Getting Started With Plan 9

ABSTRACT

This is a collection of notes about using Plan 9, mainly pointers to helpful manual pages, papers, and examples. It also describes some things you should do to set up the environment after you get Plan 9 running.

Documentation

The manual is in two parts, Volume 1 and 2. The 'manual pages' are in Volume 1, reside in `/sys/man`, and can be accessed on the web. Volume 2 contains the more prosey 'documents', which are stored in `/sys/doc` and may also be accessed on the web.

The window system and editors

The Plan 9 window system `rio` is similar in interface to the previous Plan 9 window system, 8½, as well as Unix's `mux` and the `9wm` X window manager. Unless you have experience with one of these, `rio` is likely to be unfamiliar.

The manual page *rio*(1) is the definitive reference, and a good introduction. Be aware that in Plan 9, the left, middle, and right mouse buttons are often referred to as buttons 1, 2, and 3. The paper in `/sys/doc/8½` is also helpful.

Plan 9 has two graphical text editors: `acme` and `sam`. `Sam` is a fairly traditional multi-file editor based upon structural regular expressions, while `acme` takes a more radical approach to almost everything and is a little better integrated with some aspects of the system, namely the window system and plumbing. A full introduction to either is beyond the scope of this document, but the papers in Volume 2 of the manual contain good introductions. The manual pages *acme*(1) and *sam*(1) are also helpful here.

Input

There is no real 'tty driver' in Plan 9. Instead, the window system handles input processing. The erase character is backspace, line kill is control-U, and `Delete` interrupts the process; these cannot be changed. Beware: if you're running without the window system, on the bare console, although backspace and control-U still work, `Delete` has no effect (a consequence of having the window system do the processing).

To users familiar with Windows, it may seem strange that windows are not selected or drawn with button 1. The convention is that the button used to start the operation (such as to pop up the menu) is the button used to finish it. Clicking any other button before finishing the operation will abort it. For example, if you select `Delete` from the button 3 menu and then click with button 1 or button 2, even on a window, nothing happens. If you find yourself accidentally dragging a window around by its border, keep the button you are pressing held down and click one of the others; the window goes back to its original place. The same behavior can be expected of the text editors.

The command shell

The shell `rc` is similar to the Bourne shell for simple redirection and starting processes in the background. After that, the syntax diverges greatly, but is simple and easy to understand. The paper in Volume Two is an easier introduction than the manual page *rc*(1), but the latter is worth reading for details. The directory `/rc/bin` has lots of examples.

There is no job control, either in the shell or the window system; just draw a new window.

Users and permissions

There is no super–user; the closest equivalent is the person who booted the terminal (generically called Eve; Adm owns the file server). Most devices are owned by Eve, and the local kernel will let Eve do most things commonly associated with a super–user (for example, debug or kill processes she doesn't own). Eve's power does not extend past the local machine, though, or even into the `kfs` file system. The important difference is that the `kfs` file system is being provided by a user process, which has its own permissions checking separate from the kernel, and it does not care to let the hostowner have special permissions directly.

Of course, permissions need to be bypassed once in a while. For example, the file `/adm/users` (the equivalent of a combination of Unix's `/etc/passwd` and `/etc/group`) is not writable except by members of group `adm`. Rather than put yourself into the group `adm`, we usually type `disk/kfscmd allow` to turn off permission checking on the `kfs` file server, edit `/adm/users`, make `kfs` read it back by typing `disk/kfscmd user`, and then turn permission checking back on by typing .`disk/kfscmd`disallow (If you're running a network file server, the preferred but rarer setup, all this hair–raising stuff is avoided; appropriate control is provided on the file server console and nowhere else.)

To add a new user, add the user to the `/adm/users` file and then run the `user` command; the format of the file is documented in *users*(6). (Note that on a standalone file server, the `newuser` command manages users, to avoid turning off permission checking.) The *fs*(8) and *kfscmd*(8) manual pages explain more file system commands.

