

A Plan 9 Newbie's Guide

Michael A. Covington

for
Coraid, Inc.

2008 May 28

1. Introduction

1.1. What is this document?

This is a quick-start guide to the user interface of the Plan 9 operating system based on my own experiences getting started at Coraid. For concreteness, it describes the setup actually used at Coraid. Some knowledge of UNIX and of commercial GUIs (Windows, MacOS, or recent Linux) is presumed.

This is *not* a technical guide to Plan 9. This is only a "guide for the perplexed" to teach you **just enough Plan 9** to get you started.

The information given here is not complete - my goal is to give you a useful subset. Thus I will skip some menu items and commands at every stage, the ones you do not need immediately. I assume that you will eventually read the official documentation.

1.2. What is Plan 9?

Plan 9 is an experimental UNIX-like operating system developed by Bell Labs and released as open-source freeware. It has requirements comparable to Windows 95 (e.g., 32 MB RAM on a 486) and is often used in embedded systems.

1.3. *Unicibus ipsis Unicior*

That is bad Latin for, "More UNIX than the UNIXes themselves" (cf. *Romanis ipsis Romanior*) and sums up what Plan 9 is like. Plan 9 is based on the same ideas as early UNIX - simplicity, compactness, and orthogonality - but implements some of them differently, in ways that are (I would argue) closer to the spirit of UNIX than normal present-day UNIX implementations. Specifically:

- Instead of NFS, SAMBA, etc., there is normally just one file-sharing protocol, called 9P, and it is transparent to the user. You just carry your whole

namespace from one CPU to another. When you use a terminal program such as *drawterm*, your local PC is mounted as subdirectories of */mnt/term* alongside whatever is already there.

- More of the OS and hardware are presented to you in the filesystem. For example, */dev/screen* is the screen, */dev/mouse* is the mouse, and */dev/time* is the clock. The environment variables are "files" in */env*. Control settings can be made by writing on special files rather than through system (IOCTL) calls.
- The native character set is UTF-8, i.e., ASCII plus multi-byte encodings for (much of) Unicode. Some software (ported from UNIX) still recognizes only ASCII.
- The GUI is very simple - an exemplar of how to build a GUI with the minimum amount of programming and hence the maximum amount of reliability.

In general, in Plan 9 there will not be multiple ways to do the same thing. Thus you can have more confidence that you have found the best way to do what you're doing. As in UNIX, the main architectural principle is **orthogonal combinations of simple features**.

1.4. Gotchas

Mouse usage is obligatory. You cannot use the *acme* editor, for instance, without frequently taking your hand off the keyboard.

A three-button mouse is expected. If necessary, you can simulate the middle button by holding down Shift and pressing the right button. In general:

- The left button is for moving the cursor or selecting text;
- The middle button is for making menu choices;
- The right button is for higher-level control operations;
- The scroll wheel, if present, scrolls the text up and down.

Further details depend on the software you are running.

In this document I will call the buttons left, middle, and right, but Plan 9 normally calls them 1, 2, and 3.

On the keyboard, the \uparrow and \downarrow keys do not move the cursor one line up or down. Instead, they scroll the displayed text like PgUp and PgDn but not as far. There is no key that moves the cursor up or down one line. (The \leftarrow and \rightarrow keys *do* move the cursor left and right.)

The Delete key does not delete; it interrupts execution (like Ctrl-C or Break in other operating systems). To delete text, select it and hit Backspace. (On the

Macintosh, the Delete key in the main key group is Backspace, and the one near Home and End is Delete.)

In the Acme editor, the *Del* menu selection does not delete the selected text – it deletes (closes) the edit window! Use *Cut* to delete the selection (and copy it to the snarf buffer).

1.5. Logging in and out

The login process is similar to UNIX. Your terminal is likely to be, itself, a workstation running Plan 9. If you use a PC terminal program such as *drawterm*, be sure to tell it the authentication server as well as the server to which you want to log in.

There is no logout command. Just exit your application software and disconnect or turn your workstation off.

