Setting up a computer for computing research

Ross Maloney

August 2019

Abstract

Creating a Linux system containing the latest versions of `R, gcc, dwm, vim`, and TEX is described. First a standard Debian Linux distribution is selected and adapted to fulfil the *latest* requirement. Using Debian's `apt`, base components of `gcc, make` and `X11` are then added. `R, dwm`, and `vim` with all their remaining dependencies are included from open source alone. Finally the TEX system is added. A full description of each step is given. The result is a MacPro *trash can* which dual boots into Linux or MacOS.

1 Computer environment required

The computer used was a Mac Pro *trash-can* with 256GB of SSD storage. This storage was shared between MacOS and Linux.

The computer was connected to the Internet by an NBN account which give a nominal 100Mb/sec download speed. A separate computer which also accessed the Internet in the same manor was used to download the required source code files.

Linux was the environment for main use. However MacOS was kept on the computer for there were activities for which it was better suited, for example, interchange between this computer and a iPad or iPhone. Thus dual booting of Linux and MacOS was required. The dual booting was to be done using the `refind` open source boot loader.

Space on the SSD was made for the Linux environment using `Disk Utility` on MacOS, since initially MacOS occupied the whole computer. A space of 156 GB was created for Linux leaving 80 GB for MacOS. This was thought adequate for the use of each.

A Debian Linux distribution of a network (Internet) installation type was used. Such an installation enabled creating a minimum functioning Linux environment to which other components could be added to full desired needs. For this computer those needs were;

- C compiler for creating parallel processing programs;
- networking capacity;
- a Vi editor;
- R statistical package for analysis and plotting data;
- mariaDB for database work.

Both LAN networking and Internet handling was also required but without email or web browsing. Windowing was to be via X Window using the Dynamic Window Manager (`dwn`) as the interface to that system.

Strict control over the content of the system was required. Network installation provided such a control. Further, compiling software from open source also added control. But open source software can have dependencies to satisfy. This is also true in pre-built distributions such as Debian, but in Debian it provided those dependencies which were included in the distribution. Satisfying dependencies can be a challenge with open source downloads.

At the time this work was performed `Debian 10 (buster)` was the stable Debian release, having been release several weeks before. The problem was `Debian 10` used `gcc-8` as it's compiler. By using `gcc-8`, the required compilers could be produced from the source download of `GCC-9`. But the end result would be both `gcc-8` and `gcc-9` would be on the resulting Linux system, which was an undesirable result. However, Debian made available their future `Debian 11` system, known as `bullseye`, as an *unstable* system. In `bullseye`, `gcc-9` was available as a downloadable option.

2 Linux installation

The Debian network installer image `debian-testing-amd64-netinst.iso` was downloaded from `https:/cdimage.debian.org/cdimage/daily-builds/daily/arch-latest/amd64/iso-cd`. This image was then placed on a USB memory stick. The USB stick used had a capacity of 8 GB although only 2 GB would have been large enough. On a computer running Debian Linux whose USB drive was `/dev/sdc1` the command to create the installation Debian network installer on the USB stick was:

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

This command also erased the original content of the USB stick.

2.1 Setting up

A standard installation was performed. In that installation a manual `disk partitioning` was followed. The partitioning used was:

mount point	size
/	6 GB
/tmp	20 GB
/usr	5 GB
/opt	12 GB
/home	106 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 the option to proceed without such a link.

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 change 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   192.168.22.9
network   192.168.22.0
netmask   255.255.255.0
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 `192.168.22.9` 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 this network. This gateway led to a different network (`192.168.8.0`) which contained the modem connecting to the NBN/Internet . Finally, 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 a the Debian online package mirror. Linux was then rebooted for these entries to take affect.

2.2 Additions to the basic installation

Once the system was running again, the commands:

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

were given to prepare the Debian package system for use. Then the package management interface `aptitude` and some linux firmware upgrades were installed using `apt` by the commands:

```
apt-get install aptitude
apt-get install firmware-linux-nonfree
```

Without this firmware upgrade, X Window would not work correctly on the computer. 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.

Onto a USB stick the file `refind_0.11.4-1_amd64.deb` had been downloaded from `http://sourceforge.net/project/refind`. This was the program which would enable the booting of the computer into either MacOS or Linux. This USB stick was mounted on the `/srv` mount point. Then the command:

```
dpkg -i /srv/refind_0.11.4-1_amd64.deb
```

loaded and installed this dual boot loader. It was necessary to install `refind` after the `firmware` for the firmware loading appeared to remove `refind`, or at least made it inoperable.