To create a new user, you'll want to add a line like

```
282:tor:tor:
```

to `/adm/users`, run `kfscmd user`, and then run

```
disk/kfscmd 'create /usr/tor tor tor 775 d'
disk/kfscmd 'create /mail/box/tor tor upas 775 d'
disk/kfscmd 'create /mail/box/tor/mbox tor upas 622 al'
```

to create `tor`'s home directory and mail box. Now you'll want to halt the disks and reboot to log in as `tor`; the first thing to do then is run `/sys/lib/newuser` to set up a profile and start the window system.

Note that you had to reboot to log in as a different user; this is because the kernel must authenticate to the file server in order to obtain startup files like `init` or its startup script, and the only way for a terminal to get authentication credentials is to have a user type a name and password. Of course, for `kfs` this could be sidestepped, but this is necessary when booting from a real file server on the network.

Startup

When a Plan 9 machine boots, it runs the configuration script `/rc/bin/termrc` or `/rc/bin/cpurc`, depending on whether it is a terminal or a CPU server. These set up `/dev` and initialize some common environment variables, in particular

$fileserver. This variable is used by lp, among other programs. For a standalone configuration, this should be kfs, as it is when installed; if you set up a real Plan 9 file server and boot over the network, change this to be the name of the network file server.

Network Databases

Plan 9 uses a single database to handle all network programs and services, including BOOTP, DHCP, and DNS. The database is located in /lib/ndb and controlled by /lib/ndb/local. That file lists a number of other files to include as part of the database:

```
database=
        file=/lib/ndb/local
        file=/lib/ndb/common
```

You can add as many files as you like, but due to a limitation in libndb, you need to restart all network services after changing the set of files. In practice, the easiest way to do this is usually to reboot the system. (You can, of course, make changes to individual files without rebooting; it is only necessary when you add or remove a file from the database= list). By convention, common holds information that is not site-specific, like TCP assigned port numbers. Local and the other files hold site-specific configuration.

The database contains entries for individual systems as well as for entire networks. An entry is simply a set of lines without whitespace between them, like:

```
ipnet=moscvax-cs-net ip=10.0.20.0 ipmask=255.255.255.0
        ipgw=10.0.20.1
        dns=10.0.20.2
        dns=10.0.20.3
        dns=10.0.30.2

ip=10.0.20.30 sys=lusitania ether=00a02124062f
        dom=lusitania.cs.moscvax.edu
```

For the most part, this last entry is treated as equivalent to

```
sys=lusitania
        ip=10.0.20.30
        ether=00a02124062f
        dom=lusitania.cs.moscvax.edu
```

or anything like it; items on the same line are bound more tightly than those on separate lines, but most programs ignore this information. The indentation is significant: a line with no initial whitespace begins a new entry.

When searching for an attribute of a particular system, programs first look at that system's entry (actually all of them, if there are more than one). Then the IP network is checked, and then subnetworks. Information found in smaller subnetworks overrides larger ones. Consider the following database:

```
ipnet=moscvax-net ip=10.0.0.0 ipmask=255.0.0.0
        ipsubmask=255.255.255.0
        dns=dns-server
        ntp=ntp-server
        wins=10.0.26.122
        wins=10.0.161.254
        dnsdomain=moscvax.edu

ip=10.0.1.2 sys=dns-server
ip=10.0.1.3 sys=ntp-server

ipnet=moscvax-math-net ip=10.0.0.0 ipmask=255.255.255.0
        ipgw=10.0.0.1
        ntp=math-ntp-server

ip=10.0.0.2 sys=math-ntp-server

ipnet=moscvax-cs-net ip=10.0.20.0 ipmask=255.255.255.0
        ipgw=10.0.20.1
        dns=cs-dns-server
        cpu=cpu-server
        fs=file-server
        dnsdomain=cs.moscvax.edu
        dnsdomain=moscvax.edu

ip=10.0.20.2 sys=cs-dns-server
ip=10.0.20.3 sys=cpu-server proto=il
ip=10.0.20.4 sys=file-server proto=il

ip=10.0.20.30 sys=lusitania ether=00a02124062f proto=il
        dom=lusitania.cs.moscvax.edu
        cpu=other-cpu-server
```

