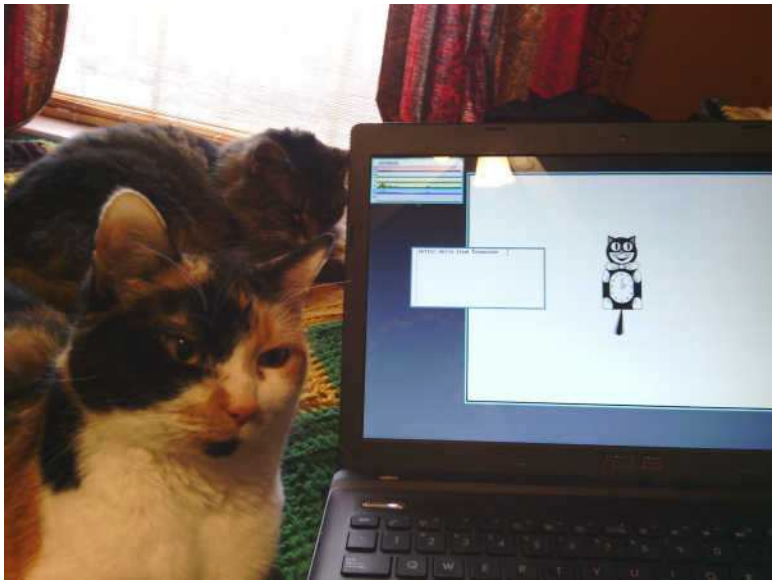
FQA 8 – Using 9front



i am form china
here is lots of people knows
and documented is chinese
the time comllexity is need you test youself
and you will find it
— wenwei peng

When applied consistently, simple conventions can combine to provide powerful results. In Plan 9, *conventions* are preferred to *rules*. This section explores the Plan 9 approach to actually using the computer.

8.1 – rc

The `rc` shell was written by Tom Duff for Research UNIX v10. It was later adopted as the shell for Plan 9. Some of its conventions are unusual compared with other command interpreters influenced by the Bourne shell. Although its syntax may seem strange at first, have patience; `rc` was designed this way on purpose. Once its (few, but powerful) features are internalized, `rc` simply gets out of the way.

Read: *Rc – The Plan 9 Shell,* `rc(1)`

8.1.1 – Prompts

Creating an `rc` function with the same name as your prompt allows you to easily double-click to select at the end of a previously typed line and then `send` it using the mouse button 2 menu (see the discussion of `rio` menus, below). This can be used to approximate a form of command history (see also the commands " and "", which print and execute the previous command, respectively).

Add something like this to your `$home/profile`:

```
fn term%{ $* }
```

In `rc` the `;` character forces the end of a line and is treated as a noop when it appears alone, so it is also possible to create a simple prompt that would require no special prompt function in order for the prompt to be effectively ignored when selecting and sending:

```
prompt=´; ´
```

Obviously, the prompt can be named however the user sees fit.

8.1.2 – /env

Note: Contents of the `/env` directory are provided by the kernel and represent a separate accounting of the shell's environment; `rc` reads `/env` only on startup, and flushes/writes `/env` only before executing programs.

8.2 – rio

`rio` is the Plan 9 window system. More accurately, `rio` multiplexes input devices with and serves a file interface to a series of rectangles, inside the boundaries of which are drawn an arbitrary arrangement of pixels. Controlling the rectangles is more straightforward, and at the same time more flexible, than what is commonly expected from most "window managers."

Read: `rio(1)`, `rio(4)`

To effectively use `rio`, you need a three button mouse. If you only have a two button mouse you can emulate the middle button by holding down the `shift` key whilst pressing the right button.

Note: Button 1, 2, and 3 are used to refer to the left, middle, and right buttons respectively.

8.2.1 - The Pop-up Menu

Pressing and holding down mouse button 3 on the gray desktop or on a shell window will give you a menu with the following options:

```
New

Resize

Move

Delete

Hide
```

Pressing and holding down mouse button 2 on a shell window results in a menu with the following options:

```
cut

paste

snarf

plumb

look

send

scroll
```

Select an item by releasing the button over the menu item. Rio uses the same button that started an action throughout that operation. If you press another button during the action the operation is aborted and any intermediate changes are reversed.

Each menu acts as a action verb selector which then requires an object (i.e. window) to be picked to indicate which window the verb is to act on. A further mouse action may then be required.

8.2.2 - Window control

Clicking on a window brings it to the front.

You can directly change the shape of a window by clicking and dragging on the edge or corner of the window border. Mouse button 1 or 2 will allow you to drag the edge or corner to a new size, and mouse button 3 will allow you to move the window.

The mouse button 3 menu contains a list of all windows that are corrently obstructed by

other windows. Selecting a label tops the window.

The pop-up menu remembers the last command chosen, so as a shortcut you can just press and release button 3 without moving the mouse between pressing and releasing to select the previous command again.

In addition, `rio` serves a variety of files for reading, writing, and controlling windows. Some of them are virtual versions of system files for dealing with the display, keyboard, and mouse; others control operations of the window system itself. These files, as well as the `window(1)` command, allow for controlling windows programmatically by reading and writing text strings. Thus simplifying the automated opening and placement of various windows with user scripts.

Read: `rio(4)`

8.2.3 – Text in rio windows

Text in a `rio` window may be freely manipulated, edited, altered, deleted and/or acted upon using either mouse chords or the options from the mouse button menus. (For an example, see the discussion of the use of `rc` prompts, above.)

The special file `/dev/text` (for the current window), or `/dev/wsys/`*n*`/text` (for window *n*) contains all text that has already appeared in the window. The contents of this file may serve as a primitive form of command history (and may be acted upon using standard command line tools), but are lost when the window is closed.

Seriously, read: `rio(4)`

8.2.4 – Scrolling

By default, a `rio` window will fill up with text and then block, obviating the need for a separate pager program (though the `p(1)` pager program still ships with the system).

Endless scrolling may be enabled by selecting `scroll` from the mouse button 2 menu.

The `up` or `down` arrow keys and `pgup` or `pgdwn` keys may be used to scroll up or down in consistently measured increments.

Holding down the `shift` key and pressing the up or down arrow key will scroll a single line in the respective direction.

9front's `rio` supports mousewheel scrolling. The heuristic employed is roughly the same as that of clicking in the scrollbar on the left of the window: when the mouse pointer is near the top of the window the scrolling increment is small, while as the mouse pointer approaches the bottom of the window the scrolling increment grows progressively larger. Presently this behavior is limited to `rio`, `sam`, and `mothra` but may later be extended to other programs.

