Tailoring a Linux environment on to a MacPro 2019

Ross Maloney

August 2020

Abstract

Setting up of a Linux environment for computing research on a MacPro 2019 (cheese grater) while preserving OS X is described. Selective booting between those two environments is the result. A minimal Debian `sid` Internet download is supplemented by selective Debian package installations. An Nvidia GPU for parallel computing is added to the hardware and software. Then source code builds of required additional components such as `R`, `mariadb`, `dwm`, `st` and `vim` are described, then LaTeX is obtained from the Internet. Bluetooth control of the MacPro under Debian is then implemented. The result is a Linux environment servicing a specific need and an OS X system providing general services, both on the same hardware. This paper is based on the tasks performed, sometimes using separated results others have discovered and posted on the web, and consolidated here into a MacPro target environment.

The hardware was a MacPro 2019 (cheese grater) with an Intel 2.5GHz Xeon W 28 core processor, 32GB of RAM, 256GB of SSD, and a AMD Radeon Pro 580X graphics processor. It was delivered with OS X Catalina (OS X 10.15.3) installed. This hardware had the T2 security chip installed.

The indigenous OS X software which is installed by default on the MacPro only handled a small part of the needs. The majority of those needs where fulfilled by Linux, which could be configured exactly to the needs. Although at any one time OS X or Linux would be in use, the hardware would be shared. So dual booting of OS X and Linux was required. This is not a natural state for the MacPro hardware. Having bought the hardware, the manner of its use was not predefined. Since the hardware was not to change nor OS X removed, warranty was not invalidated.

A Debian Linux using a Internet installation of the `bullseye sid` distribution was used. A minimal functioning system resulted. Package installation across the Internet was used to add specific needed components. Those components were boosted by needed programs which were built from source code downloaded from the Internet. The result was a computer environment tailored for the specific need of software development and computing research.

1 Dual booting

The standard approach is to de-activate the T2 security chip. This is done by booting the MacPro into Recovery mode by holding the `command` and `r` key on the keyboard down simultaneously while the MacPro boots. From the window which resulted, the `Utilities` menubar was selected then the `Startup Security Utilities` entry, on which the `No Security` radio button was set and the MacPro rebooted for this to take affect.

In OS X Catalina an account `ssor` was created. A copy of the file `refind-bin-0.11.5.zip` was downloaded from `sourceforge.net` (the latest version `refind-bin-0.12.0.zip` failed to

work correctly). This file was copied into the base directory of the `ssor` account and the utility `unzip` applied to the file which resulted in the creation of a `refind-11.5` directory. The `Disk Utility` was used to create a 109GB partition on the OS X SSD to contain Linux.

A copy of the Debian file `debian-testing-amd64-netinst.iso` from `https://cdimage.debian.org/cdimage/daily-builts/daily/arch-lattest/amd64/iso-cd/` with the date 2020-05-25 was downloaded onto a different Linux box connected to the Internet. This image contained `bullseye sid` with Linux kernel 5.6.0-1 and GCC 10.1.0. An installation USB stick was created using a 8GB flash drive by the command:

```
dd if="debian-testing-amd64-netinstall.iso" of=/dev/sdc bs=1M; sync
```

This USB stick was inserted into a USB port on the MacPro and used to install Linux on the removable SSD connected to a USB-C port on the MacPro. The installation was started by re-booting the MacPro.

Prior to this installation another Debian netinst ISO, dated 2020-01-05, containing Linux kernel 5.3.0-3 and GCC 9.2.1 had been used. Several attempts to install Linux onto the MacPro SSD failed using that installation USB stick. A SanDisk 1TB Extreme Pro portable SSD was purchased and connected to the MacPro via a USB-C port. Onto that SSD Linux was successfully installed. The same mobile SSD was used for this installation. The Linux system in the following was not on the MacPro SSD but on this 1TB external SSD.

Once Linux had been installed (see below), the MacPro was reboots into Recovery Mode by holding down simultaneously the `command` and `r` keys on the keyboard during the boot process. From the `Utilities` menubar across the top of the resulting window, the `terminal` entry was clicked. Into the terminal window which resulted the commands:

```
cd /Volumes/Macintosh\ HD/Users/ssor/refind-10.5
./refind-install
```

were typed to install the `rEFInd` boot manager onto the MacPro boot partition of it's SSD.

