features to the host CPU which it [cannot understand](#). The box in green are errors related to sound, I'm not so worried about those, and explain my sound related issues with Ubuntu above. You can edit `basic.sh` further to straighten these out, but the VM still booted with these errors present.

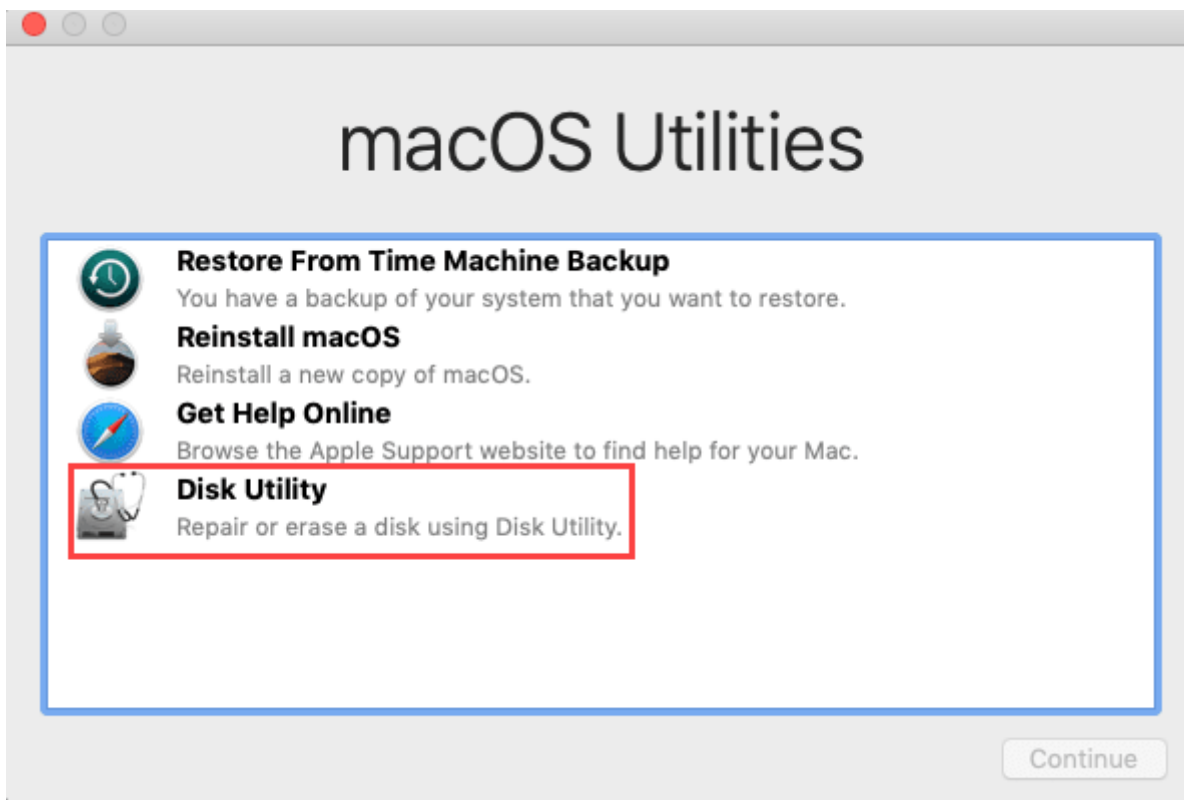
The VM boots to the Clover boot manager, where you can use your keyboard to hit return (no mouse at this point) and "Boot macOS Install from macOS Base System" will begin:



The macOS utilities screen is now displayed and you should have mouse support. If your mouse pointer is out of synchronisation with the one on the VM you will need to adjust the settings in the basic.sh file. Shut the VM down using the apple icon top left and edit basic.sh in nano once more:

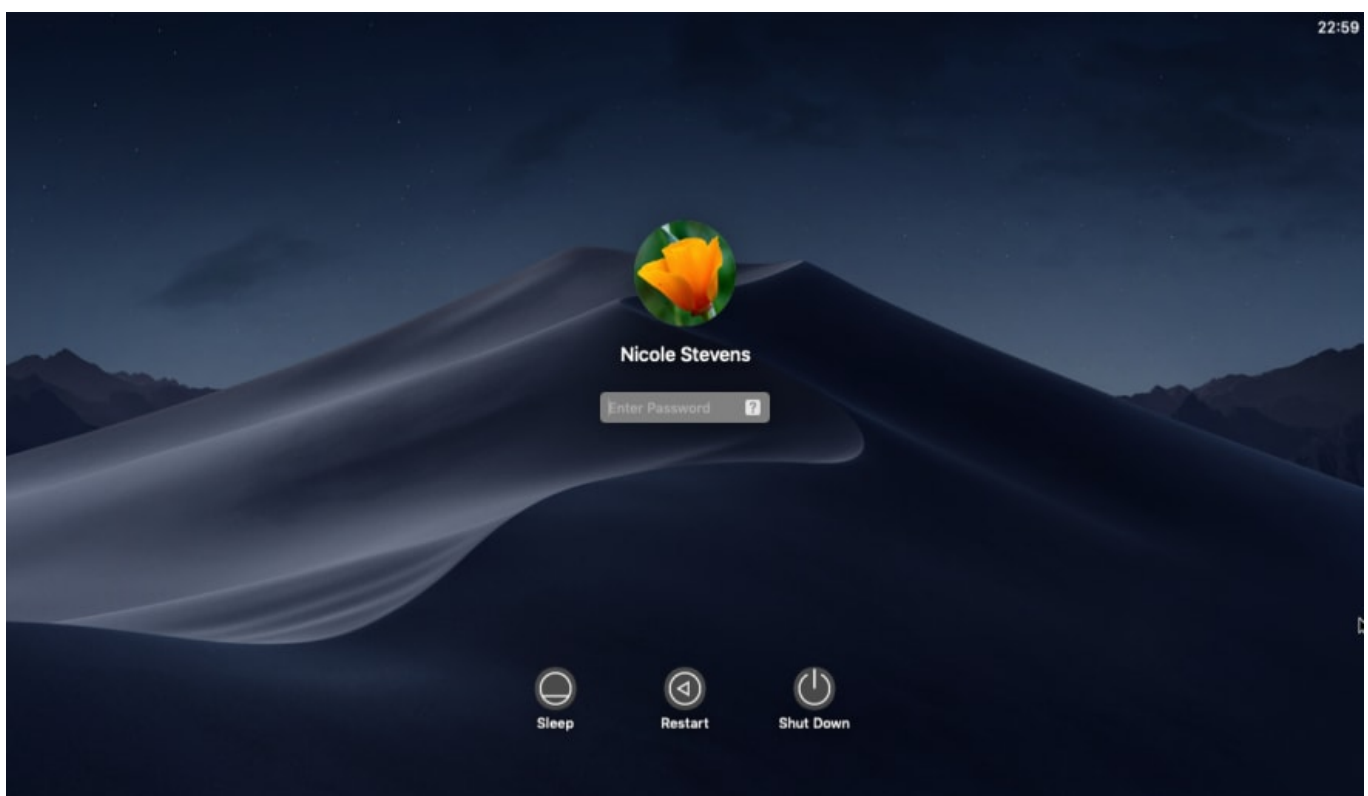
```
qemu-system-x86_64 \
-enable-kvm \
-m 2G \
-machine q35,accel=kvm \
-smp 4,cores=2 \
-cpu Penryn,vendor=GenuineIntel,kvm=on,+sse3,+sse4.2,+aes,+xsave,+avx,+xsaveopt,+xsavc,+xgetbv1,+avx2,+bmi2,+smep,
-device isa-applesmc,osk="$OSK" \
-smbios type=2 \
-drive if=pflash,format=raw,readonly,file="$QVMF/OVMF_CODE.fd" \
-drive if=pflash,format=raw,file="$QVMF/OVMF_VARS-1024x768.fd" \
-vga qxl \
-device ich9-intel-hda -device hda-output \
-usb -device usb-kbd -device usb-tablet \
-netdev user,id=net0 \
-device e1000-82545em,netdev=net0,id=net0,mac=52:54:00:c9:18:27 \
-device ich9-ahci,id=sata \
-drive id=ESP,if=none,format=qcow2,file=ESP.qcow2 \
-device ide-hd,bus=sata.2,drive=ESP \
```

The usb device line needs to be altered from usb-mouse to usb-tablet. Run basic.sh once more and your mouse should now align.