Note: While the behavior of the arrow and page keys is fairly consistent between programs, mousewheel scrolling is not. So far, `shift` + `up` or `down` is only supported in `rio` windows.

8.2.5 – Mouse Chording

Almost anywhere — `sam(1)`, `acme(1)`, `window(1)` — you can use the following mouse chords:

`mb1` — Select text.

`mb1 double click` — Select word under cursor, or at the end/start of a line, select the whole line.

After selecting with `mb1` and while still holding `mb1` down (these chords also work with text selected by double-clicking, the double-click expansion happens when the second click starts, not when it ends):

`mb2` — Cut text.

`mb3` — Paste text (can be reverted by clicking `mb2` immediately afterwards).

To snarf (copy), click `mb2` immediately followed by `mb3`.

8.2.6 – Keyboard Shortcuts

Almost anywhere — `sam(1)`, `acme(1)`, `window(1)` — you can use the following shortcuts:

`Ctrl-u` — Delete from cursor to start of line.

`Ctrl-w` — Delete word before the cursor.

`Ctrl-h` — Delete character before the cursor.

`Ctrl-a` — Move cursor to start of the line.

`Ctrl-e` — Move cursor to end of the line.

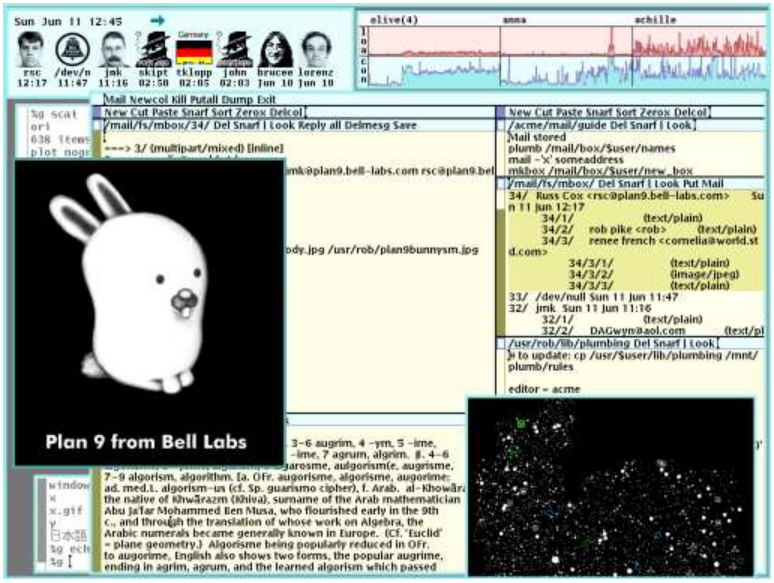
`Ctrl-b` — Move cursor to the position immediately after the prompt. (`rio` only)

Read: UNIX Keyboard Bindings

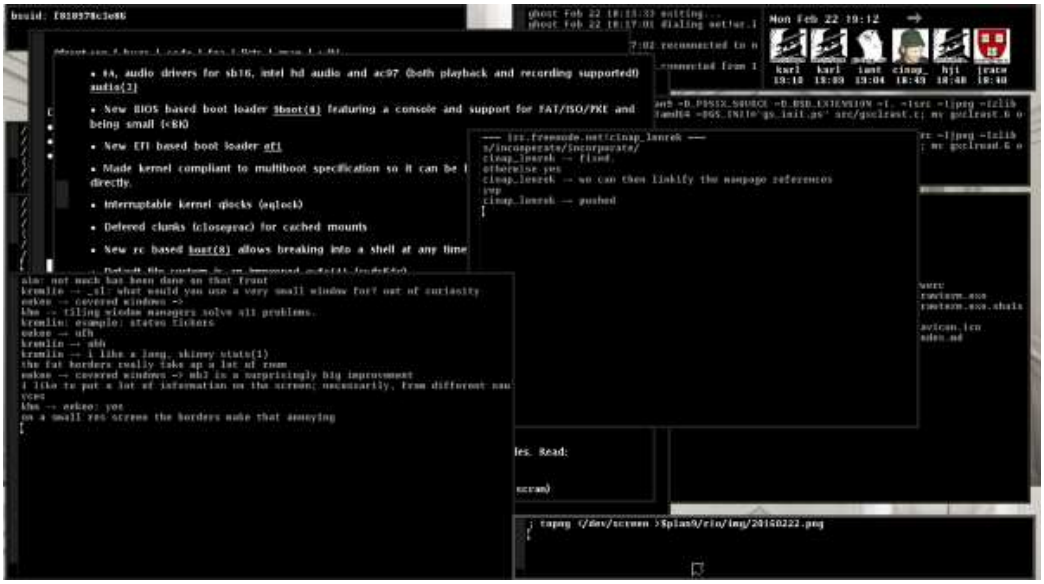
In a `rio(1)` window, scroll up or down one line by holding `shift` and pressing the up or down arrow.

8.2.7 – Color scheme

`rio` looks like this:



`rio`'s color scheme may be modified by editing the .c configuration files and re-compiling:



Note: Someone will mock you for doing this.

See: http://plan9.stanleylieber.com/rio

Rob Pike, rio's author, was all like:

> the clean appearance of the screen comes mostly from laziness, but the color scheme is (obviously) deliberate. the intent was to build on an observation by edward tufte that the human system likes nature and nature is full of pale

> colors, so something you're going to look at all day might best serve if it were also in relaxing shades. renee french helped me with the specifics of the color scheme (she's a professional illustrator and my color vision is suspect), once i'd figured out how i wanted it to look. there are still some features of the color system that i put in that i think no one has ever noticed. that's a good thing, in my opinion; the colors should fade away, if you'll pardon the expression. having used other systems with different approaches to color screens, most especially windows XP (extra pukey), i think tufte was right.