Suppose the database is being queried to find a DNS server to use on lusitania. First lusitania's entry will be checked, finding nothing. Then, since lusitania's IP address is a Class A address (with default netmask 255.0.0.0), the entry for the 10.0.0.0 network `moscvax-net` will be checked. Note that `moscvax-math-net` is ignored because it is the entry for a 10.0.0.0 network with netmask 255.255.255.0. A DNS server will be identified for `moscvax-net`, but since it also has a `ipsubmask` entry, we go down to the next smaller network, `moscvax-cs-net`, and also find one there, so we use that one instead. Similarly, if we had been searching for an NTP server, we would have found and kept the one defined in `moscvax-cs-net`. If a machine is running Plan 9, and thus the IL protocol, it should be marked as such in the database; otherwise it will be treated as only using TCP and UDP.

The program `ndb/ipquery` is useful in checking to see that your databases say what you want them to say.

IP Configuration

The program `ip/ipconfig`, run by `/rc/bin/termrc` at startup, configures an Ethernet adapter, if any; if you have not defined a network database entry that has your Ethernet address and supplies IP address (`ip`), IP gateway (`ipgw`), and IP network mask (`ipmask`), `ipconfig` will use DHCP to attempt to acquire this information. If you have no DHCP server running on your Ethernet, it will sit for a minute before it times out. You'll also very likely want to define `dns` entries pointing at your DNS servers, so that `ndb/dns` will be of help in resolving names.

The DNS resolver uses automatic suffix addition to translate unknown names; the `dnsdomain` database entries specify a list of suffixes to try. In the above example, a host in moscvax−cs−net will translate alice by first looking for alice, then for alice.cs.moscvax.edu, and then alice.moscvax.edu. Note that as with other values, the entries do not chain: if there were no dnsdomain=moscvax.edu line in the moscvax−cs−net network entry (but at least one dnsdomain= line), adding moscvax.edu (as specified in the higher level moscvax−net entry) would not be tried.

Network Daemons

Network daemons are started by *listen*(8), the equivalent of Unix's inetd. Listen has no configuration file; instead it watches a directory full of daemon scripts named protoNNN where proto is il or tcp and NNN is a port number. The default directory is /rc/bin/service.

For security reasons, a number of network daemons are disabled in the distribution. The relevant files have an underscore _ prefixed to their names. These provide interfaces to services, such as print spoolers, that you should probably only turn on once you have the security structure established on your environment. To turn on the services, rename the files to remove the leading underscore.

Reading mail

To read mail on Plan 9, you currently need a Plan 9 system that runs an SMTP server and spools your messages; attempting to read messages from a mounted Unix or NFS file system will not work, as Plan 9 uses different mechanisms to lock mailboxes.

There are a few ways around this, though; one is to have a program download your mail from a POP3 server and hand it off to the Plan 9 mail programs to save in a local mailbox, and then edit your mail that way. Alternate solutions include writing a POP3 or IMAP4 client that presents your mail as a file system, as *fs*(4) does for mailboxes. A version of *fs*(4) that supports POP3 mailboxes exists but is still being soaked; it may appear before long.

If you are going to run an SMTP server, you should edit the files /mail/lib/smtpd.conf and /mail/lib/blocked to configure it.

Looking in the other direction, Plan 9 comes with POP3 and IMAP4 servers.

Sending mail

To send mail from Plan 9, you need to configure the outgoing mailer; its main configuration file is /mail/lib/rewrite, which is supplied as an empty file. The manual *rewrite*(6) is worth reading. You'll want to start by copying either rewrite.gateway or rewrite.gateway+default from the same directory and editing it to suit. Setting the smtp= attribute in your network database is all that is necessary to use rewrite.gateway, which sends all mail containing an @ sign to your local mail gateway. Mail to unqualified names (names without @*somewhere*) will still be delivered to local mailboxes. If you would like all mail to unqualified names to have a default domain added, start with rewrite.gateway+default and edit to suit.