With the previous installation (using Debian dated 2020-01-05) dual booting into Linux through rEFInd could take several attempts. While the boot process continued, output concerning what was happening appeared on the screen. In such attempts, if not successful, the MacPro would would reboot itself and another attempt could be applied. However, once booted into Linux, the system remained stable. With this Debian installation (dated 2020-05-25) the boot process occurred without problems.

2 Linux specifics

Linux was installed with the SanDisk 1TB portable SSD attached to a USB-C port of the MacPro. From the Debian installation screen, the `install` option was used. A manual disk partitioning was performed when that option appeared in the Debian installation. The partitions set on the 1TB SSD were:

mount point	size
/boot	168 MB
/	10 GB
/usr	10 GB
/opt	15 GB
/tmp	20 GB
/home	945 GB

In this partition layout the `/tmp` directory was for compiling the open source software required. The `/usr` directory was to contain both the system's software together with the user contributed software using the `/usr/local` sub-directory. The `/opt` directory was to contain the TEX system. It was subsequently shown those partition sizes were adequate.

During the installation process a `apt mirror` could not be linked. This was overcome by adopting that option and proceeding without such a link. This apparent short coming was rectified manually after installation was completed.

Upon completion of the installation, the following modifications were performed using the `vi` editor which was installed automatically during the installation. The address in `/etc/resolv.conf` was changed from `192.168.22.250` to `192.168.8.248` which was the address of the modem connected to the NBN/Internet. Next the file */etc/network/interfaces* was opened and the `dhcp` word contained there was replaced with `static`. Then the entries:

```
address   192.168.22.10/24
broadcast   192.168.22.255
gateway   192.168.22.250
dns-nameserver   8.8.8.8
```

were added on the next lines. This gave the network address of the computer to be `192.168.22.10` on the network `192.168.22.0` together with the address `192.168.22.250` of the device on that network which provided a means for a network packet to get off that network. This gateway led to a different network (`192.168.8.0`) which contained the modem connecting to the NBN/Internet. Then the installation failure to link to a `apt` server was rectified. In the `/etc/apt/sources.list` file, the lines:

```
deb http://ftp.iinet.net.au/debian/debian  bullseye main contrib non-free
deb-src http://ftp.iinet.net.au/debian/debian  bullseye  main
```

were added towards the top of the file. The address `http://ftp.iinet.net.au` was used as the Debian online package server.

Linux was then rebooted for these entries to take affect. It was at this point booting into OS X Recovery and installing `rEFInd` as described above was performed. After that installation the MacPro was booted into Linux.

2.1 Additions to the basic installation

Once rebooted into Linux, the network interface was tested for the network was the source of additional software. The commands:

```
ping 192.168.22.10
ping 192.168.22.7
ping 8.8.8.8
ping www.google.com.au
```

were used. In these `192.168.22.10` was the MacPro's network address, `192.168.22.7` was another Linux computer on the LAN, `8.8.8.8` was a name server on the Internet, and `www.google.com.au` was a named Internet site which in this case was used to test name resolution of the Internet address. Each of these commands performed correctly.

The process of tailoring the software began by using the Debian `apt` package server. The commands:

```
apt-get update
apt-get upgrade
```

```
apt-get install aptitude
apt-get install firmware-linux-nonfree
apt-get install rsync
apt-get install feh
apt-get install unzip
apt-get install libreadline-dev
apt-get install bluez
apt-get install aspell-en
```

were given, first to prepare the Debian package system for use, and then to download and install six packages. The `aptitude` program facilitated determining what packages were available together with selecting, loading and installing them using the Debian `apt` Internet access package system. It was an alternative to the `apt-get` command line application. Without the firmware upgrade, X Window would not work correctly on the MacPro. At this stage Linux rebooted so as to activate the firmware which had been installed.

Two package group categories were installed using the `aptitude` program from it's `Not Installed Packages` menu. From the `devel` category leading from that menu, the packages `gcc-10, g++-10, gfortran-10`, and `make` were installed. Then from the `x11` category, `xorg` and `xorg-dev` were installed. In each case the required pre-requisites were automatically loaded as well.

To make these components operate in standard ways, the following was performed. To handle the compilers the following commands were performed:

```
cd /usr/bin
ln -s /usr/bin/gcc-10 gcc
ln -s /usr/bin/g++-10 g++
ln -s /usr/bin/gfortran-10 gfortran
```

where `/usr/bin` was the directory where `aptitude` stored those components (which is the standard place). The functioning of the `gcc` compiler was tested by compiling a `hello world` program.