1.6. Jargon

The following terms are not common outside the Plan 9 milieu:

chord	simultaneous pressing of more than one mouse button (used in Acme)
put	save to disk.
rune	a Unicode character (corresponding to one or more UTF-8 bytes)
snarf	copy to the snarf buffer (comparable to the Windows clipboard)
sweep	to move the mouse across text while holding down a button
zerox	duplicate a window with its contents

2. The *rio* windowing system

When you log on to Plan 9, you are in (text) console mode, but normally your profile, in file `/usr/username/lib/profile`, will immediately launch *rio*, the windowing system (which replaces an earlier windowing system called 8½).

Rio tends to baffle newcomers because its initial display is a blank gray screen (Figure 1).

To make things start happening, right-click anywhere on the gray surface and choose *New* (Figure 2). Release the right mouse button, then press it again and drag it across the screen, marking out a rectangle as you do so. Voilà – you have a window (Figure 3). You can have as many as you want. Left-click in any

window to bring it to the front.

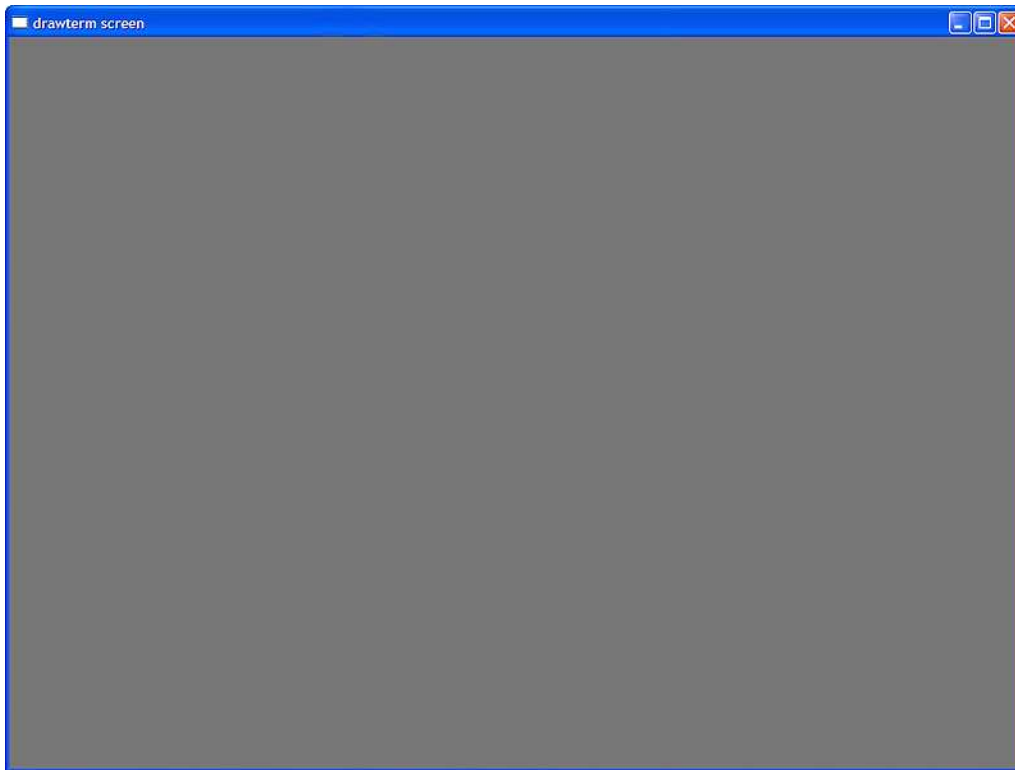


Figure 1. *Rio* opening screen (in a terminal window) – yes, it’s all blank gray.



Figure 2. Right-click on New, then right-drag to create a window.

2.1. Keyboard usage

There is no "terminal control" or "cursor control" in *rio*. The window and the keyboard are simply streams of characters. In the window, a substring of the stream can be selected (just as in any other GUI), and if the selection is zero-length, it serves as the cursor or insertion point (again, just like any other GUI). What you type always replaces the selected text.

