README

Brian W. Kernighan
bwk@research.att.com

This brief document is intended to help you get started using Plan 9. It is written by a casual Plan 9 user who is not in any way part of the Plan 9 group, and is aimed at ordinary users with a UNIX background.

Getting Started

I'm assuming that you or someone you trust has read, understood, and performed the instructions in the Plan 9 installation procedure. I'm also assuming that you have at least looked at the overview paper, Plan 9 from Bell Labs, which explains what Plan 9 does and how it goes about it. There are also some helpful explanations and examples in the paper The Use of Name Spaces in Plan 9; both these papers are in this volume. I am further assuming that you have the Plan 9 manual, Volume I, near to hand, and are willing to read manual pages for commands as their names appear here.

How you get started after you turn your terminal on depends on how your Plan 9 system is set up; the details are different for a stand-alone system or a terminal connected to a file server and a CPU server. The latter is mostly what I'll talk about.

In much the same way that .profile is executed by the shell on UNIX systems, the file lib/profile is executed by the shell on Plan 9 when you log in. When your account was created, someone ran the newuser command, which created a few directories (bin, bin/rc, lib, and tmp) in your home directory $home, then set up a profile lib/profile that looks like this, though with more frills:

```
bind -a $home/bin/rc /bin
font = /lib/font/bit/pelm/euro.9.font
switch($service){
case terminal
        prompt=('term% ' '      ')
        exec 8½
case cpu
        bind -b /mnt/term/mnt/8½ /dev
        prompt=('cpu% ' '      ')
}
```

Many of the interesting bits of Plan 9 are implicit in this file; most important, it starts the window system.

The 8½ Window System

The window system for Plan 9 is called 8½; the terminal case in the profile starts 8½ with the line

```
exec 8½
```

8½ provides less of the flexibility and certainly far fewer features than xterm on X terminals, but I prefer it.

The most important difference is that 8½ treats all text on the screen alike; with the mouse you can edit anything you can see, so you can fix up and resubmit commands, fiddle the output of a program or its input and resubmit it, and so on. This ability to edit the past is liberating to such a degree that once you use it, you'll never want to go back to something like xterm.

8½ can be called recursively; you can make a new window, run 8½ in it, and everything you do there is insulated from the surroundings. 8½ does not provide any analog of the virtual window management of, for example, VTWM, nor does it provide zillions of (or even a few) icons, but you can move a window almost off the screen, and you can hide it and then recall it from a pop-up menu, which is loosely equivalent to iconifying it.

You can also ask 8½ to run a file of commands when it starts; most people put this request into their profile:

```
exec 8½ -i lib/windows
```

Normally this is used to set up windows that you always use:

```
#!/bin/rc
window 'x0 y0 x1 y1' command line
...
```

where *x0,y0* and *x1,y1* are the coordinates of the window in question (and *y* increases down the screen). The window command opens a window at the specified place, then runs the command in it. The command wloc will tell you the names and locations of all windows, in the right format to be inserted directly in a file. Set up the windows and programs the way you want them, then run wloc and snarf its output.

Commands

Most of the standard UNIX commands exist in almost the same form on Plan 9; this includes standbys like cat, ls, cd, pwd, cp, mv, diff, grep and awk. You'll notice minor differences in behavior but for the most part you don't have to think about this.

The rc Shell

One big difference: Plan 9 uses a different shell, called rc. For running commands interactively, it's almost the same as the Bourne or Korn shells, so filename metacharacters like * and ? behave the same, and simple redirection with > and >> and | are the same. Quoting is simpler: double quotes and backslash have no special meaning, and single quotes quote anything. To get a single quote into a quoted string, double the quote:

```
echo ''''
```

prints a single quote.

For programming, rc is almost unrelated to sh, which is a nuisance. In particular, control flow operators like if and for have a different syntax.