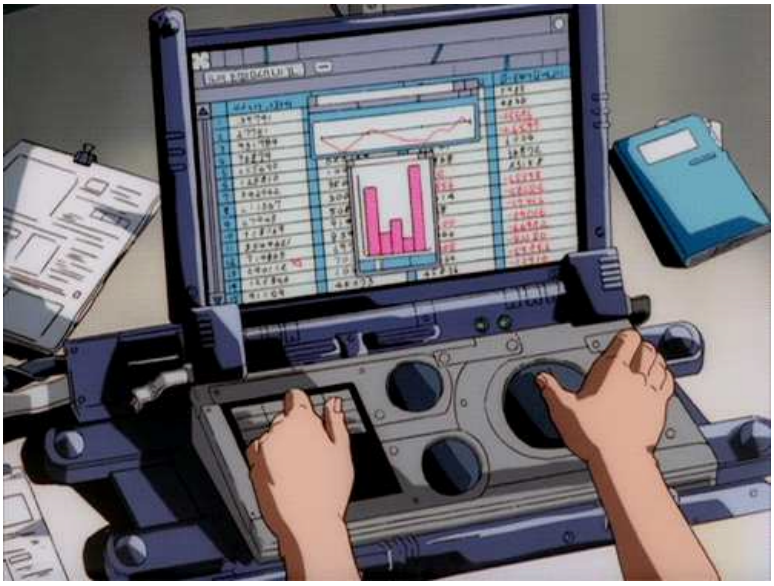
Rob Pike, 2003

> The color scheme was an attempt to honor a point made originally in a little brochure by Edward Tufte that the colors of nature are soft and quiet and peaceful to look at, while most computer screens are covered in glaring bright colors. When color came to the system I wanted it to be pleasant.

Rob Pike, 2008

See: edwardtufte.com

8.2.8 – Why is rio like this?



Window systems should be transparent. That's the argument put forward in the famous paper by rio's author, Rob Pike.

Beyond this, Rob offered an explanantion (in response to a question on the 9fans mailing list) of some of the choices made in the design of 8½ and `rio`:

> functioning cursor keys would still be a speed benefit.

This feels true but is false. There were some fascinating experiments done a few years ago in which people were given a long, tedious editing task. Some of the people were keyboard fans, some were mouse fans. Both folks were asked to do the task two ways, in random order, once using the mouse to do the editing, once using cursor keys etc. Regardless of their predilections, which was stated up front, after the experiment everyone who did the task agreed that it was faster to use the keyboard than the mouse to complete the task. Everyone. Here's the kicker: everyone was wrong. They were being timed, and in fact the reverse was true. Although they thought the keyboard was faster, doing the task using the mouse was faster for everyone, by a substantial fraction.

The explanation, besides the obvious that arrow keys are actually pretty slow if you're going more than a line or character, is that people feel the mouse wastes time because you need to grab it and move it, but it's time well spent. The part of the brain that uses keyboard commands to move the cursor is a higher-order function, and thinking and planning how to use the keys to get to the destination blocks thinking about the editing task at hand. But using the mouse is done by a lower-order part of the brain, which keeps the editing part of the brain clear. There's less task switching going on when you use the mouse, so you work more efficiently.

If you don't believe me, the story is here:

http://www.asktog.com/readerMail/1999-12ReaderMail.html

Thanks to some forgotten 9fan who mentioned this a while back. I didn't know about these experiments when I said, long ago, that using arrow keys to

> point at a display is like telling someone how to go somewhere by giving directions, while using a mouse is like pointing at a map. In fact, I never used a screen editor until I had a mouse, for just this reason.

Rob Pike, 2001

8.2.9 – tips

8.2.9.1 – Taking a screenshot

To capture the entire screen:

```
topng </dev/screen >screen.png
```

To capture only the current window:

```
topng </dev/window >window.png
```

It is also possible to capture *other* windows:

```
topng </dev/wsys/n/window >window.png
```

where *n* is the number of the window being captured.

Read: `rio(4)`

8.2.9.2 – Prevent console messages from overwriting the screen

To capture console messages in a `rio` window, open a new window and:

```
cat /dev/kprint
```

8.3 Text Editors

8.3.1 – sam

The text editor `sam` was created by Rob Pike for Research UNIX V9 (circa 1986), and later included with Plan 9.

See: http://sam.cat-v.org

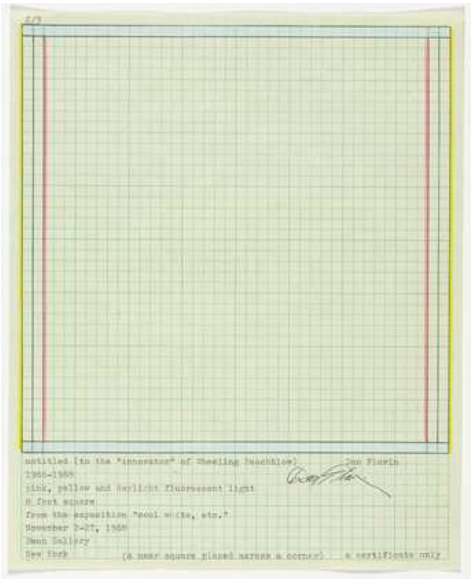
Read:

The Text Editor sam — The original paper by Rob Pike.

A Tutorial for the Sam Command Language — Documents the editing language.

sam quick reference card

`sam(1)` man page



Dan Flavin, *Document for Untitled (to the "innovator" of Wheeling Peachblow),* 1968

8.3.1.1 – Scrolling

9front's slightly modified version of `sam` supports mousewheel scrolling in the same manner as `rio`.

Read: *FQA 8.2.4 – Scrolling*

8.3.1.2 – Mouse Chording

9front `sam` supports the same mouse chording as `rio`.

Read: *FQA 8.2.5 – Mouse Chording*

8.3.1.3 – Why does sam have a separate snarf buffer from rio?

The program's author, Rob Pike, says:

> was a consequence of running over 1200 baud when sam was first written. you didn't want every cut and paste to bounce off the remote end at that speed. nowadays that argument has less weight. on the other hand, i still kinda like that you can have an editing session that doesn't corrupt what you have in rio's snarf buffer. i tried the unified way in acme and i often (not always) miss the old way.