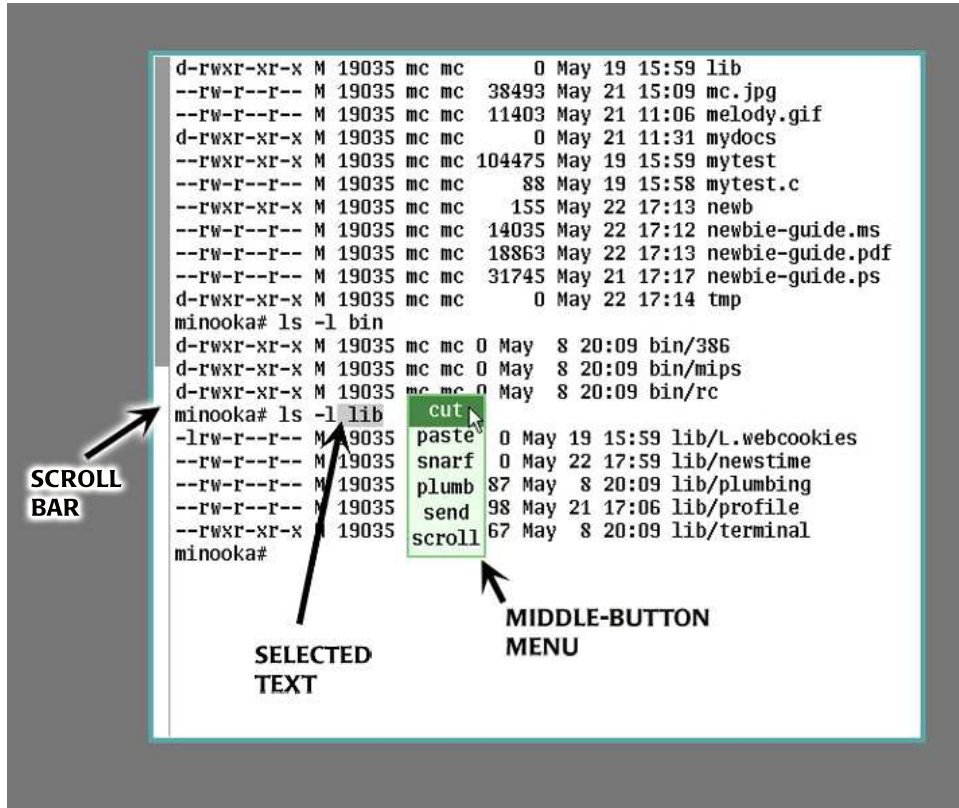


Figure 3. A window in *rio*.

The → and ← keys move the cursor left or right. Ctrl-A moves it to the beginning of the line, or rather to the input point (i.e., the first place on the line that you can type); Ctrl-E moves it to the end. Backspace deletes one character to the left; Ctrl-U deletes from the cursor to the end of the line; Ctrl-W backspaces a whole word instead of just one character.

2.2. It doesn't scroll!

One of the biggest differences between *rio* and other kinds of console windows is that in *rio*, when the output runs off the bottom of the window, the window doesn't scroll. Instead, the process that is writing the output will block and wait for you to scroll down by hitting ↓ or PgDn or by using the mouse.

This is like having | more built into the output of everything that sends characters to the screen. If you really don't like this behavior, just middle-click in the existing window and choose Scroll, or alter your profile to launch *rio -s* instead of plain *rio*.

2.3. Mouse usage in *rio*

Left-click to bring any window to the front.

Left-drag the edges or corners of any window to resize it.

If the window holds more text than you can see, then there is a scroll bar at the left of it (Fig. 3). With the mouse on the scroll bar, you can left-click to move up, right-click to move down, or middle-drag to move up and down. You can also scroll with the ↑ and ↓ keys.

To make a window go away, right-click in it, choose Delete, and right-click on it again.

The left button is also used to select text. Click once, and you move the cursor (which is a zero-length selection) to the place where you clicked. Double-click, and you select the whole word. To select more than a word, click and drag. Or, to select a string of words that are in quotes or parentheses, double-click the closing quote or parenthesis.

As in other GUIs, when you type, your typing will replace what is selected.

The middle button brings up the following menu:

- cut** Delete the selection and put it in the snarf buffer (clipboard).
- paste** Paste the contents of the snarf buffer at the current insertion point.
- snarf** Copy the selection into the snarf buffer.
- plumb** Send the selection to the *plumber*, an interprocess communication system.
- send** Paste the contents of the snarf buffer into the window after the output point (not the current selection, but the end of the text in the window) and simulate pressing Enter. This is a quick way to send a command to the system.
- scroll** Toggle whether the window automatically scrolls when more text is written at the bottom.