The last rule in the rewrite files calls /mail/lib/qmail, which both queues the message and starts a daemon to try to deliver the messages currently in the queue (see *qer*(8) for more details). On systems not always connected to the internet, you may wish to use /mail/lib/justqmail instead, which only queues the message, and does not start a delivery daemon. When you are connected to the internet or your mail gateway, you can run /mail/lib/kickqueue to have the daemon try to send mail.

/mail/lib/remotemail, which actually delivers mail via SMTP, contains a default domain name used for unqualified outgoing mail; you will want to change it from yourdomain.dom to something more appropriate.

Each local user has a mail directory /mail/box/*$user*; among other things, it usually contains a mailbox and a headers file; the contents of the latter are included in all outgoing messages from that user. To add a "full name" field to your outgoing mail, add

```
From: "Glenda" <glenda>
```

to the headers file.

SSH

To use the Plan 9 SSH tools, you must generate a host key for your system. To do this, run aux/ssh_genkey while permission checking is off; this will create the files /sys/lib/ssh/hostkey.public and /sys/lib/ssh/hostkey.secret; be sure that hostkey.secret is not readable by any user but the one used to run network listeners.

You can also use ssh_genkey to create a key for yourself:

```
cd $home/lib
aux/ssh_genkey -d $user
cat $user.secret >> userkeyring
chmod 400 userkeyring
```

The file $user.public10 created in your personal lib directory is suitable for sharing with Unix machines, e.g., for putting in your authorized_keys file on a Unix machine.

Printing

The Plan 9 print spooler lp does not take the same arguments as the BSD lpr, nor as the lp that ships with some Unixes. It is worth reading the manual page to get up to speed on useful options. The file /sys/lib/lp/devices contains example devices for printing to printers on parallel ports or BSD lpr-based queues.

Lp sends Postscript to the named destination, so if you are using a parallel printer it will need to be a Postscript printer. If you have a printer supported by Ghostscript, you can use lp to generate Postscript and then send it via Ghostscript:

```
% man -t man | lp -dstdout |
        gs -s'DEVICE=bjc600' -s'OutputFile=/dev/lpt1data' \
                -dBATCH -dQUIET -dNOPAUSE -
%
```

TeX

The Plan 9 documentation is in troff; however, TeX has been compiled and brought up under the ANSI/POSIX Environment (APE). It is a separate package from the main distribution due to its size. You can download the archive tex.9gz from the Plan 9 web server and then install it by typing wrap/inst tex.9gz. This is a TeX distribution that uses the kpathsearch modifications; the main library tree is kept in /sys/lib/texmf.

A real Plan 9 network

Plan 9 is not intended to be run only on a standalone machine; if you have a couple machines to use, once you are comfortable with the system you are encouraged to set up a dedicated file and cpu server, and try running your terminals with no local state other than their bootstrap programs.

Configuring a standalone cpu/authentication server

When configuring a Plan 9 network, the first machine to set up is a standalone cpu and authentication server. After you have this, you can go on to configure a file server and then boot terminals and other cpu servers from it.

Start by installing the distribution as though creating a standalone terminal. Reboot the system, and log in as `glenda`.

Decide what name to give your cpu server owner. This is the user that all the cpu servers run as. We'll name the user `bootes`. Choose a password for `bootes`. (Just keep it in your head for now.) Edit `/adm/users` as described above to create any users you want the *kfs* file server to know about.

Edit `/rc/bin/cpurc`. Just before `ndb/cs`, add a call to `ip/ipconfig`, specifying the appropriate numbers if you don't want to use DHCP:

```
ip/ipconfig -g gateway.addr ether /net/ether0 ip.addr ip.mask
```