Rob Pike, 2003

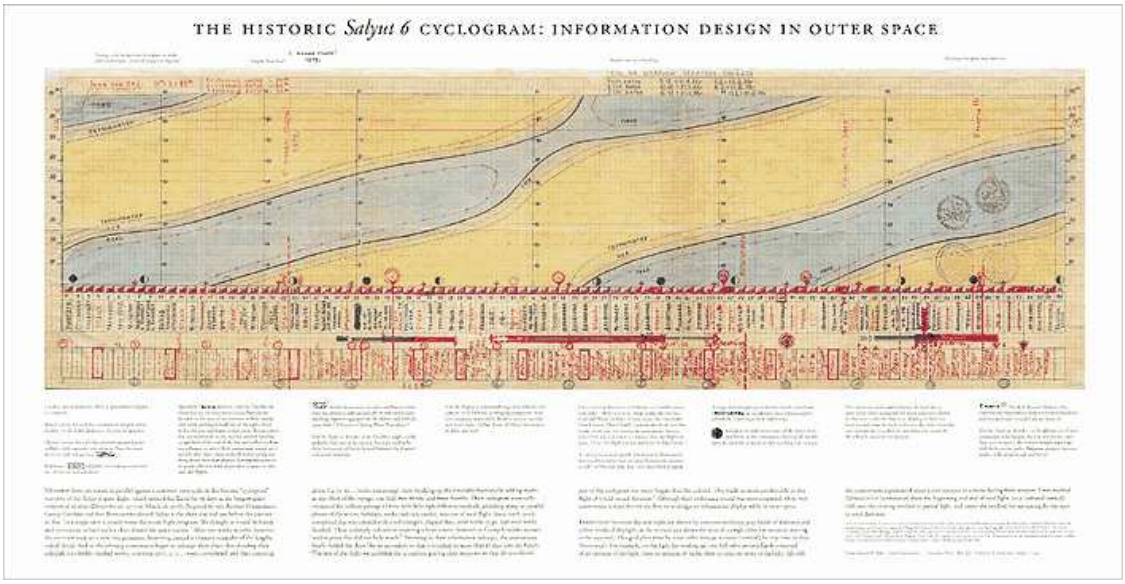
8.3.1.4 – Keyboard Shortcuts

`Esc` — Cut (and consequently, snarf) the selected text.

`Ctrl-b` — Switch focus to the edit window.

8.3.2 – acme

> *There is also an alternative user interface, acme(4), that some people use as their editor.*
> — Geoff Collyer



Handmade cyclogram by Russian cosmonaut, Georgi Grechko.

8.4 – Internet

Sending and receiving bits via alien protocols.

8.4.1 – Mail

Read: `mail(1)`, *FQA 7.7 – Mail server configuration and maintenance*

8.4.1.1 – upasfs

From `upasfs(4)`:

> Fs is a user level file system that caches mailboxes and presents them as a file system. A user normally starts fs in his/her profile after starting plumber(4) and before starting a window system, such as rio(1) or acme(1). The file system is used by nedmail(1), acme(1)'s mail reader, and imap4d and pop3 (both pop3(8)) to parse messages. Fs also generates plumbing messages used by biff and faces(1) to provide mail announcements.

Read: `upasfs(4)`, `pop3(8)`, `faces(1)`

8.4.1.1.1 – Reading gmail via IMAP

```
upas/fs -f /imaps/imap.gmail.com/your.username@gmail.com
```

The first time this command is run, you should see an error that looks something like this:

```
upas/fs imap: server certificate 22471E10D5C1E41768048EF5567B27F532F33
        not recognized
upas/fs: opening mailbox: bad server certificate
```

To add this certificate to your system, type:

```
echo ´x509 sha1=22471E10D5C1E41768048EF5567B27F532F33´ \
        >>/sys/lib/tls/mail
```

Once `upas/fs` is running, you can open as many additional gmail mailboxes (labels) as you wish:

```
echo open /imaps/imap.gmail.com/your.username@gmail.com/yourlabel \
        yourlabel >/mail/fs/ctl
```

Note: Opening large mailboxes over a slow 9p link will be very slow.

8.4.1.2 – nedmail

`nedmail` is a command line mail client similar to the classic mail client shipped with Research UNIX.

Read: `nedmail(1)`

8.4.1.3 – nupas

Read: *Scaling Upas,* by Erik Quanstrom **Note:** Erik's nupas has been merged with 9front's upas.

8.4.2 – NNTP

Read: `newt(1)`, `nntpfs(4)`

8.4.3 – IRC

8.4.3.1 – ircrc

`ircrc` is an IRC client implemented in `rc`. It is included with 9front.

Read: `ircrc(1)`

8.4.3.2 – irc7

A persistent IRC client was written in the c programming language by Andrey Mirtchovski. It has been modified slightly by 9front users (mainly, adding an `-e` flag to the `ircsrv` program that implements SSL connections).

8.4.3.3 – ircs

A persistent IRC client was written in the c programming language by jpm.

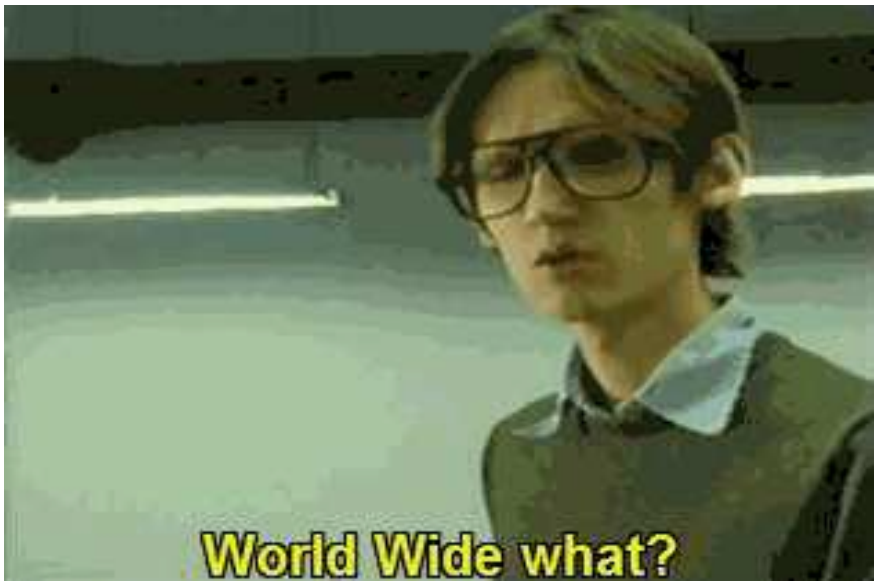
8.4.3.4 – wircrc

A windowed version of `ircrc` was implemented in rc by cinap_lenrek. Several unsanctioned versions with various additions have since occasionally been spotted.