Two package 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-9, g++-9,` gfortran-9, gccbrig-9, 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-9 gcc
ln -s /usr/bin/g++-9 g++
ln -s /usr/bin/gfortran-9 gfortran
ln -s /usr/bin/gccbrig-9 gccbrig
```

where /usr/bin is 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.

To use X Window (the xorg components of the installation) a window manager was needed. For this dwm was used. The dwm source file dwm-6.2.tar.gz was downloaded from https://dwm.suckless.org and stored on /tmp. It was then *detarred* and in the file config.mk contained in the resulting sub-directory, the line CC = cc was replaced by CC = gcc before running make. The resulting executable dwm was copied to the /usr/local/bin directory. But dwm uses the st terminal program. The st source file
texttttst-0.8.2.tar.gz was downloaded from https://st.suckless.org and stored on /tmp. This file was *detarred* and in the file config.mk contained in the resulting sub-directory the line # CC = c99 was changed to CC = gcc before running make. The resulting executable st was copied to the /usr/local/bin directory.

Into the texttt/root home directory a file .xinitrc was prepared using the vi editor. The single line /usr/local/bin/dwm was placed into this file.

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.

3 R from source

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-8. So to take the easy option of installing R via the apt-get system would have resulted in both gcc-8 and gcc-9 on the Linux system. 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 must also be built.

3.1 bzip2

The bzip2-1.0.6 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 bzip2-1.0.6
cd bzip2-1.0.6
./configure
make
make install
```

3.2 pcre

The `pcre-8.43.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 pcre-8.43.tar.gz
cd pcre-8.43
./configure -enable-unicode-properties -enable-pcre16 -enable-pcre32 \
    -enable-pcregrep-libz -enable-pcregrep-libbz2
make
make install
```

3.3 openssl

The `openssl-1.1.1c.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-1.1.1c.tar.gz
cd openssl-1.1.1c
./config -enable-shared
make
make install
```

3.4 curl

The `curl-7.65.3.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-7.65.3.tar.gz
cd curl-7.65.3
./configure --with-ssl
make
make install
```

3.5 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
```

3.6 R – attempt 1

With the dependencies built and install, `R` could be built. The `R-3.6.1.tar.gz` file containing the source code was downloaded from `https://cran.r-projecr.org/src/base/R-3/` and stored in the `/tmp` directory. It was then processed by the commands:

```
cd /tmp
tar -xzf R-3.6.1.tar.gz
cd R-3.6.1
./configure --without-readline --disable-java
make
make install
```

The `--without-readline` option was used because of errors which `configure` otherwise showed before terminating. Those errors were related to a symbol `UP` being undefined. Because `java` was not available on the computer, the `--disable-java` option was used.

The resulting R executable worked. However, in the `R terminal`, editing functions such as `up arrow` from the keyboard to retrieve previous typed command lines did not work.

3.7 readline

The error problem when configuring `R` with `readline` included was related to the latest version (version 8.0) of `readline`. So the previous version (7.0) `readline-7.0.tar.gz` was downloaded from `ftp.gnu.org/gnu/readline`, after which the commands:

```
cd /tmp
tar -xzf readline-7.0.tar.gz
cd readline-7.0
./configure
make
make install
```

were used to generate the `readline` library.

3.8 R – attempt 2

With an older version of `readline` available, the `R` source file `R-3.6.1.tar.gz` was processed using the commands:

```
cd /tmp
tar -xzf R-3.6.1.tar.gz
cd R-3.6.1
./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`.

4 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 completed, the commands:

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

were used to perform the actual installation of the TeX package. In the dialogue following starting the execution of the `install-tl` script, the `/opt/texlive` directory was selected as the destination for the installation. Also in the dialogue, option `O` was used to select `create symlinks to standard directories`. This step resulted in removing the need to manually modify the `PATH` environment variable to find the TeX directories.

5 Vim

A terminal version of `vim` was required as opposed to the standard version (`vi`) included in the Debian Linux installation. This version of `vim` required to use of a `curses` library. The `ncurses` package fulfils that requirement.

The file `ncurses6.1.tar.gz` which contained the source code was downloaded from `https://ftp.gnu.org/gnu/ncurses/` and stored in the `/tmp` directory. It was then processed by the commands;

```
cd /tmp
tar -xzf ncurses-6.1.tar.gz
cd ncurses-6.1
./configure
make
```

```
make install
```

The file `vim-8.1.1888.tar.gz` 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
tar -xzf vim-8.1.1888.tar.gz
cd vim-8.1.1888
./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/vim81/colors
```

6 End result

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

mount point	used	available
/	12%	4.6GB
/tmp	1%	19.0 GB
/usr	38%	2.7GB
/opt	62%	4.1 GB
/home	1%	94.0 GB

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

The surprise was a Debian system which was available with the latest software, notable that of the GCC collection. GCC version `gcc-9` on the `apt` Debian `bullseye` distribution was the same as the latest version on the `https://ftp.gnu.org/gnu` GNU home site. In time GCC version 9 will be on the *stable* distribution of Debian as opposed to the *testing* distribution, the contents of which can change over night.

Unfortunately a lot of the Web material which supposes to address building `R` from source, does `R` but not building the software necessary for building `R` instead downloading pre-built versions of such software. Learning the dependencies of `R` and the dependencies of those dependencies, and building each dependency was one of the challenges of this work. It is worth putting in the public domain the solution found.