2.4. Hold mode

Normally, just as in other operating systems, typed input is sent to the running program whenever you press Enter. However, in "hold mode," many lines of input are held for you to edit, until you give the go-ahead to process them.

In hold mode, the type on your screen turns blue. Hold mode is started and stopped by pressing Esc. Some programs, such as *mail*, turn on hold mode

automatically while you type your input; when sending e-mail (for example) you must press Esc to exit hold mode and press Ctrl-D to simulate end of file.

The right button, we have already discussed; it is mainly for creating and removing windows. For more about *rio*, type the command *man rio*, which works just like the UNIX *man* command.

3. The *acme* full-screen editor

Inspired by Niklaus Wirth's Oberon system, *acme* is a minimalist full-screen editing and development system. In operation, it looks something like Figure 4.

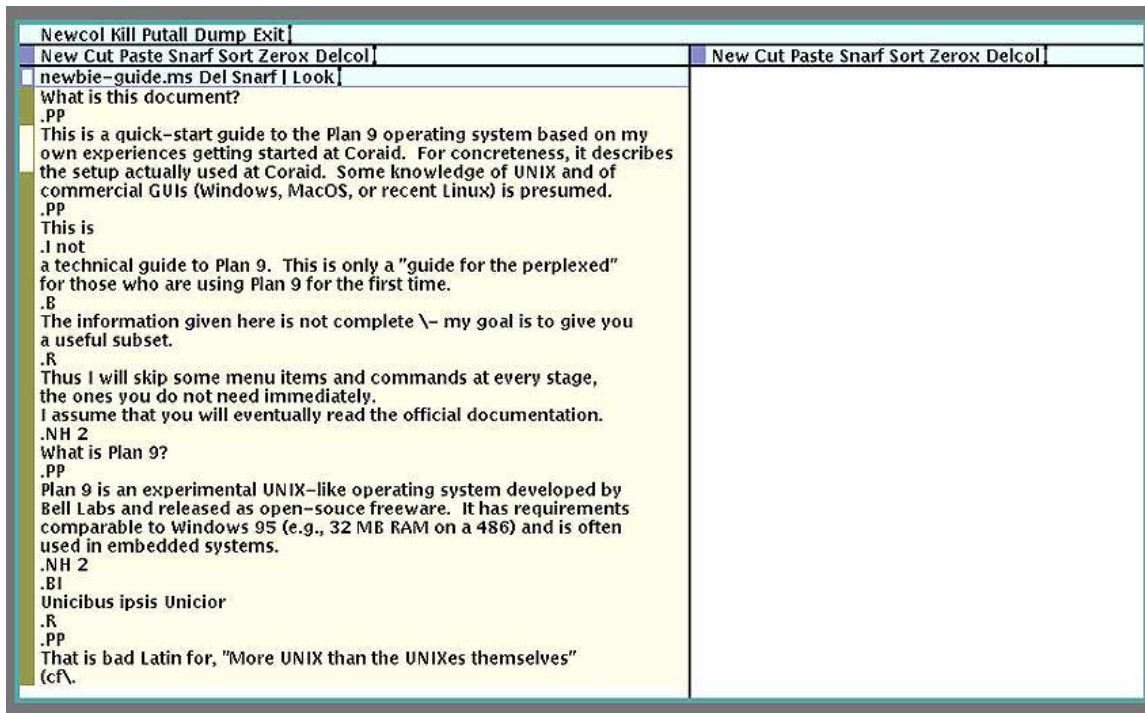


Figure 4. An *acme* editing window.

3.1. Typing in *acme*

Acme is more like a plain text window than you would guess. Text selection and scrolling work just the same as in *rio*. There are no drop-down menus; the commands at the top are executed by middle-clicking them.

The biggest difference between *acme* and *rio* is that in *acme*, you do not click on a panel in order to type into it; just put the mouse cursor on it, and that's where

your typing will go.

3.2. Quick start – how to create or edit a file

One way to start editing an existing file is to type a command such as:

```
acme filename.txt
```

Acme will open.

Make modifications, then middle-click on Put, then middle-click on Exit.

More commonly, you type *acme* without arguments, then right-click on folders or file names to choose what to edit.

To create a file, type *acme* without any arguments. Middle-click on New. The cursor will go to the beginning of the third menu line; type the filename there, then middle-click on Put.