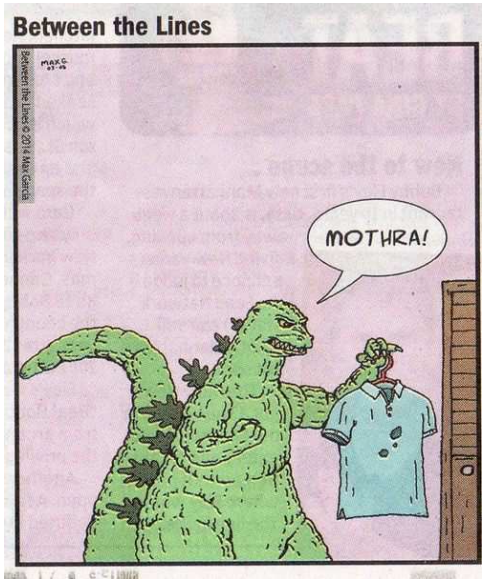
8.4.4 – FTP

Read: `ftpfs(4)`

8.4.5 – HTTP



8.4.5.1 – mothra



`mothra` is a trivial web browser written in 1995 by Tom Duff. It ignores Javascript, CSS and many HTML tags. It was dropped from Plan 9 after the 2nd Edition, but has been picked up and (somewhat) refined for 9front. `mothra` now uses `webfs`, and no longer supports non-HTTP protocols.

Read: `mothra(1)`, `webfs(4)`

8.4.5.2 – abaco

no.

8.4.5.3 – hget

`hget` is a command line HTTP client (similar to programs such as `curl` or `wget`) that started out as a c program in Plan 9 from Bell Labs, but was re-implemented in `rc` for 9front. `hget` now uses `webfs` and no longer supports non-HTTP protocols.

Read: `hget(1)`, `webfs(4)`

8.4.5.4 – charon

The Inferno operating system can be run hosted on Plan 9, and includes a GUI web browser called charon, which implements ECMASCRIPT 1.0 as well as additional HTML attributes.

Note: `charon` is ancient and is not really a sufficient replacement for 9front's web browsers. The rudimentary javascript support can be useful for some simple tasks.

8.4.5.5 – i

There exists an unfinished/buggy port of `charon` from Inferno's limbo programming language to Plan 9 c.

8.4.6 – SSH

Several SSH clients exist for Plan 9, none of which are perfect.

8.4.6.1 – ssh

9front used to ship with the original Plan 9 native SSH1 client from Bell Labs. It has since been deleted.

A new SSH2 client has been written from scratch to replace it. The new client supports only chacha20-poly1305 cipher and curve25519 Diffie-Hellman for key exchange (as well as plain passwords).

Read: `ssh(1)`

8.4.6.1.1 – sshfs

9front ships with an sshfs client that implements the SFTP protocol over the existing `ssh(1)` client.

Read: `sshfs(1)`

8.4.6.2 – ssh2

Programmers at Coraid created a Plan 9 native SSH2 client that was picked up (and completely rewritten) by Bell Labs. It is currently not included with 9front.

Note: There are bugs and expected features are missing. Consult the source.

8.4.6.3 – scpu

Two 9front users (taruti and mischief) worked on an SSH2 client written in the Go programming language. It has been extended to work with Plan 9 factotum(4), but still does not fully honor complex Plan 9 dial(2) strings.

8.4.6.3.1 – Public Key Authentication

The `scpu` command can be configured to use public key authentication:

```
auth/rsagen -t 'service=ssh' >$home/lib/ssh/key
auth/rsa2ssh -2 $home/lib/ssh/key >$home/lib/ssh/key.pub
cat $home/lib/ssh/key >/mnt/factotum/ctl        # must be present before running
```

Then add the contents of `$home/lib/ssh/key.pub` to `$HOME/.ssh/authorized_keys` on the remote host.

Note: This same key may be used for multiple hosts.

8.4.6.4 – OpenSSH

Plan 9 user fgb ported OpenSSH 4.7p1, OpenSSL 0.9.8g 19 Oct 2007 to Plan 9. It is available in his contrib directory (on the Bell Labs server), or a 386 binary is available here (to install, unpack it over /): openssh.tgz.

8.4.6.5 – sftpfs

An implementation of sftpfs was created for Plan 9 that can work with either the native SSH clients or fgb's OpenSSH port.

8.4.6.5.1 – Mounting a remote u9fs share over SSH

The `u9fs` program runs on UNIX and serves an unencrypted `9P(2)` share. It is possible to mount such a share over SSH.

With `ssh`:

```
srv -s 5 -e ´ssh -u sl -h wm ´´/usr/local/bin/u9fs \
        -u sl -na none´´´ wm /n/wm
```

With `ssh2`:

```
srv -s 5 -e ´ssh2 -l sl wm ´´/usr/local/bin/u9fs \
        -u sl -na none´´´ wm /n/wm
```

With `scpu`:

```
srv -s 5 -e ´scpu -u sl -h wm -c \
        ´´/usr/local/bin/u9fs -u sl -na none´´´ wm /n/wm
```

In all cases, an SSH connection is opened to remote UNIX host `wm`, logged in with user `sl` and mounted on Plan 9 under `/n/wm`.

Read: `u9fs(4)`, `srv(4)`

8.4.7 – secstore

Typing in lots of passwords over and over again is annoying.

Secstore authenticates to a secure-store server using a password and optionally a hardware token, then saves or retrieves a file. This is intended to be a credentials store (public/private keypairs, passwords, and other secrets) for a `factotum`.

Read: *FQA 7.4.3 – secstored* for information on setting up the secstore server, and: *FQA 7.4.3.1 – Adding users to secstore* to add users.

Once a user has been added to secstored, the user may add to the file read by factotum at startup. To do so, open a new window and type

```
% ramfs -p; cd /tmp
% auth/secstore -g factotum
secstore password: [user's secstore password]
% echo ´key proto=apop dom=x.com user=ehg !password=hi´ >> factotum
% auth/secstore -p factotum
secstore password: [user's secstore password]
% read -m factotum > /mnt/factotum/ctl
```

and delete the window. The first line creates an ephemeral memory-resident workspace, invisible to others and automatically removed when the window is deleted. The next three commands fetch the persistent copy of the secrets, append a new secret, and save the updated file back to secstore. The final command loads the new secret into the running factotum.

The ipso command packages this sequence into a convenient script to simplify editing of files stored on a secure store. It copies the named files into a local ramfs and invokes ditor on them. When the editor exits, ipso prompts the user to confirm copying modifed or newly created files back to secstore. If no file is mentioned, ipso grabs all the user's files from secstore for editing.

By default, ipso will edit the secstore files and, if one of them is named factotum, flush current keys from factotum and load the new ones from the file.