If DHCP will not get the system name correct, you may wish to add `sysname=`*system* just before the `ip/ipconfig` line. Uncomment the calls to `ip/dhcpd` and `ip/tftpd` if you want to run those servers. (Note that early copies of the distribution have a `cpurc` that says `ip/dhcp` rather than `ip/dhcpd`; if yours does, you should correct this.) Replace the two `aux/listen` lines with

```
auth/keyfs -wp -m /mnt/keys /adm/keys >/dev/null >[2=1]
aux/listen -q -d /rc/bin/service -t /rc/bin/service.auth il
aux/listen -q -d /rc/bin/service -t /rc/bin/service.auth tcp
auth/cron >>/sys/log/cron >[2=1] &
```

to start the key server, the network listeners, and cron. You'll probably want to uncomment the call to `aux/timesync` as well. Bind in some extra devices by adding

```
for (i in '#m' '#i' '#S' '#t')
        bind -a $i /dev >/dev/null >[2]/dev/null
```

to the top of `cpurc`, just after the `facedom=` line. If you leave out `'#m'`, *rio* will not start when you boot your server (sometimes a desirable property).

Generate some SSH keys for the host:

```
aux/ssh_genkey
chgrp -u bootes /sys/lib/ssh/hostkey.*
```

Edit the contents of `/lib/ndb` to fit your network, as described above. Be sure to list `auth=`*authserver* as an attribute in your network's entry. If you are going to use NTP for `timesync`, make sure you define an `ntp=`*ntpserver* attribute as well. It's a good idea to check that you got your network config right by trying

```
ndb/ipquery ip your.ip.add.ress auth
ndb/ipquery ip your.ip.add.ress cpu
ndb/ipquery ip your.ip.add.ress ntp
```

Edit `/lib/ndb/auth` to say that the cpu server user is allowed to become any other user (given the appropriate credentials). A sample file might be

```
hostid=bootes
        uid=!sys uid=!adm uid=*
```

Optionally, configure your mail setup (described above) by editing `rewrite`, `remotemail`, `smtpd.conf`, and `names.local`, all in `/mail/lib`.

Remove `/rc/bin/service/il566` and `/rc/bin/service/tcp567`. They proxy calls for the authentication services to the authentication server. On the authentication server itself, you'll want to run different services on those ports. To run them,

```
cd /rc/bin/service.auth
mv authsrv.il566 il566
mv authsrv.tcp567 tcp567
```

Compile and install a disk-based cpu server kernel:

```
cd /sys/src/9/pc
mk 'CONF=pccpudisk' 9pccpudisk
9fat:
cp 9pccpudisk /n/9fat/9pccpud
```

Note that you can't call it `9pccpudisk` because the bootstrap loader does not know about long FAT filenames. Edit `/n/9fat/plan9.ini` to say `bootfile=sdXX!9fat!9pccpud`, where `sdXX` is the disk with the `9fat` partition.

Create an `nvram` partition on your disk. (At present, this only works if you make it on `sd00` or `sdC0`; we will fix this real soon. You can change this by editing `/sys/src/9/boot/key.c`.) Use `disk/prep`: the `nvram` partition need only be a single sector, which you can easily steal from the beginning or end of your swap partition. For good measure,

```
echo blahblahblah >/dev/sdC0/nvram
```

so that the checksum won't be correct when you next boot. This will force the cpu server to ask you for authentication information.

Reboot the machine. When it comes up, it should load the `9pccpud` kernel and then complain about the nvram checksum being incorrect. Having done this, it will prompt for a password, authentication id, and authentication domain. Enter `bootes`'s password and `bootes` for authentication id. The authentication domain is not used, except for debugging. It is often an internet domain like `moscvax.edu`. It must not be blank.

Once *rio* comes up, create some user accounts in the key database by using `auth/changeuser` (see *auth*(8)). In particular, be sure to create an account for `bootes` using the same password you just typed.

At this point you should be all set. Test out a new user account by attempting to telnet into the auth server as that user, using `netkey` to respond to the challenge.

Note that you have configured a standalone cpu server whose primary purpose is as an authentication server. A set of patches to *kfs* exists to make it act as a network file server, so that you could use one machine to support the network rather than needing two. This is not the best way to run a Plan 9 system — *kfs* is slow and can't provide *dump*-style backups, and it's easier to administer the network when each machine has only one job. Even so, for small installations (one or two people) this is a decent way to get terminals booting without any local state. The *kfs* patches are currently being soaked and may appear before long.