3.3. Columns and panels (*acme* windows)

Acme is normally divided into two columns of panels ("windows"). As a beginner, you may want to get rid of the second (right-hand) column. To do so, middle-click the word *Delcol* above it.

New panels may not pop up where you want them. You can move them around by left-dragging the square that is at the upper left corner of each panel, just to the left of its name.

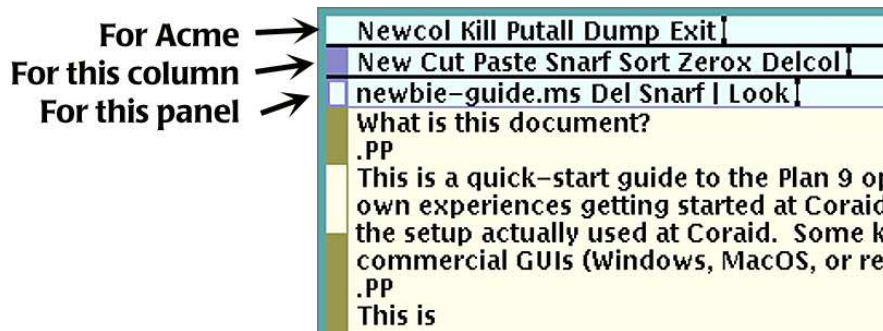


Figure 5. Menu (tags) at the top left of *acme*.

The menu ("tags") at the top of the left column are in three rows: one for all of *Acme*, one for the left column, and one for the current panel (Fig. 5).

And here's what all the menu items do:

Main menu

- Newcol** Create another column of panels.
- Kill** Interrupt currently executing command(s).
- Putall** Save all open files to disk.
- Dump** Write the state of Acme to a file so it can be resumed.
- Exit** Leave Acme.

Column menu

- New** Create another panel (window) in this column.
- Cut** Delete the selected text from the screen and place it in the snarf buffer.
- Paste** Copy the snarf buffer into the text at the insertion point.
- Snarf** Copy the selected text into the snarf buffer (from anywhere in the column).
- Sort** Arrange the panels in order by their names.
- Zerox** Create a copy of the current panel (thus giving you two windows into the same file).
- Delcol** Delete this whole column of panels.

Panel menu

- filename** The name of the file being edited. You can type over it to specify that the next Put will use a different filename.
- Del** Delete the whole panel (not the selection!). If the contents are "dirty" (need saving), you will get a warning, but a second Del will succeed.
- Snarf** Copy the selected text into the snarf buffer (from within this panel only).
- Undo** Undo the most recent editing change.
- Put** Save the file to disk. (Not shown if the file doesn't need saving.)
- Look** Search (requires an argument; see below). Other commands can go here.