Read: secstore(1), secstore(8)

8.4.8 – drawterm

drawterm is a program that users of non-Plan 9 systems can use to establish graphical cpu connections with Plan 9 cpu servers. Just as a real Plan 9 terminal does, drawterm serves its local name space as well as some devices (the keyboard, mouse, and screen) to a remote cpu server, which mounts this name space on /mnt/term and starts a shell. Typically, either explicitly or via the profile, one uses the shell to start rio. The original version is effectively abandoned, but is available here: http://swtch.com/drawterm

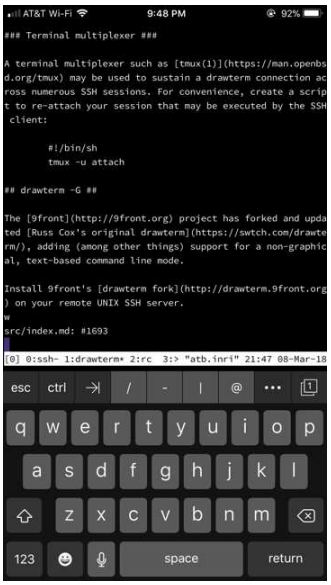
There also exists a fork of Russ Cox's drawterm that incorporates features from 9front, most importantly DP9IK authentication support (see `authsrv`(6)) and the TLS based `rcpu`(1) protocol: `http://drawterm.9front.org`.

Note: The fork is the preferred version of drawterm for use with 9front because the old auth protocol is considered deprecated and the old CPU listeners are now disabled by default.

8.4.8.1 – Connect to Plan 9 from a mobile device

Use an SSH client to connect to a remote UNIX SSH server that can run the 9front fork's `drawterm -G`:

`http://helpful.cat-v.org/Blog/2017/11/29/0/`



8.5 – Audio

Read: `audio(1)`, `audio(3)`

8.6 – External Media

8.6.1 – Mount an ISO9660 CD-ROM

```
mount <{9660srv -s} /n/iso /dev/sdD1/data # cd-rom drive
```

or:

```
mount <{9660srv -s} /n/iso /path/to/9front.iso
```

Read: `dossrv(4)`

8.6.2 – Burn a CD-ROM

```
cdfs
cp 9front.iso /mnt/cd/wd
rm /mnt/cd/wd
```

Read: `cdfs(4)`

8.6.3 – Mount a FAT formatted USB device

FAT formatted USB devices are automatically mounted under the `/shr` directory.

Note: Devices must be FAT or FAT32 formatted; exFAT is not supported.

8.7 – Emulation

8.7.1 – Linux Emulation

`linuxemu` is a program that can execute Linux/i386 ELF binaries on Plan 9. Semi-modern web browsers and other Linux programs may be run using `linuxemu` (if necessary, in conjunction with the `equis` X11 server).

Note: `linuxemu` can only be run on a Plan 9 system booted with a `386` kernel and binaries.

BOOTSTRAP

To run `linuxemu`, you need a Linux root file system packed into a tarball:

http://felloff.net/usr/cinap_lenrek/mroot-linuxemu.tbz

http://plan9.stanleylieber.com/linuxemu/mroot.tgz

The `mroot-linuxemu.tbz` version contains no symlinks and can be extracted with plain Plan 9 tools `bunzip` and `tar`.

The `mroot.tgz` version contains the same Debian Sarge base as `mroot-linuxemu.tbz`, but with several additional packages pre-installed:

- 9base
- dmenu-4.1.1
- dwm-5.8.2
- gcc 3.3.5

linux-kernel-headers

mercurial 0.9.4

opera 10.11

python 2.3.5

xlib-dev

and more.

You can create your own `mroot` with `debootstrap` on Debian Linux, or help write an installer that unpacks and installs an alternative distribution on Plan 9... In any case, `linuxemu` is not hardwired to any Linux distribution!

RUNNING

Use the provided `linux` script to chroot into your Linux `mroot`. The `linux` script is neccesary because for Linux programs to run, shared libraries from your `mroot` have to appear in its `/lib` and `/usr/lib` directories, while configuration files are expected to be in `/etc`. The script will build a private namespace and then bind the Linux `mroot` over the Plan 9 root. The original Plan 9 namespace is mounted within `linuxemu` under `/9`.

Assuming `mroot` is located in the current directory, start `linuxemu` like this:

```
linux -r ./mroot /bin/bash -i
```

If the `-r` option is omitted, the Linux `mroot` defaults to `/sys/lib/linux` on the Plan 9 machine.

In the Linux `mroot`, `/etc/resolv.conf` should be changed to match your network nameserver. In addition, `/etc/apt/sources.list` should be updated to a working Debian mirror. Sarge packages can still be accessed at:

`deb http://archive.debian.org/debian-archive/debian sarge main`

EXAMPLES

Linux X11 programs may be used in conjunction with the `equis` X11 server. For example, to run the Opera web browser under your Linux `mroot`, start `equis` in a `rio` window, start `linuxemu` in another `rio` window and then from within `linuxemu`:

```
dwm & # X11 window manager
opera & # web browser
```

Opera should (eventually) appear in the `equis` window. A window manager (this example uses dwm) is recommended so that X11 programs interact with window resources properly.

DEBUGGING

If `linuxemu` crashes, use acid to figure out whats going on:

```
mk acid
acid -l linuxemu.acid <pid>
```

Then you can issue the following commands:

```
ustk()
```

dump a (userspace) stacktrace for the current thread:

```
umem(Current())          dump the memory mappings
ufds(Current())          dump the filedescriptor table
utrace(Current())        dump the internal tracebuffer
                         (enabled by -d option)
```

Use `xasm()` and `xcasm()` for disassembly for Linux code.

Read: `acid(1)`

You can also enable full trace logging:

```
linux -r ./mroot -dd /bin/bash -i >[2]/tmp/linuxemu.log
```

This slows `linuxemu` down considerably. In case of race conditions, it often happens that the bug disapears when doing full trace logging!