All shell scripts have to begin with

```
#!/bin/rc
```

Environment variables are set by

```
var = 'anything'
```

where the quotes can be omitted if *anything* contains no spaces. Environment variables are accessed as $var; certain variables are initialized when your process begins, including user (your name), home (your home directory), and service, which is terminal when you are running on your terminal and CPU when you are running on a CPU server.

Directories and Search Paths

One of the unifying ideas in Plan 9 is that all resources are accessed as file systems. Central to this is management of the name space so that you can select and arrange the resources you want to use.

The `bind` and `mount` commands manipulate the name space. In particular, the command

```
bind -a $home/bin/rc /bin
```

in the profile binds the directory `$home/bin/rc` after (`-a`) the directory `/bin`, forming a *union directory.* (More precisely, it makes `/bin` an alias for the union.) Other than the current directory, the shell searches only `/bin` for commands to run, but it searches all the directories that have been unioned together. By convention, your personal shell scripts are placed in `$home/bin/rc`.

When you first log in, several directories are bound to `/bin`, including `/rc/bin`, which contains shell scripts, and `/$cputype/bin`, which contains binaries for your current CPU type. The variable `cputype` is the type of processor you are running on, typically one of `386`, `sparc`, `mips`, or `68020`. When you run a program such as `ls`, the version for your current CPU type will be found in `/bin` and executed. If you subsequently execute the `cpu` command to access a CPU server, in that process and those started by it, `cputype` will be the type of the CPU server, and the `ls` command (again in `/bin`) that you run there will be the right binary for that CPU.

This mechanism of union directories replaces the search path of conventional UNIX shells. As far as you are concerned, *all* executable programs are in `/bin`. Try

```
lc /bin
```

to see the names of all executable programs.

Interesting File Systems

UNIX users are familiar with the idea that devices like disks and tapes are part of the file system. Plan 9 carries that idea a lot further. If you look at the directory `/dev`, you will see some familiar names. Try

```
cat /dev/time
```

a couple of times, for example. Or, after you have snarfed some text using the button 2 menu item, try

```
cat /dev/snarf
```

Note that some files like `cons` and `mouse` occur more than once. `/dev` is a union directory, and these are multiple occurrences of the same file. The first `/dev/mouse` refers to the current window, and the next one to the enclosing window (which is probably the whole screen at this point). Try

```
cat /dev/mouse
```

then move the mouse around inside and outside of the current window.

The shell environment is kept in a directory called `/env`; each shell environment variable is stored in a file. Try

```
cat /env/font
```

for example.

Running processes are found in `/proc`; each process is a directory, and each file in that directory accesses some aspect of the process. For example, the `status` file contains (textual) status information about the process. Try

```
awk '{print $1}' /proc/*/status
```

to get a list of the names of the running processes.

You might also find it interesting to poke around in /net; all network connections are managed as file systems as well. In all of these cases, the service presents a file-system interface to its clients, although the implementation behind is not generally a traditional file system.

Finally, it's worth looking at /sys, which is a conventional file system that contains directories of source code, libraries, headers, manual pages, documentation, and the like. This roughly matches similar directories found on UNIX systems.

Fonts

One aspect of 8½ that you can change is the font it uses for displaying text. There is a default font, but normally the font variable is set explicitly in the profile:

```
font = /lib/font/bit/pelm/euro.9.font
8½ -f $font
```

The font euro.9.font is a collection of almost any character you might find in European languages, including Cyrillic, Greek, and a bunch of special characters. There are other fonts that include oriental languages as well, and a variety of sizes.

Plan 9 uses the Unicode character set throughout, which means that the system and the various programs all deal comfortably with a very large character set. (Think 16 bits, or 64K characters.) So if you want to edit files in languages that use more than ASCII characters, or run grep or awk over them, it just works. You may have trouble printing such characters on standard printers, but they will appear fine on the screen.

Editing

The standard Plan 9 editor is called sam; it's a particularly good multi-file editor, it provides regular expression syntax the same as in the venerable ed (which also exists), and you can snarf text from one of its windows and paste it into other 8½ windows or vice versa. The mouse idioms for sam and 8½ are the same. It will also edit files on other systems if there is a network connection.