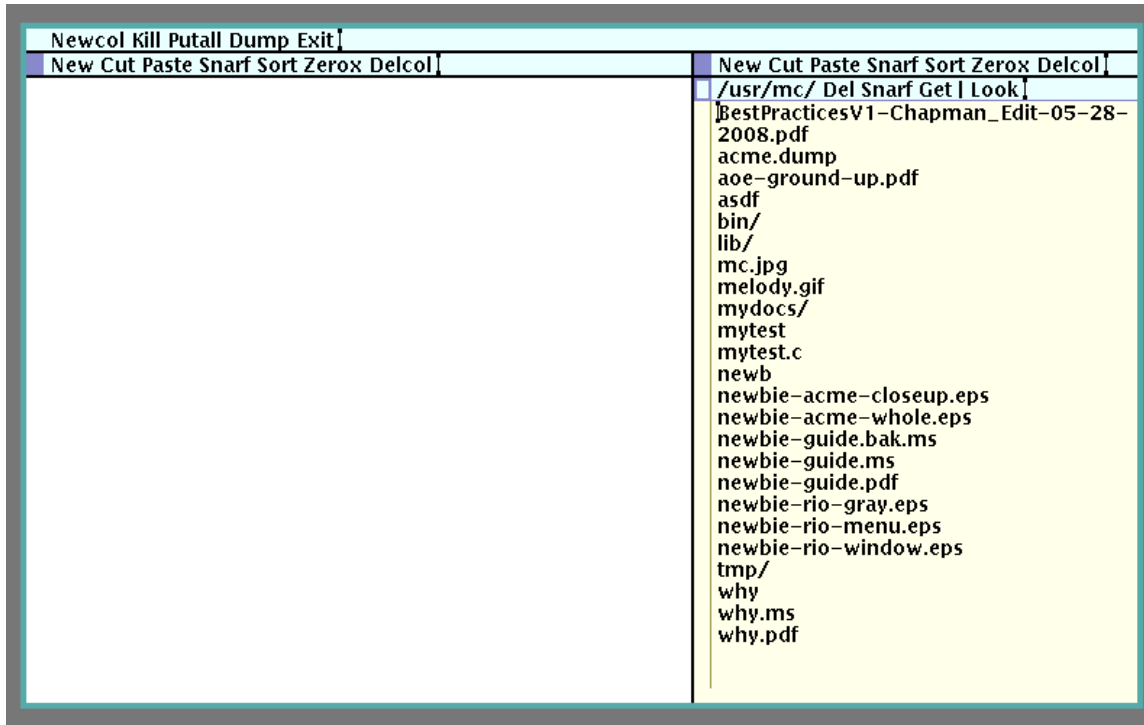


Figure 6. Browsing files in *acme*.

3.4. Browsing files and directories

If you start *acme* with no command line argument, the second column will become a browse window for selecting files and directories, as shown in Fig. 6. Right-click on any file to open it in the editor, or on any directory to explore it.

3.5. Editing commands

In the menu at the top of each panel, after the character | (the bar), you can type a command. The command "Look" is usually provided there by default. There are two ways to use it:

- You can type an argument after it, and then drag across "Look" and the argument with the middle mouse button;
- You can execute "Look" without an argument, in which case it will use the selected text in the editing panel as its argument.

Either way, "Look" means "find the next occurrence of this text and select it."

You can type other commands after the bar. In particular, if you type

```
Edit s/abc/def/g
```

you change all occurrences of `abc` to `def` throughout the selection (*not* throughout the file). To do a search-and-replace throughout the file, type

```
Edit ,s/abc/def/g
```

where the comma denotes an unlimited range of lines.

Any command from the editor *sam* can be typed in *acme* this way.

3.6. Mouse usage

A quicker way to find the next occurrence of a word is to right-click the word itself. To do this to a phrase that contains a space, you can either right-sweep the phrase (drag across it while holding down the right button), or select it (by left-dragging) and then right-click on it.

The use of the left and middle buttons has already been described. (The left button selects text in all of the ways that work in *rio*.) Much more information about *acme*, including the use of mouse-button "chords" (combinations), is described in *man acme*.

4. Communicating with personal computers

To access a Plan 9 system from a Linux, Windows, or MacOS system, you will run a terminal emulator called *drawterm*, available from <http://swtch.com>. Normally, you will write a script to start *drawterm* with the right CPU server and authentication server as arguments, such as:

```
drawterm -c minooka.coraid.com -a tyty.coraid.com
```

Under Windows, add `start /w` at the beginning of the command to give *drawterm* its own window.

When *drawterm* launches, it will complain that it "cannot chdir to" a directory on your PC. That is not a problem.

Drawterm acts like the console of a Plan 9 system; you can create windows in the usual way. In addition, *drawterm* mounts the root directory of the (first) hard disk of your PC into `/mnt/term` in addition to whatever is already mounted there. That enables you to copy files back and forth between Plan 9 and the PC transparently.

The Windows version of *drawterm* has a known problem – you can read and write the PC's hard disk but not list its directories. Thus

```
lc /mnt/term/Windows
```

will not show anything, but you can still read and write particular files on the PC

by name. To sort this out, try a Plan 9 command such as

```
date >/mnt/term/myfile.txt
```

and see where *myfile.txt* ends up.

5. Reading documentation

The *man* command works as in UNIX, but you do not have to specify paging or pipe to *more* because the Plan 9 screen does not scroll.

To view man pages elegantly converted to PostScript, use the *-P* option, as in:

```
man -P lc
```

To view PostScript, PDF, JPEG, and numerous other graphical formats, use *page*, as in:

```
page myfile.pdf
```

Page has a middle-button menu for advancing to the next page, etc.

