NetBSD for the Advanced Minimalist

rambius

ABSTRACT

This document describes a NetBSD setup that relies on as few dependencies as possible. It is equally suitable for small laptops and for bigger machines.

1. Why Going Minimal?

The short answer is: because we can. The long answer is that more software needs more maintenance. As BSD users we are spoilt - it takes two commands on NetBSD to upgrade all your packages:

```
$ sudo pkgin update
$ sudo pkgin upgrade
```

Still it does requires time and effort. Hence the question - why do I need to update something I don't use.

1.1. My Path to Minimal or No Dependencies

My first laptop ran FreeBSD with KDE. I enjoyed it but KDE upgrades sometimes failed - back then I had time to fix them. Then during a NYCBSD meeting G. R. told me he was using XFCE. I tried it and it was a little bit lighter and required less dependencies. Next, I worked on a project where I did not have access to a window manager, but I managed to set up a convenient working environment, see my talk at NYCBUG from 2019-09-04.

Then out of curiosity I bought a Pinebook laptop and installed NetBSD on it. I did not want to do any extensive configuration of a window manager and I didn't want to deal with many updates. I started using it mainly in text-mode.

Finally in December 2023 I was travelling in Europe and I was stranded for a day in Germany because of a snow storm. I messaged and emailed (from my phone) my family over several channels to let them know that I was ok. While I was waiting for flights I asked myself how much of that communication I could have done from that Pinebook laptop. How much of my daily work can be done on almost vanilla NetBSD? I took out the laptop, a pencil and a paper notebook and started working on that idea.

This talk is the product of that effort.

2. NetBSD Installation on the Pinebook

The Pinebook comes with Android on its 16GB eMMC storage. I never removed the Android because the machine can boot from an MicroSD card. There is a NetBSD live image that can be written on an microSD card.

These links are useful:

- https://wiki.netbsd.org/ports lists what NetBSD port supports what CPU and architecture;
- https://wiki.netbsd.org/ports/aarch64 describes the port that runs on the Pinebook;
- https://cdn.NetBSD.org/pub/NetBSD/NetBSD-9.3/evbarm-aarch64/ contains the binaries for that port; and finally
- https://cdn.NetBSD.org/pub/NetBSD/NetBSD-9.3/evbarm-aarch64/INSTALL.html contains the installation instructions.

The installation consists of downloading the NetBSD image, transfering it to an SD card, and making the SD bootable. I am using another NetBSD machine for these steps.

2.1. Downloading the NetBSD Image

Downloading the image is as easy as

```
$ curl -sS -C - https://cdn.NetBSD.org/pub/NetBSD/NetBSD-9.3/evbarm-aarch64/binary/gzimg/arm64.img.gz -o arm64.img.g
$ gunzip arm64.img.gz
```

2.2. Transfering the image to a microSD Card

The next step is to copy the uncompressed image to a microSD card. You need a microSD card, an SD card reader and depending on the reader an SD card adapter. Put the the microSD card either in the adapter and then the adapter in the reader or in the reader directly. Plugin the reader in the machine where you downloaded the image. Find in dmesg what device the microSD card maps to. Write the image to that device:

```
$ sudo dd if=arm64.img of=/dev/rld0c bs=1m conv=sync
```

2.3. Making the SD Bootable

The Pinebook uses the U-Boot bootloader. You need to put the U-Boot's Secondary Program Loader on the microSD card so that it knows how to boot NetBSD. NetBSD provides a package with the U-Boot's SPL for Pinebook:

```
$ sudo pkgin install u-boot-pinebook
$ cd /usr/pkg/share/u-boot/pinebook
$ sudo dd if=u-boot-sunxi-with-spl.bin of=/dev/rld0c bs=1k seek=8 conv=sync
```

The dd commands can be dangerous. Make sure you have the correct output devices especially if you run them on a Pinebook - you may destroy your boot microSD card.

2.4. Booting the Pinebook

Put the microSD card and turn on the laptop. It loads the NetBSD OS from the card. The image is configured to resize the NetBSD partition to fill the whole card and to resize the / slice to fill the whole partition. It reboots again and the system is ready.

There were certain rough edges after my first boot. The image enables only one tty that the system uses to send the console messages. Those messages mangled my input. I got a flood of errors about the wirless adapter I had plugged in. I unplugged it so that I can use the tty.

3. Initial Configuration

We now need to create the initial user, give them permissions to further configure the system, enable the network, etc.