3 GPU installation

A Graphics Processing Unit for development and execution of parallel programs was required. This unit was in addition to the AMD Radeon GPU installed in the MacPro hardware as delivered which was to perform display of graphics only while the additional unit was to be used for parallel program use alone.

This card was to provide parallel computing alternative to the multi-core of the MacPro CPU.

3.1 Hardware considerations

GPUs from AMD and Nvidia were available. However, Nvidia GPUs supported CUDA which was widely used in parallel programming which also led onto OpenACC programming.

A MSI Geforce RTX 2700 Super Ventus card was used. MSI manufactured and marketed for Nvidia. This contained 2560 CUDA cores and 8 GB of GDDR6 memory. It was a compromise of price against performance. An MSI RTX 2800 cost 50% more but had 3072 CUDA cores with 8 GB of memory, while a MSI RTX 2080 Ti was over twice the cost with 4352 CUDA cores and 11 GB of memory. There were other cards in in RTX 2700 range, bu the card selected offered the best price per hardware combination.

The GeForce RTX 2700 card came without any software or hardware installation support. This card required a PCIe edge connector several of which were available on the MacPro 2010. The card also required a 6 pin and a 8 pin Auxiliary power connections. On the PC these are run from the power supply, but with the MacPro 2019 they are taken from connectors on the printed circuit board of the MacPro. From the Apple online store a set of AUX Belkin manufactured cables was obtained. From this set, two 8 pin to 6+2 pin cables were used to connect the 6 pin and 8 pin Auxiliary of the card to two 8 pin AUX connectors on the MacPro. The card was positioned in in PCIe socket immediately above the Radeon graphics card supplied on the MacPro; as far from the CPU as possible to enable heat removal from the card.

The video output sockets on this graphics card remained covered as they had been supplied.

3.2 Software to make the GPU function

The Nvidia CUDA software development toolkit and their driver for the GPU card were required. These were available from the Nvidia web site of from the Debian package web site; the Debian site was used. The packages were *non-free* package components. The following software installation was done after the Nvidia card had been installed in the MacPro.

First it was ensured the latest packages were to be accessed by giving the commands:

```
apt-update
apt -y full-upgrade
```

After which the MacPro Linux installation was rebooted.

A check was then made to ensure the Nvidia card was installed correctly and recognized by the hardware. The command:

```
lspci | grep -i vga
```

gave output showing both the AMD/ATI (Radeon) and Nvidia (GeForce RTX 2700) were seen. The PCIe address of the Nvidia card was given as 0d:00.0. To obtain more detail of that card, the command:

```
lspci -s 0d:00.0 -v
```

was used. At the bottom of the information produced showed the `nouveau` kernel driver and modules were linked to the card. These were open source alternatives to the Nvidia components. There Nvidia replacements were included in installation performed by the command:

```
apt install -y nvidia-driver nvidia-cuda-toolkit
```

This download/install process to some time to run to completion. Once complete, the Linux system was rebooted. This installation process had added gcc-8 and gcc-9 to the system, plus other packages which were required dependent software of the specific nvidia packages. Linux was rebooted to enable the inclusions in the installation to take effect. The command:

```
lspci -s 0d:00.0 -v
```

then showed `nouveau` kernel driver and modules had been replaced by `nvidia` as required.

A test of the resulting GPU system was confirmed by running the command (which had been installed as part of the installation)::

```
nvidia-smi
```

which gave a summary of the nvidia card on the MacPro.

Further information about the nvidia card was obtained, first by the installation:

```
apt install -y hashcat
```

and subsequently running the command:

```
hashcat -I
```

This confirmed the card's identity, and indicated a clock speed of 1785 MHz, 7882 MB of memory, and 40 processors on the card. The processing power of those 40 processors in contrast to the 28 cores on the MacPro is an open question.

4 Preparing for X Window

To use X Window (the `xorg` components of the installation) a window manager was needed. For this `dwm` was used. As a substitute for the traditional `xterm` terminal emulator, `dwm` used `st` which had to installed as well.

4.1 dwm from source

The `dwm` source file `dwm-6.2.tar.gz` was downloaded from `https://dwm.suckless.org` and stored on the `/tmp` directory. Once detarred on `/tmp`, in the file `config.mk`, `cc` was replaced by `gcc`. In the file `config.def.h`, `Mod1Mask` was changed to `Mod4Mask` which replaced the use of the `option` key with the `command` key an Apple keyboard for controlling `dwm`. Then the commands:

```
make
cp dwm /usr/local/bin
```

were used to build `dwm` and place it.