6. Creating documentation with *troff*

The classic UNIX *troff* program is available in Plan 9 and understands UTF-8 Unicode. Here I give only a quick conceptual orientation to *troff*, with references to fuller documentation.

6.1. Running *troff*

Troff input files normally have names ending in ".ms". A typical command to run *troff* (at Coraid) is:

```
troff -Tutf -ms -mpictures myfile.ms | lp -dstdout | ps2pdf > myfile.pdf
```

That is: In UTF-8 compatible mode, load the "ms" and "mpictures" macro packages and process *myfile.ms*, then pipe it to *lp* (which converts it to PostScript), then pipe it to the PDF converter.

6.2. Kinds of *troff* markup

The input to *troff* is a text file annotated with several kinds of commands, among them:

- (1) Lowercase two-letter commands native to *troff*, such as *.bp* at the beginning of a line
- (2) Uppercase two-letter commands supplied by a macro package, such as *.PP* at the beginning of a line.

(3) Special inline codes starting with `\` and occurring anywhere in the text.

The second type is predominant; you will mostly use codes defined in the "ms" macro package, summarized in *man ms* and documented in tutorial form in:

Lesk, M. E. (1978), "Typing documents on the UNIX system: using the -ms macros with troff and nroff," reprinted in various places.

The source file for this document is a useful set of *troff* examples.

6.3. Some important but un-obvious codes

`\` marks a comment from its position to the end of the line.

However, blank lines are significant. To completely comment out a line, begin it with: `.\`

Here are some commonly used inline codes:

`\c` (at end of line): ignore line break.

`\-` dash or minus sign (longer than hyphen).

`\` required space (typed as `\` followed by a space); joins words to be treated as a single word.

`\(ua` up-arrow (many other special character codes resemble this one).

`\f1` switch immediately to roman type.

`\f2` switch immediately to italic type.

`\f2` switch immediately to bold type.

`\f3` switch immediately to bold italic type.

The normal way to switch to italics, boldface, etc., is to use macros such as `.I`, `.B`, `.R` (italics, bold, roman). These are used in three ways. First, you can start a block of text with `.I` and end it with `.R`, like this:

```
.I
This text will be
set in italics.
.R
Now we are back to roman.
```

Second, you can write the macro with one word right after it (or multiple words joined with backslash-space):

```
We will have just one
.I word
in italics.
```

Third, you can add a second argument, which is something (typically a punctuation mark) that should immediately follow the italicized word but be set in roman type, with no intervening space:

```
Did you say
.I bonjour ?
I thought so.
```

To learn more about *troff*, use the documentation and tutorials that are abundant on the World Wide Web.

7. Programming in C

7.1. How to run the compiler

Plan 9 supports a slightly extended version of Standard C. Here's a sample program, which we'll call *myfirst.c*:

```
#include <u.h>
#include <libc.h>

void main()
{
    print("Greetings, earthlings!\n");
}
```

The names of the compiler and linker depend on the CPU architecture. For the 386, they are *8c* and *8l*. (That's a lowercase L, not a digit 1.)

To compile, link, and run *myfirst.c*, do this:

```
minooka# 8c myfirst.c
minooka# 8l -o myfirst myfirst.8
minooka# myfirst
Greetings, earthlings!
```

Note that the first step creates *myfirst.8*; the second step would create *8.out* if it were not told to create *myfirst* instead; and in the third step, you do not have to write *./myfirst* because in Plan 9, your current directory is already on the path.

7.2. About *u.h*

In the sample program, the line

```
#include <u.h>
```

defines architecture-dependent features such as the size of *uchar*, and should always be included before any other *includes*. To learn more about this and other headers, see *man 2 intro*.

7.3. For more information

To learn more about the C compiler, and especially its Unicode-compatible i/o system, see */sys/doc/comp.ps* and */sys/doc/compiler.ps* on your Plan 9 system.

7.4. The low road: *pcc*

If all you want to do is compile a UNIX program to run under Plan 9, and you aren't interested in maximum performance, you can use the POSIX-compatible C compiler, *pcc*. In this situation you will use UNIX headers such as *stdio.h* and functions such as *printf*, leave out any mention of *u.h*, and compile and run your program this way:

```
minooka# pcc -o mysecond mysecond.c
minooka# mysecond
Hello, world!
```

This uses APE, the POSIX-compliant subsystem of Plan 9.