3.1. Setting the root's Password

The image's original root password is the empty string. We login as root and we change it immediately.

```
login: root
Password:
# passwd
```

3.2. Adding a Non-root User

Working as root all the time is dangerous. We need to create a non-root user who can run commands as root. The command su switches identities. A user can run it if they are in the wheels group:

```
# useradd -m -G wheel rambius
# passwd rambius
# exit
login: rambius
Password:
$ su -
Password:
#
```

After that change we will never login directly as root.

3.3. Configuring the ttys

The image enables only one tty that the system use to send the console messages. Edit /etc/ttys to enable the additional ones:

```
$ su -
# vi /etc/ttys
# kill -s HUP 1
# exit
$ exit
```

Now we switch with Ctrl-FN to ttyN.

3.4. Configuring the Wireless

Now we plug the wireless adapter back in. The wpa_supplicant daemon is responsible for the wifi connections. We have to configure it and start it. Its configuration file is /etc/wpa_supplicant.conf where we put the network's details, namely its SSID and the password:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=wheel
update_config=1

network={
        ssid="myssid"
        psk="mypass"
        scan_ssid=1
}
```

Next we configure it to start on boot by adding the following line it to /etc/rc.conf:

```
wpa_supplicant=YES
```

We either reboot or we start the daemon manually

```
# service wpa_supplicant start
# ping nycbug.org
# ifconfig
```

3.5. Configuring the Package Manager pkgin

I said that I would use a minimal amount of third-party packages; I did not mean no packages at all.

The command pkg_add installs binary packages pre-built for the corresponding architecture. I prefer to use pkgin because of its searching capabilities. I use pkg_add to install pkgin only.

pkg_add needs to know where it can find the packages. The environment variable PKG_PATH specifies their location.

```
$ su -
# PKG_PATH=http://cdn.netbsd.org/pub/pkgsrc/packages/‘uname -s‘/‘uname -p‘/‘uname -r‘/All
# export PKG_PATH
# pkg_add pkgin
```

I initially tried PKG_PATH with https, but it failed and I switched to http.

‘uname -s‘ returns the OS’s name, in this case NetBSD , ‘uname -p‘ returns the architecture name, for example aarc64, ‘uname -r‘ returns the OS’s version. The reason for so many parameters is that the packages are built for many architectures and for several versions.

pkgin is almost ready now. We need to adjust its repositories list in its configuration file /usr/pkg/etc/pkgin/repositories.conf and fetch the list with remote packaes. I only had to remove the hard-coded values for the OS’s version and architecture:

```
# tail -1 /usr/pkg/etc/pkgin/repositories.conf
https://cdn.netbsd.org/pub/pkgsrc/packages/NetBSD/$arch/$osrelease/All
# pkgin update
```

3.6. Installing and Configuring sudo

The first package we will install with pkgin is sudo, because we want to not use su. sudo can run only a specific command as root in a non-root session thus decreasing the possibilities for errors.

```
# pkgin install sudo
```

The configuration file /usr/pkg/etc/sudoers specifies what users and groups can run what commands. It already has a commented entry that allows members of the wheel group to run any command. We don’t edit the file directly; instead we use the visudo command.

```
# visudo
```

We uncomment the line

```
%wheel ALL=(ALL:ALL) ALL
```

We exit the su session and from now on we use only sudo for any command that requires elevated permissions.

Usually in a more restricted environent you can only run specific commands with specific arguments. To view these commands use -l option of the command sudo:

```
$ whoami
rambius
$ sudo -l
User rambius may run the following commands on arm64:
   (ALL : ALL) ALL
```

For a complete treatment of sudo read the book Sudo Mastery by Michael Lucas.

This concludes the initial configuration of the system. Next we will discuss various programs and applications.

4. Program and Applications

We can now do a lot of useful tasks with our small laptop, for example:

- write a book with groff;
- organize our expenses with awk (it’s the tax season);
- develop with cc, make, etc.

What follows is a list of programs and application that I use regularly or find them interesting.

4.1. The "Intergrated Development Environment" tmux.

My user interface of choice is tmux. It is in NetBSD's base system and does not require separate installation. It is a terminal multiplexer that allows splitting the screen in several horizontal or vertical panes. My usual development style is to have 3 panes - one for the code I work on, one for its unit test and one with a shell where I compile and test. When writing this document with nroff I use only two panes - one to write it in vi and one to view it. I put the processing commands in a Makefile and I call make from vi. You can also open several windows in tmux and have as many panes as you want in them. Think of windows as tabs.

The biggest advantage of tmux is that you can detach from your session and then attach to it later. The sessions also survives network and VPN disconnects. For example