8.7.2 - Nintendo

Emulators for several Nintendo video game consoles ship with the system:

gb — Game Boy

gba — Game Boy Advance

nes — Nintendo Entertainment System

snes — Super Nintendo Entertainment System

Read: `nintendo(1)`

8.7.3 – Sega

An emulator for the Sega Megadrive/Genesis video game console ships with the system:

md — Sega Mega Drive/Genesis

Read: `sega(1)`

8.7.4 – Commodore

An emulator for the Commodore 64 home computer ships with the system:

c64 — Commodore 64

Read: `commodore(1)`

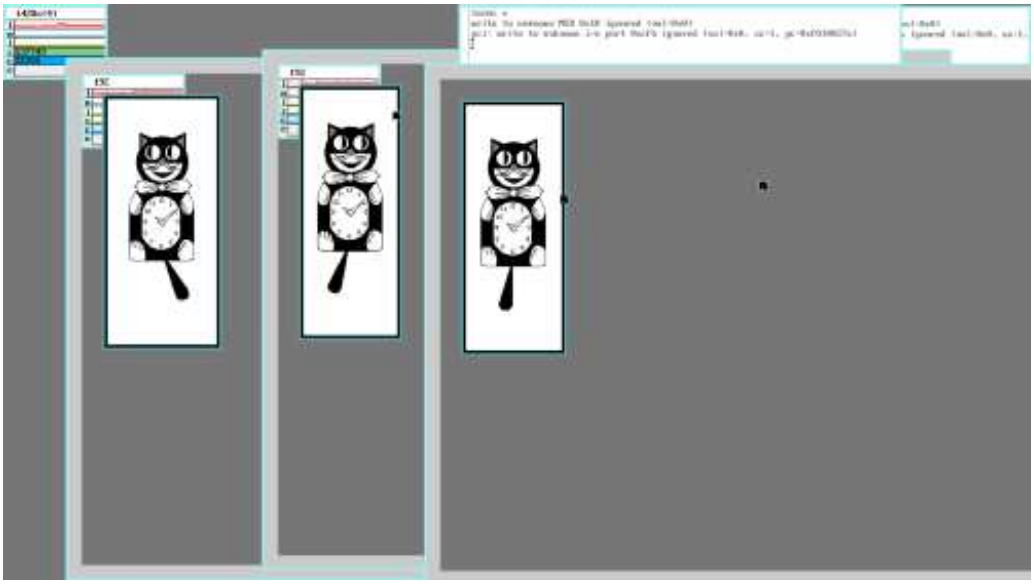
8.7.5 – PC

An emulator for PC compatible computers ships with the system:

vmx — virtual PC

Read: vmx(1), vmx(3)

8.7.5.1 – Virtualization Using vmx(1)



vmx(1) simulates a virtual PC running a specified kernel file, by using virtualization extensions found on recent intel processors. Currently, only 9front and recent Linux and OpenBSD kernels are supported.

The virtual PC is configured on vmx(1)'s command line, and the hardware specified is seen as virtio devices. It will use one of the host's CPU cores, and will run on the same architecture as the host.

Note: vmx executes the operating system's kernel directly, acting as a bootloader. It therefore needs explicit support for it unless the kernel is in multiboot format.

Note: vmx currently works on intel processors only, and requires a number of virtualization features. To check if your processor is supported, use icanhasvmx(8).

Basic examples:

- Boot 386 kernel with 1 GB of RAM, a 9front iso as a disk, a network interface through ether0 and a 640x480 framebuffer:

```
vmx -M 1G -d 9front.iso -n ether0 -v 640x480 /386/9pc
```

- Instead of a framebuffer, use con(1) to connect to the console:

```
window -scroll ´bind ´´#|´´ /n/p; \
        <>[3]/n/p/data1 {echo 3 >/srv/pipe; \
        con -r /n/p/data}´
vmx -c /srv/pipe -M 1G -d 9front.iso /386/9pc ´console=0´
```

8.7.5.1.1 Block Devices

It may be desirable to attach a disk to the virtual PC. One may then specify a number of files to be used as raw disk images with the `-d` flag. The files may be virtually anything so long as `vmx(1)` can overwrite them.

The common options here include plain files, `sd(3)` disks, or ISO images.

The fastest way to generate a big plain file is to create a sparse file. For example, to create a 4 GB sparse file with `dd(1)`:

```
dd </dev/zero -of dicks -bs 1 -count 1 \
        -seek '{echo 4*1024*1024*1024-1 | pc -n}
```

Using a real disk might yield somewhat faster performance. For example, using a USB:

```
vmx -d 9front.iso -d /dev/sdUxxxxx/data -v 640x480 /386/9pc
```

Use real disks with caution! `vmx` may induce kernel panics in the guest, for instance through bugs or quirks in the virtio devices' implementation. Beware that the host crashing may also trash your disks -- for instance, giving the guest too much memory, which is always allocated in full on start up, will trigger an OOM on the host.

8.7.5.1.2 Ethernet

If network connectivity is required, the `-n` parameter specifies an interface to bridge as a virtio ethernet card. `vmx(1)` will then send and receive traffic on this interface like the host. Wireless ethernet interfaces may also be used without any additional work. The interface can also be a dial string or a plain file. The emulated card's MAC address is random by default, and can be changed using an optional `ea:` prefix.

For example, to bridge an ethernet interface and use `DE:AD:BE:EF:CA:FE` for the virtio device's MAC:

```
vmx -d 9.img -n ea:deadbeefcafe!ether0 -v 640x480 /386/9pc
```

8.7.5.1.3 OpenBSD

OpenBSD kernels may change radically between releases. Only 6.1 and later have been tested. Keep in mind that the versions of the kernel passed to `vmx(1)` and the system provided on a disk must be in sync.

Besides the various kernel files and optional devices, little is needed to coerce OpenBSD to work.

To use the OpenBSD installer, first find a `bsd.rd` kernel. To then use an existing OpenBSD install, use a `bsd` kernel instead. A networked install may be used if an ethernet interface is specified on the command line: it will use OpenBSD's `vio(4)` driver. Otherwise, an `install??.fs` file may be used as a disk.

Note: OpenBSD/386 does not support plain framebuffer graphics. You would need to either use VESA, or configure a COM device and add a `tty=` option to the command line.

For example, to install OpenBSD 6.2 to a disk file using an install image and VESA graphics:

```
vmx -d obsd.img -d install62.fs -v vesa:640x480 bsd.rd
```

Boot options are given as the kernel's command line. The root device is specified with the `device=` option, and if unset, is queried by OpenBSD's bootloader.

To use VESA with X11, one must specify the `-v` argument with a vesa: prefix, one or more display modes, and set `machdep.allowaperture=2`.