4.2 st from source

The `st` source file `st-0.8.4.tar.gz` was downloaded from `https://dwm.suckless.org` and stored on `/tmp`. Once detarred on `/tmp`, in the file `config.mk`, `# CC = c99` was replaced by `CC = gcc`, then the commands:

```
make
cp st /usr/local/bin
```

were used to build `st` and place it.

4.3 Making X usable

Into the `/root` home directory a file `.xinitrc` was prepared using the `vi` editor. The two lines:

```
exec /usr/local/bin/st -f 'Liberation Mono-14' &
/usr/local/bin/dwm
```

were placed into this file. The first line would start the `st` terminal simulating software using a 14 point font, while the second line would start the `dwm` window manager.

By using the command `startx` X Window was brought into operation with the `dwm` manager front end as dictated by the `/root/.xinitrc` file. This was done while logged in under `root`. If

other users were to load X Window in this manner, a similar `.xinitrc` file would be inserted into their home directory.

5 Bluetooth for control

The MacPro 2019 contained a Broadcom `BCM4364` chip to implemented WiFi and bluetooth. Of interest was bluetooth, particularly to support an Apple bluetooth keyboard and mouse. Both these devices could be connected to the MacPro when running OS X, but not Linux. Under OS X, the keyboard could be connected and the connection continued across reboots of OS X. This connection also continued through to `rEFInd`. However, after selection of Debian and booting of Debian, the connection ceased. It appears the `BCM4364` is a chip manufactured by Broadcom for Apple alone. This chip is also present on the Macbook Pro and iMac.

5.1 Attempt to borrow from OS X

How did OS X handle bluetooth? To find the chip-set used, on OS X the terminal command:

```
system_profile SPBluetoothDataType
```

was given. The output produced gave the chip-set as `4364B3` and the firmware version to be `v50 c4177`.

Again on OS X the terminal command:

```
ioreg -l | grep RequestedFiles
```

listed firmware files used. All of these four files were prefixed by `C-4364__s-B3/` which corresponded to the bluetooth chip-set. Of those four files, the `hanauma-X3.trx`, `hanauma-X3.clmb` and `P-hanauma_M-HRPN_V-m__n-7.9.txt` were of interest. These file, located in directory `/usr/share/firmware/wifi/X-4364__s-B3` on OS X were copied to a thumb-drive. Those files together contained the firmware used by OS X on the MacPro to initialise bluetooth. Then Linux was rebooted on the MacPro using `rEFInd`.

These three files were than saved into the `/lib/firmware/brcm` after the sub-directory `brcm` had been created on the `/lib/firmware` directory. The files were copied and translated as:

```
hanauma.trx -> brcmfmac4364-pcie.bin
hanaumma-X3.clmb -> brcmfmac4364-pcie.clm_blob
P-hanauma_M-HRPN_V-m__m-7.9.txt -> brcmfmac4364-pcie.Apple Inc.-MacPro7.,1.txt
```

Then Linux was rebooted so this firmware was executed.

The command:

```
dmesg | grep brcm
```

showed the firmware files which had been installed were loaded but initialisation of `fw_nvram` had not occurred. Bluetooth by using the provided MacPro chip-set appeared, at least at that time, not possible.

5.2 Targus dongle

An alternate approach to providing bluetooth was by using a bluetooth USB dongle. A Targus Bluetooth 4.0 Dual-Mode Micro USB Adaptor was used despite only Window 10 compatible be-

ing claimed on the packaging. All the firmware files in the /lib/firmware/brcm directory were removed. Then, while Linux was running, the Targus dongle was plugged into one of the USB-A sockets on the MacPro and the command:

```
dmesg | egrep -i 'blue|firm'
```

executed. The output produced showed the error line:

```
firmware: failed to load brcm/BCM20702A1-0a5c-21e8.hcd
```

indicating a firmware file being absent from the /lib/firmware/brcm directory.

From https://github.com/winterheart/broadcom-bt-firmware/tree/master/brcm the file broadcom-bt-firmware-master.zip containing Broadcom firmware files was downloaded and unzip applied. One of the files revealed in that collection had the name of the file reported in the error message. This file was copied unchanged into the /lib/firmware/brcm directory before rebooting Linux with the Targus dongle remaining in place. Then the dmesg command showed the bluetooth firmware loaded successfully.