```
$ ssh sdf.org
sdf$ tmux
# Open windows and panes
# Ctrl-b d to detach or network drops later:
$ ssh sdf.org
sdf$ tmux attach
# and continue your session
```

tmux also has a status line where it can dispay data about the current session or about the system. That brings us to the next topic.

4.2. Battery Monitoring and envstat

Laptops get plugged and unplugged all the time especially if you have cats or dogs. The command envstat shows the supported devices' sensor values:

```
$ envstat
                              Current  CritMax  WarnMax  WarnMin  CritMin  Unit
[axppmic0]
               ACIN present:     TRUE
               VBUS present:    FALSE
            battery present:     TRUE
                   charging:    FALSE
               charge state:   NORMAL
            battery voltage:    4.176                                       V
     battery charge current:    0.001                                       A
  battery discharge current:      N/A
            battery percent:      100                                    none
   battery maximum capacity:    4.844                                      Ah
   battery current capacity:    4.843                                      Ah
[sunxithermal0]
            CPU temperature:   43.000                                    degC
          GPU temperature 1:   44.000                                    degC
          GPU temperature 2:   44.000                                    degC
```

The device axppmic0 has a sensor that reads the battery percentage. We can extract the value with awk:

```
$ envstat | awk '/battery percent/ { printf("%d%%", $3) }'
```

For convenience we put the command in a shell script ~/bin/bat.sh and we can show its output in tmux's status line. tmux's behavior is controlled by its options and the command

```
$ tmux show-options -A
```

lists all of them. The option status-right specifies the contents of the right side of the status line. We create ~/.tmux.conf and set the status-right option in it:

```
$ cat ~/.tmux.conf
set -g status-right "\"#{=21:pane_title}\"%H:%M %d-%b-%y #(bat.sh)"
```

The notation #(<command>) show the command's output in the option's value. After we change ~/.tmux.conf we restart tmux and can observe the battery percentage.

4.3. Email with mutt

Mutt is a text-based email client. We install it with

```
$ sudo pkgin install mutt
```

We will demonstrate how to use it with a gmail account. Enable the two-factor authentication in your Google account. Then generate an App Password for mutt and put the it in ~/.mutt/.gmailpass. Next, you need to create a mutt configuration file in ~/.muttrc:

```
set ssl_starttls = yes
set ssl_force_tls = yes

set from = "me@gmail.com"
set realname = "Me Me"

set imap_user = "me@gmail.com"
set imap_pass = `cat ~/.mutt/.gmailpass`

set folder = "imaps://imap.gmail.com"
set spoolfile = "+INBOX"
```

The folder variable specifies the location of the mailboxes ; the spoolfile one defines the user's inbox where the mail arrives. This setup is enough if you want to read your email.

I have thousands of messages in my inbox. Mutt was slow on start because it had to fetch them every time. However, it can cache the messages and their headers locally:

```
set header_cache = "~/.mutt/cache/headers/"
set message_cachedir = "~/.mutt/cache/bodies/"
```

If the header_cache variable specifies a directory, we have to create it in advance.

By default mutt sorts the email by the date received. If you want the latest on top use

```
set sort = "reverse-date-received"
```

Now we are ready to configure mutt to send mail. We need the following entries in ~/.muttrc:

```
set smtp_url = "smtps://me@gmail.com@smtp.gmail.com"
set smtp_pass = "$imap_pass"
```

We can test sending email from the command line:

```
arm64$ echo "test from mutt" | mutt -s "test from mutt" me@gmail.com
TLSv1.3 connection using TLSv1.3 (TLS_AES_256_GCM_SHA384)
No authenticators available
Could not send the message.
```

The first attempt failed. mutt depends on cyrus-sasl to do the authentication. That package does not install any autenticators. After some trial and error I discover it needs cy2-login:

```
$ sudo pkgin install cy2-login
$ echo "test from mutt" | mutt -s "test from mutt" me@gmail.com
TLSv1.3 connection using TLSv1.3 (TLS_AES_256_GCM_SHA384)
```

It sends email successfully now. This conclude mutt’s configuration.

4.4. Internet Relay Chat

NetBSD provides several IRC clients. You can search for them with

```
$ pkgin search irc
```

You can search for packages by their names or by any pattern.

We install irssi:

```
$ sudo pkgin install irssi
```

We will demonstrate how to connect to #nycbug channel at libera.chat.

4.4.1. Nickname Registration

You need to register your nickname with libera. If you have done so you can skip it.