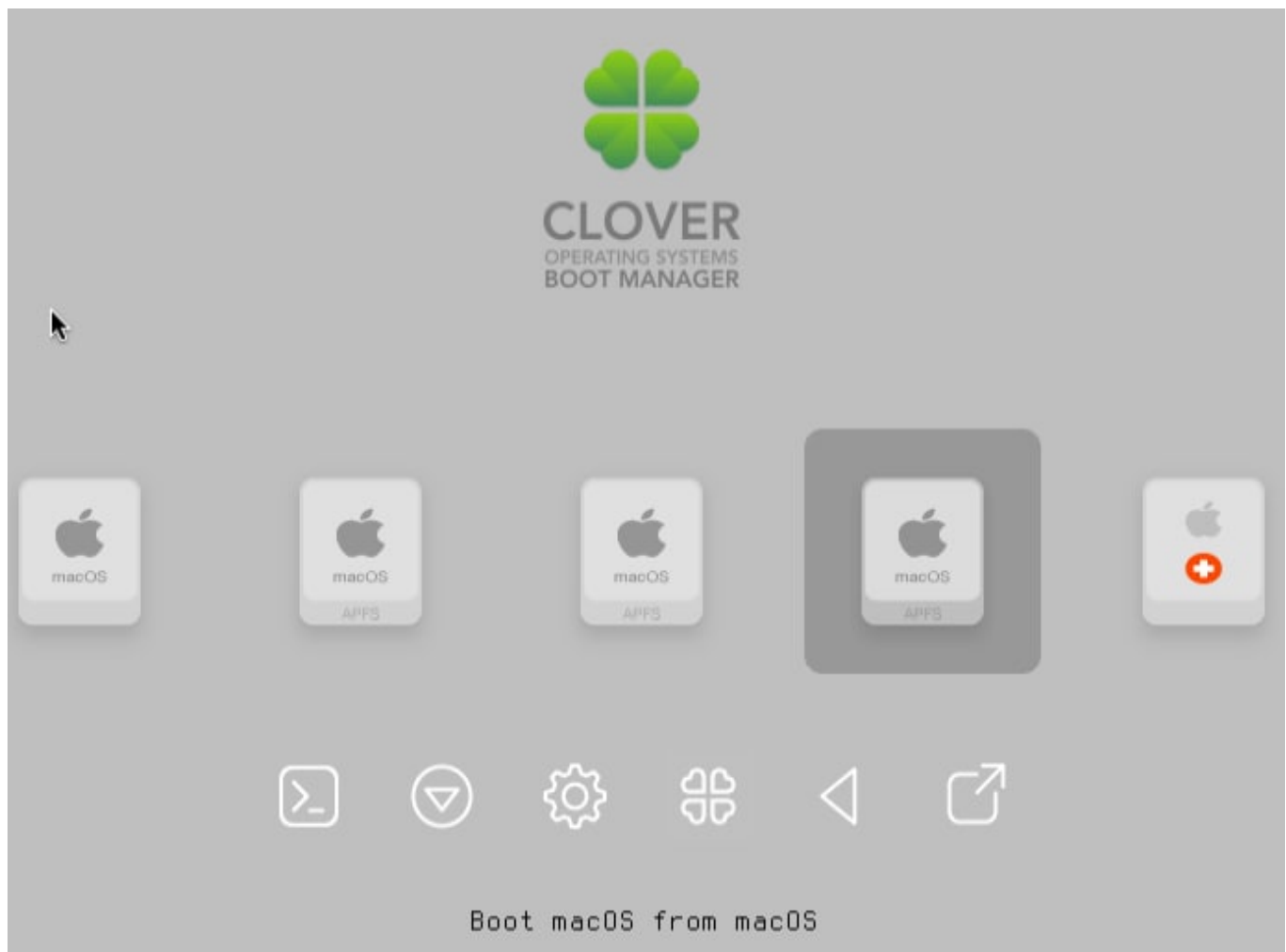
As mentioned in the steps from the [macOS-Simple-KVM](#) instructions, you need to partition the disk first. If you enter Disk Utility, find your QEMU HARDDISK from the left hand menu and Erase it and partition it.

You can now return back to the macOS Utilities menu and choose Reinstall macOS, from here it is the same as a standard reinstall, this does take quite a while! No further tweaks were required from this point on:



You can move in and out of fullscreen with ctrl+alt+backspace and remove the VM banner with "Machine View" at the top with ctrl+alt+F.

If you now leave your setup as-is with no further tweaks you will be sent into the Clover bootloader each time you reboot or run your VM:



I've got a few more options to choose from than you will have as I'd been playing around with things before I took this. The key thing is you need to use the arrow keys to choose "Boot macOS from macOS", this will boot your install and you are good to go!