Included in the bluez package downloaded using apt-get was the bluetoothctl program. The bluetoothctl program was started and the command scan on entered into it. The switch on the side of the Apple bluetooth keyboard was turned on. The message:

```
Device 80:4A:14:6F:64:F4 Ross Maloney's keyboard
```

was produced by bluetoothctl giving the MAC address of the keyboard and a description of the device to act as confirmation. Then the command pair 80:4A:14:6F:64:F4 was entered into bluetoothctl. This command was simplified as bluetoothctl uses command completion in response to a tab key being typed. In response to the bluetoothctl message of confirm passkey 014225 (yes/no): the response yes was given. Next the command connect 80:4A;14:6F:64:F4 was given, and then the command trust 80:4A:14:6F:64:F4, before the final two commands scan off and quit were given to bluetoothcrl. At this point the bluetooth keyboard was usable.

After each reboot of Linux, the above activation of the Apple bluetooth keyboard had to be repeated. To solve this problem the file /etc/bluetooth/main.conf was edited. The statements DiscoverableTimeout = 0 and Discoverable=true were placed on separate lines in the [General] part of the file content. Then the two commands:

```
systemctl start bluetooth
systemctl enable bluetooth
```

were executed. After reboot the Apple bluetooth keyboard could not be used to make a rEFInd selection, but timing out would select Debian Linux reboot. However, when Debian booted, the Apple bluetooth keyboard could be used to login.

6 R from source code

Since version 3.2.2 of the R source, curl was required for building R. However, R also is a package on the Debian apt-get package system. It also had dependencies, one of which was gcc-9. So to take the easy option of installing R via the apt-get system would have resulted in both gcc-9 and gcc-10 on the Linux system being created. This was undesirable. So building R from source code was followed.

In addition to curl, R requires bzip2 and xz in addition to curl. It also uses readline and java but they can be removed at the configuration stage of building R. However, curl depends on

pcre and openssl. So to build R from source code, other open source programs were required to be built.

6.1 bzip2

The bzip2-latest.tar.gz file containing the source code was downloaded from https://sourceware.org/bzip2/download.html and stored in the /tmp directory. It was then processed by the commands:

```
cd /tmp
tar -xzf bzip2-latest.tar.gz
cd bzip2-1.0.8
make
make install
```

6.2 pcre

There were two versions of pcre available, the pcre2 version, being the latest and suggested version was used. The pcre2-10.35.tar.gz file containing the source code was downloaded from https://ftp.pcre.org/pub/pcre and stored in the /tmp directory. It was then processed by the commands:

```
cd ../tmp
tar -xzf pcre2-10.35.tar.gz
cd pcre2-10.35
./configure
make
make install
```

6.3 openssl

The openssl-1.1.1d.tar.gz file containing the source code was downloaded from https://www.openssl.org/source and stored in the /tmp directory. It was then processed by the commands:

```
cd /tmp
tar -xzf openssl-1.1.1d.tar.gz
cd openssl-1.1.1d
./config -enable-shared
make
make install
```

6.4 curl

The curl-7.68.0.tar.gz file containing the source code was downloaded from https://curl.haxx.se/download.html and stored in the /tmp directory. It was then processed by the commands:

```
cd /tmp
tar -xzf curl-7.68.0.tar.gz
cd curl-7.68.0
./configure --with-ssl
make
make install
```

6.5 xz

The `xz-5.2.4.tar.gz` file containing the source code was downloaded from `https://tukaani.org/xz` and stored in the `/tmp` directory. It was then processed by the commands:

```
cd /tmp
tar -xzf xz-5.2.4.tar.gz
cd xz-5.2.4
./configure
make
make install
```

6.6 readline

Without `readline` the command interface to `R` was crude and thus requiring the use of `readline` in building of `R`. It was found downloading the `libreadline-dev` package from the Debian apt store was the best approach for obtaining this library and associated header files; `libreadlline` was include in the standard installation of the Debian system.

6.7 R

With an older version of `readline` available, the `R` source file `R-4.0.0.tar.gz` was processed using the commands:

```
cd /tmp
tar -xzf R-4.0.2.tar.gz
cd R-4.0.2
./configure --disable-java
make
make install
```

The resulting `R` executable worked correctly. From this basic `R`, additional packages such as `ggplot2` and `data.tables` could be included by using the `install.packages()` function from the `R terminal`.