```
$ irssi
/set nick <yournick>
/network
/connect liberachat
/msg NickServ register <yourpass> <youremail>
```

You will receive an email with a verification command. It looks like

```
/msg NickServ verify register <yournick> <token>
```

You type it back in irssi and your nickname is registered.

4.4.2. Connectiong to a Channel

All IRC commands start with a "/". /network lists the predefined networks in irssi. liberachat is there and we can /connect to it. Next we register the nickname and then we verify the registration. Now we are ready to join #nycbug channel:

```
$ irssi
/connect liberachat
/msg NickServ identify <yournick> <yourpass>
/join #nycbug
```

The /msg command identifies us with the password we registered. We are now ready to send messages to the channel.

4.5. Web Browsing

There are few text-based browsers. I have been using lynx for quite some time and as expected few sites render well in it. It does not support JavaScript and if a site needs it then it will not work. elinks has some JavaScript support but I have never tried it. w3m can display images on suitable terminals.

To install lynx run

```
$ sudo pkgin install ca-certificates
$ sudo update-ca-certificates
$ sudo pkgin install lynx
```

The package ca-certificates contains the root certificates of several certificate signing autorities. Without them lynx throws https-related errors when it opens https sites and curl refuses to work with them.

To use lynx run

```
$ lynx https://nycbug.org
```

Don’t expect first-class experience with text-based browsing. I tried to log into GitHub with lynx to copy a repository's URL - I could not.

Still there is an interesting application I want to show. When we open any graphical web browser we could just type a query and it searches the Internet for it with its configured search engine. Lynx is interactive but we can automate it with expect. Consider the following script:

```
#!/usr/pkg/bin/expect -f
if { $argc == 0 } {
   puts "$::argv0 <search query>"
   exit 1
}
spawn lynx https://lite.duckduckgo.com/lite
expect "Search"
send "\t"
send "$argv"
send "\r"
expect "<return>"
send "\r"
interact
```

This tells expect to open DuckDuckGo in lynx (or whatever search engine we prefer), wait for the term "Search" to appear on the screen, send a tab to focus the text field in the search form, send the search query and press Enter two times. For example:

```
$ sudo pkgin install tcl-expect
$ ./duck.ex "nycbug dmesg"
```

I used a similar approach to automate a http login session from lynx and download a file. I could not use curl, because it was using cookies.

4.6. Sound and Sound Applications

The command audiocfg list the available sound devices:

```
$ audiocfg list
0: [*] audio0 @ ausoc0: sun50i-a64-audi
      playback: 2ch, 48000Hz
      record:   2ch, 48000Hz
      (PR) slinear_le 16/16, 2ch, { 48000 }
1: [ ] audio1 @ ausoc1: hdmi-audio
      playback: 2ch, 48000Hz
      record:   unavailable
      (PR) slinear_le 16/16, 2ch, { 48000 }
```

The device with * is the default one. The same command can test it by playing a tone:

```
$ audiocfg test 0
0: [*] audio0 @ ausoc0: sun50i-a64-audi
      playback: 2ch, 48000Hz
      record:   2ch, 48000Hz
      (PR) slinear_le 16/16, 2ch, { 48000 }
  testing channel 0... done
  testing channel 1... done
```

Initially I did not hear anything because the volume was 0. The command mixerctl shows and control the volume and othe sound settings:

```
$ mixerctl -a
outputs.master=0,0
outputs.mute=off
outputs.source=headphones
inputs.line=0,0
inputs.headphones=0,0
record.line=0,0
record.mic=96,96
record.mic.preamp=off
record.mic2=96,96
record.mic2.preamp=off
record.agc=96,96
record.source=
$ mixerctl outputs.master
outputs.master=0,0
```

The option -a shows all settings. Providing a setting's name shows its value. The option -w can set a value:

```
$ mixerctl -w outputs.master=50
outputs.master: 0,0 -> 48,48
$ audiocfg test 0
0: [*] audio0 @ ausoc0: sun50i-a64-audi
      playback: 2ch, 48000Hz
      record:   2ch, 48000Hz
      (PR) slinear_le 16/16, 2ch, { 48000 }
  testing channel 0... done
  testing channel 1... done
$ mixerctl -w outputs.master++
outputs.master: 48,48 -> 52,52
```

If test command plays its tones we have working audio.

4.6.1. Playing Music

The command audioplay can play .wav files. https://samplelib.com/ contains test files in various formats without any license that can be used for testing. I downloaded several wav files and tested audioplay with them:

```
$ mixerctl outputs.master
outputs.master=200,200
$ audioplay *.wav
```

WAV stores the audio information in uncompressed form without information loss and the files are larger. On the other hand MP3 uses lossy data compression. mpg123 plays MP3 files. I installed it and downloaded several mp3 files from https://samplelib.com and played them:

```
$ mpg123 *.mp3
```

mpg123 is a versatile player. You can create playlist, play random tracks, loop, etc.

It is instructive the see the size ratios of the WAV and MP3 counterparts. I wrote a script that calculates them:

```
$ cat src/wav_to_mp3.sh
#!/bin/sh
cd ~/downloads
printf "Name\t\tWAV\tMP3\tRatio\n"
for len in `seq 3 3 15`; do
  name=sample-${len}s
  mp3_size=`stat -f "%z" mp3/${name}.mp3`
  wav_size=`stat -f "%z" wav/${name}.wav`
  ratio=`echo "${wav_size} / ${mp3_size}" | bc`
  printf "%s\t%s\t%s\t%s:1\n" ${name} ${wav_size} ${mp3_size} $ratio
done
$ src/wav_to_mp3.sh
Name        WAV  MP3    Ratio
sample-3s   563756      52079     10:1
sample-6s   1127468     103070    10:1
sample-9s   1691180     154062    10:1
sample-12s  2254892     205470    10:1
sample-15s  3382316     307453    11:1
```

The MP3 compression resulted in ten-fold disk space decrease and in loss of quality. If you have the raw audio file you can control the quality versus size when you compress it similar to the text compression utilities like gzip. Data compression is a really interesting topic and I hope somebody gives a talk about it.

4.6.2. Podcasts

The podcasts are radio programs made available for downloads over the Internet. Usually the producer provides an RSS feed and the clients fetch the feed, parse it and download the content and play it now or later. We will use as examples BSDNow's and BBC's feeds.

The package that downloads the podcasts is podcastdl:

```
$ sudo pkgin install podcastdl
```

We create a file that contains the feeds' URLs and we call podcastdl with it. Once it has finished we play them with mpg123.

```
$ cat podcasts.cfg
https://feeds.fireside.fm/bsdnow/rss
http://downloads.bbc.co.uk/podcasts/radio4/fooc/rss.xml
$ podcastdl -c podcasts.cfg -o . -d 3
$ mpg123 *.mp3
```

The options -o specifies the directory the podcasts will be downloaded to, and -d - the period in days.

4.7. External Media

Every now and then we still need USB sticks and CDs. This section shows how to initialize them and mount them. The process is similar to initializing a hard drive and deserves attention.

4.7.1. USB Sticks

Plugin the stick and check in dmesg how the system recognize it:

```
$ dmesg
[ 127060.584075] sd0 at scsibus0 target 0 lun 0: <PNY, USB 3.2.1 FD, PMAP> disk removable
[ 127060.595366] sd0: fabricating a geometry
[ 127060.595366] sd0: 115 GB, 118236 cyl, 64 head, 32 sec, 512 bytes/sect x 242147328 sectors
[ 127060.615008] sd0: fabricating a geometry
```

It recognizes it as sd0 and its device entry is /dev/sd0. Next we write 0s to its 1MB:

```
$ sudo dd if=/dev/zero of=/dev/rsd0c bs=1m count=1
1+0 records in
1+0 records out
1048576 bytes transferred in 0.347 secs (3021832 bytes/sec)
```

Note that we specify the output device, that is the USB stick as /dev/rsd0c. That means we access the raw device of the corresponding device files. The raw devices provide direct access bypassing the operating system's buffers and caches.

Next we need to create an partition on the USB stick. The partition can be of almost any filesystem provided the systems that will mount it support it. We can use an MSDOS partition as it is the the most popular type. The command that creates partions is fdisk:

```
$ sudo fdisk -au sd0
```

The program fdisk prompts for input. In this session we tell it to change partition 0. We specify its filesystem type as MSDOS (sysid 11), set its boundaries - make it occupy the whole USD drive and make it active.

Next we need to initialize the filesystem:

```
$ sudo newfs_msdos /dev/rsd0e
```

We have fully initialized the USB drive and we can start using it:

```
$ sudo mount_msdos /dev/sd0e /mnt
$ cd /mnt
$ sudo touch myfile
$ sudo cp ~/file.txt .
$ cd
$ sudo umount /mnt
```

Note that we have to use sudo to mount the stick and for all the commands that write to the USB stick. This contrasts with other operating systems that automatically mount everything plugged into them. In the past the setting the vfs.generic.usermount sysctl to non-zero allowed a non-root user to mount external devices. Now it cannot be enabled if the kern.securelevel sysctl is above 0.

4.7.2. CDs and DVDs

I had no luck with external USB CD / DVD drives on the Pinebook. I have an Apple USB SuperDrive. When I plugged it it refused to accept any CD. I tried with another external CD drive I got from Amazon. I was able to put a CD in it, but it refused to spin.

I used another NetBSD machine for the two examples below.

4.7.2.1. Mounting a Data CD

Mount a data CD is similar to mount an USB drive -just need to change the filesystem and the device:

```
$ sudo mount_cd9660 /dev/cd0a /mnt
$ ls /mnt
$ cp /mnt/somefile ~
$ sudo umount
$ sudo chown me ~/somefile
```

4.7.2.2. Ripping a Music CD

We need two packages to extract the music tracks from a CD. cdparanoia extracts them and saves them to WAV files.

```
$ sudo pkgin install cdparanoia
$ sudo cdparanoia -B
$ chown me *.wav
```

Next we need an encoder, for example bladeenc, to convert them to MP3s.

```
$ sudo pkgin install bladeenc
$ bladeenc *.wav
```

Optionally we could fetch the data about the CDs from CDDB. There are two command-line rippers in NetBSD packages collection that were supposed to do it, but they use the old FreeDB that does not exist anymore and they crash. That is a good opportunity to send patches or create a new package.

4.8. Typesetting

There are at least 3 text processing systems on Unix - groff, TeX / LaTeX and Docbook. Their main characteristics is that they separate content from presentation and thus the content can be transformed into several formats. Their input is usually plain text and can be easily version-controlled.

Docbook defines the content in XML format. Then XSLT transformations convert it to HTML, PDF, man pages, etc.

TeX / LaTeX is popular for math typesetting. TeX defines low-level formatting. LaTeX defines more logical structures like books, articles, section, subsections, table of contents, bibliographies, etc. The output can be DVI, PS and PDF. TeX Live is a cross-platform distribution for TeX typesetting available as a NetBSD package.

We will discuss in more detail the nroff / groff programs. They are part of the base system. Their input consists of plain text with formatting commands. Furthermore they are programmable and macros can group formatting commands creating logical elements of the text. For example, instead of explicitly centering a line and making it bold a macro can declare it as a title. This provides a great amount of flexibility - if we redefine a macro but preserve its logical name we can change the document appearance significantly. The macros are grouped in packages.

This document uses groff with the ms package and I will share my experience. The structure of a groff document is

```
.TL
Title
.AU
Author
.AB
Abstract
.AE
.NH
Numbered Heading
.PP
Paragraph
.NH 2
Numbered Subheading
.PP
Paragraph
.DS
Verbatim
.DE
.NH
Next Numbered Heading
```

B The general form of the command that renders the document

```
$ groff -ms -T<output> doc.tr > doc.<out>
```

The command that renders it for the terminal similar to man is

```
$ groff -ms -Tascii doc.tr > doc.out
$ less doc.out
```

Here is my development cycle. I put the groff command in a Makefile:

```
advmin.out: advmin.tr
        groff -ms -Tascii ${.ALLSRC} > ${.TARGET}
```

In a tmux window I have two panes. In the right one I open the groff document in vi. In the left one I view the output with less. I run :!make vi command to run groff and then in the pane with less I press R to refresh it.

In the final stages I generate the ps output:

```
advmin.ps: advmin.tr
        groff -ms -Tps ${.ALLSRC} > ${.TARGET}
```

There are many way to convert the ps output to a pdf. I chose to use the command ps2pdf from the package ghostscript.

```
$ sudo pkgin install ghostscript
$ ps2pdf advmin.ps
```

As a make target that looks like:

```
advmin.pdf: advmin.ps
        ps2pdf ${.ALLSRC}
```

Of course when I view it I need X and a PDF reader like mupdf.

5. Open Problems