By the way, regular expressions have been cleaned up: all programs except rc support the same regular expressions, which are pretty close to those found in egrep on UNIX systems.

The CPU Server

In the Plan 9 world view, one is meant to run interactive programs such as editors on the terminal and compute-intensive programs such as compilers on a CPU server, which runs faster and has a higher bandwidth to the file server. The cpu command connects you to a CPU server so your computation runs faster, but everything else stays the same. The mechanism is quite different from either remote login (which does not preserve the name space you are currently working in) or network file system access (which does not change the process). The line

```
bind -b /mnt/term/mnt/8½ /dev
```

in your profile arranges that all the devices (including mouse, keyboard and screen) associated with your terminal are inherited by the CPU server so they continue to work in a CPU window.

Connecting to UNIX Systems

It is likely that your Plan 9 system will be connected by some network to a UNIX system. The command `con` connects to another system (typically UNIX); the command `rx` is rather like the `rsh` command on UNIX systems, for executing a single command on another machine.

If the UNIX system cooperates, it is also possible to mount a UNIX file system in the Plan 9 name space so that files on the UNIX side are accessible from Plan 9. The command

```
9fs machine
```

establishes the connection and mounts the files; thereafter the root of the target file system is in the Plan 9 directory `/n/`*machine*.

Plan 9 CPU servers answer FTP, `rlogin`, and `telnet` requests. If you want your own personal access privileges, you will need to reply to a challenge using a SecureNet key or equivalent; otherwise, user `none` is permitted unchallenged access, sufficient to access many globally available services and databases (see section 7 of the manual). Similar restrictions apply when accessing Plan 9 file servers over NFS; in this case, the program *9auth* initiates the challenge/response dialog.

If your Plan 9 machine shares a disk with MS-DOS, as it might on a PC, you can access the DOS file system through `/n/c:`, and other disks as `/n/a:`, etc. This is a convenient way to get information into and out of the PC world.

Backup and Recovery

Normally the state of the Plan 9 file system is recorded every day or so; on our system, it's stored on an optical disk. If your Plan 9 system is suitably equipped, you should be able to run another service that makes the past state of the file system accessible (read only). For example, if you run the command

```
9fs dump
```

it mounts this file system on `/n/dump`. At that point, you can `cd` into the past:

```
cd /n/dump/1995/0401/usr/you
ls -l
```

puts you in your directory as it was on April 1, 1995. This really is a file system, so all the normal commands work fine; you can `diff` a file from then with one on some other date, or copy an old version to the present. Plan 9 has no backup or recovery programs; this mechanism subsumes them all.

Programming in Plan 9

Most programming in Plan 9 is done in ANSI C, with the usual supporting tools like YACC available. One difference of note: `make` has been largely supplanted by `mk`, which is cleaner but different. As with the shell, it takes time to internalize the differences.

For each supported CPU type, there is a C compiler to generate code (named using a single letter mnemonic), with a version of that compiler that may be executed on any CPU type. The `mkfile` normally encapsulates this, and `/sys/src/cmd` contains examples that you can adapt easily.

Although ANSI C is supported, the Plan 9 libraries are not ANSI and the standard ANSI header files normally are not found. Compiling C programs is different enough that you should read the paper called How to Use the Plan 9 C Compiler before starting.

If you are importing or exporting a C program, you will want to use the ANSI/POSIX environment (APE), which really does provide for portability, including a complete set of POSIX-compatible libraries and some POSIX tools. The compiler driver is called pcc. The command

```
ape/psh
```

will bind the right files and start a POSIX-compliant shell.

Although these POSIX tools are useful for exchanging programs with the outside world, you will generally be more productive using the native Plan 9 tools where possible.

Envoi

Plan 9 is not UNIX. If you think of it as UNIX, you'll often be frustrated because something doesn't exist or works differently. If you think of it as Plan 9, however, you'll find that most of it works very smoothly, and that there are some truly neat ideas that make things much cleaner than you have seen before.