7 LaTeX

Difficulty was encountered in using an Internet installation of the TeX system. The alternative of downloading the files onto the computer and then using those files to install the system was used. The first step was to download the files from a `CTAN` server with `rsync` capability. This was done using the command:

```
rsync -av rsync://mirror.aarnet.edu.au/pub/CTAN/systems/texlive/tlnet /tmp
```

which stored into the `/tmp` directory. When this download completed, the commands:

```
cd /tmp/tlnet
./install-tl
```

were used to perform the actual installation of the TEX package. In the dialogue following the starting of the execution of the `install-tl` script, the the option `D`, followed by option `1` enabled directory `/opt/texlive` to be selected as the destination for the installation. Then option `O` followed by option `L` was used to select `create symlinks to standard directories` with a return key typed after each of the displayed directories. This step resulted in removing the need to manually modify the `PATH` environment variable to find the TEX directories. The installation started upon giving the `[I]` option.

Both the download and installation steps took a number of minutes to complete.

8 Vim from source code

A terminal version of `vim` was required as opposed to the standard version (`vi`) included in the Debian Linux installation. This version of `vim` used the `curses` library. The `ncurses` package fulfils that requirement and it had been installed as part of the Debian installation.

8.1 Vim

The file `vim-master.zip` containing the latest source code was downloaded from `https://github.com/vim/vim/releases` and stored in the `/tmp` directory. It was then processed by the commands:

```
cd /tmp
unzip vim-master.zip
cd vim-master
./configure --enable-gui=no
make
make install
```

The preferred colour rendering was `inkpot`. The file `inkpot-master.zip` containing the source code was downloaded from `https://github.com/ciaranm/inkpot` to the `/tmp` directory. It was then processed by the commands:

```
cd /tmp
unzip inkpot-master.zip
cd inkpot-master/colors
cp inkpot.vim /usr/local/share/vim/vim82/colors
```

The personal configuration file when using `vim` was written in the home directory. This file `.vimrc` contained the statements:

```
set guioptions-=T
set autoindent
colo inkpot
syntax on
set backup
```

```
set textwidth=80
set showmatch
```

9 MariaDB from source code

The open source `MariaDB` is a fork of `mysql` designed to provide the same if not better service than `mysql`. Starting with version 5.4.0. `MariaDB` used `cmake` to build the source code. But `cmake` was not to be included in this Linux environment. So the last version (version 5.3.12) of `MariaDB` to use standard `make` was adopted for use here.

First a user for `mariadb` was created by the statements:

```
groupadd mysql
useradd -g mysql mysql
```

The file `mariadb-5.3.12.tar.gz` was downloaded from `https:/downloads.mariadb.org/mariadb/5.3.12` into the `/tmp` directory. Editing of a number of source files was necessary for the compile to be successful. After detarring the downloaded file those edits were:

- in file `client/mysql.cc`, at line 2718 the `'Ø'` was replaced by `NULL`;
- in file `server-tools/instance-manager/protocol.cc`, at line 27 the `(char)` was replaced by `(uchar)`; and
- in file `server-tools/instance-manager/instance_map.cc`, at lined 529 and 534 the `'Ø'` was replaced by `NULL`.

After this, the commands:

```
./configure --prefix=/usr/local/mysql
make
make install
```

were successful in building and installing `MariaDB`. Then the command:

```
cp support-files/my-medium.cnf /etc/my.cnf
```

was used to setup the configuration file for subsequent running of `mariadb`. Then the command sequence:

```
cd /usr/local/mysql
chown -R mysql .
chgrp -R mysql .
./bin/mysql_install_db --user=mysql
chown -R root .
chown -R mysql var
```

was used to complete the installation.

For each restart of `mariadb` (also known as `mysql`) the command:

```
/usr/local/mysql/bin/mysqld_safe --user=mysql &
```

was used to restart the server, and:

```
/usr/local/mysql/bin/mysql
```

to run the client.

10 End result

For the majority of the time this MacPro will run Linux. A system with software related specifically to our needs, and the latest version of such software was produced. It was important that the final storage of the configured system only held such software. The resulting storage distribution was:

mount point	used	available
/boot	61%	56MB
/	6%	8.2GB
/tmp	1%	18GB
/usr	77%	2.1GB
/opt	21%	28GB
/home	1%	797GB

which indicated the partitioning scheme appropriate for installing a standard system was also appropriate for the final system.

The process described here for installing Linux on a MacPro may also apply to other Mac processors of the modern type.

This Linux installation with auxiliary SSD connected to a USB C port and a GPU installed on the PCIe bus of the MacPro 2019 did not interfere with the dual booting and operation of Mac OS X.