Example usage:

```
vmx -M 1G -c /srv/pipe -n ether0 -d /dev/sdUa2595/data \
        -v vesa:640x480,800x600,1024x768 \
        bsd ´tty=com0´ ´device=sd0a´ ´db_console=on´
```

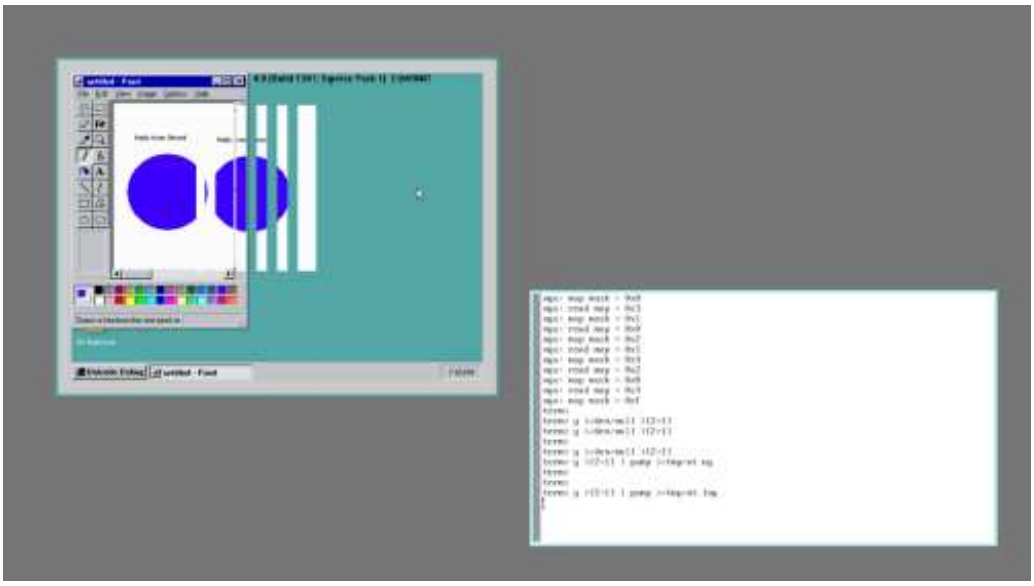
8.7.5.1.4 Linux

You will need both a kernel and an initrd which will be used as a module. You must also specify the root disk on the kernel's command line.

An example with Alpine Linux:

```
vmx -M 1G -n ether0 -d alpine-standard-3.6.2-x86_64.iso \
        -d alp.img -m initramfs-hardened -v vesa:800x600 \
        vmlinuz-hardened ´root=/dev/vda1´
```

8.7.5.1.5 Windows NT



Classified.

8.8 – Additional Software

8.8.1 – 9front sources server

Additional 9front software is available from a 9P share that is accessible from any Plan 9 system:

```
9fs 9front
```

The following files and directories will then be available under `/n/`:

`9front/` — 9front source

`9front.torrent` — torrent of current 9front ISO image

`extra/` — third party software source

`fqa/` — troff sources for 9front Frequently Questioned Answers

`hardware/` — known working hardware (sysinfo, firmware, manuals, etc.)

`iso/` — current 9front ISO image(s)

8.8.2 – 9front contrib

Some 9front users maintain a contrib directory on an official 9front 9P share (similar to the contrib arrangement provided by Bell Labs) that is accessible from any Plan 9 system:

```
9fs 9contrib
```

User directories will then be available under `/n/contrib/`, and a mostly completete mirror of the defunct Bell Labs sources server will be available under `/n/sources/`.

These directories are also accessible via HTTP: `http://contrib.9front.org`

8.8.3 – Other public 9p servers

A list of active public 9p servers is maintained here: `http://www.9paste.net/qrstuv/9pindex`

8.9 – Bootstrapping architectures not included on the ISO

8.9.1 – amd64

To setup the `amd64` port, install the `386` port from the ISO, then cross compile and install the `amd64` binaries and kernel.

Read: *FQA 5.2.2.1 – Cross compiling, FQA 7.2.5 – How do I install a new kernel?*

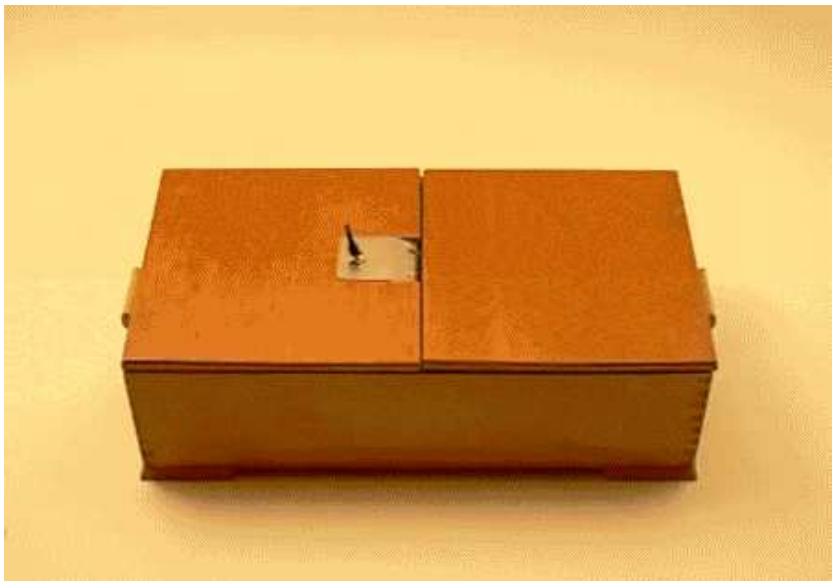
8.9.2 – Raspberry Pi

The most convenient way to use an rpi is to cross compile and install the `arm` binaries and the `bcm` kernel on the network file server, and then tcp boot the rpi.

Read: *FQA 5.2.2.1 – Cross compiling, FQA 6.7.1 – How do I tcp boot?*

Outdated and possibly confusing instructions for installing directly onto the rpi's sd card are detailed in *Appendix J – Junk*

8.10 – ACPI



Plan9front currently has partial ACPI support for PCI interrupt routing and system shutdown.

8.10.1 – Enabling ACPI

ACPI is enabled with the presence of `*acpi=` boot parameter.

This will create the `/dev/acpitbls` file that can be used to read the systems acpi tables. Specifying `*acpi=0` will make acpi tables accessible thru the file but not use it in the kernel.

8.12 – Revision Control

8.12.1 – cvs

OpenCVS was ported to Plan 9.

An implementation of a `cvs` file server, called `cvsfs`, was also created for Plan 9.

8.12.2 – git

More than one person has claimed to have ported `git` to Plan 9, but no code has ever been made public. Please stop telling unsuspecting newbs that `git` is available for Plan 9.

In the meantime, someone wrote a shell script wrapper that attempts to replicate some basic `git` actions by downloading a zip file from the repository and performing operations on it.

8.12.3 – Mercurial

9front ships with Mercurial.

Read: *FQA 5.2.1.1 – hgrc*

See also: `hgfs(4)`

8.12.4 – svn

No.