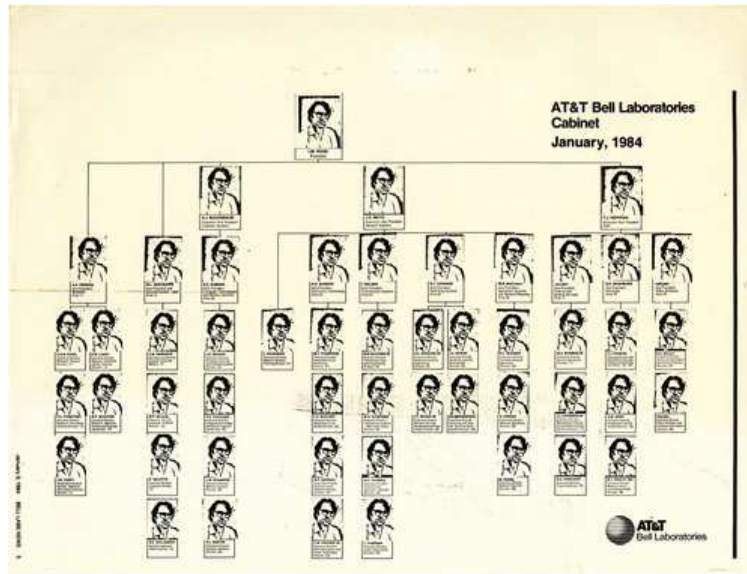
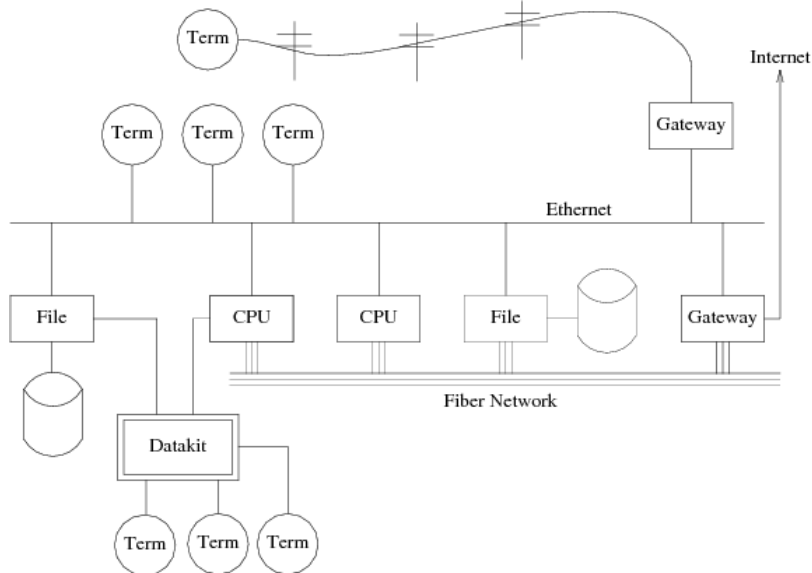


## FQA 7 – System Management



### 7.1 – Plan 9 Services Overview



In order to be an effective system manager it is a good idea to understand how the system is designed, and how it is intended to be used.

A Plan 9 installation consists of a disk file server, an authentication server, and one or more cpu servers and terminals—all sharing the same disk file system.

That said, Plan 9 services may be run on separate machines, all together on one

machine, or in various combinations. The original design of Plan 9 assumed that each network service would run on separate hardware; by design, individual components of the system are generally unaware if they co-exist on the same machine or are distributed amongst separate machines.

This document will describe individual services as if they are all running separately.

Read: *Designing Plan 9, Plan 9 From Bell Labs, The Organization of Networks in Plan 9*

### 7.1.1 – What is the kernel?

The kernel is a service that provides processes and resources to users active on an individual machine. Every Plan 9 machine boots a kernel.

At boot time the kernel takes on the identify of `$user` (the user who logs in at the console), which becomes the `hostowner` of the system. The `hostowner` in turn 1.) controls access to the kernel's resources, 2.) serves as the auth identity (`authid`) of the machine and the services it provides.

**Note:** The `hostowner` differs from the concept of `root` on a UNIX system, where a single user `root` may take control of all processes *and* files on the system. By contrast, even the `hostowner` of a Plan 9 file server cannot violate file permissions on the file server, except when permissions checking is disabled on the console or when entering special commands at the console of the file server. The `hostowner` controls only the *processes* running on the local machine. This fundamental separation between control of processes and file permissions is exploited throughout the Plan 9 system, but can be confusing for users coming from a UNIX background.

### 7.1.2 – What is the file server?

In a traditional Plan 9 network there is one disk file server, typically the only machine with a physical hard disk, that serves files to all other machines on the network. In most cases, other machines are either diskless or only use their disks for local caching. Ken Thompson's original Plan 9 file server ran a unique, special-purpose kernel that *only* served files, and whose configuration could only be changed at the console. In 9front, the file server runs a normal kernel and typically also runs as a `cpu` server (for remote access).

9front supports two different disk file systems for use on the file server: `cwfs` and `hjfs`. `cwfs` is a userspace port of Ken Thompson's original Plan 9 file server. `hjfs` is a new, experimental file server that stores both the cache and worm on a single partition (and thus requires less disk space to be used effectively). Both are reasonably robust.

Read: *The Plan 9 File Server* (deprecated, but partially applies to `cwfs`), `cwfs(4)`, `hjfs(4)`

**Note:** Since most Plan 9 systems have no disk, security of the file server is largely protected from breaches of security in its clients. The fewer the programs that run on the file server, the more isolated it can be from security holes in programs.

**Note:** Users seeking access to the file server must be added as a user on the file system

itself, and, if auth is enabled, added to the auth server's user database.

**Note:** Some users choose to run remote cpu or auth servers as stand-alone systems, each with their own local disk file systems. The distinction between all these types of systems is fuzzy and can become even fuzzier as services are enabled and disabled in different combinations.

### 7.1.3 – What is the auth server?

The auth server manages authentication for an entire Plan 9 network. It boots a normal kernel but is usually run on a separate, diskless machine that performs no other functions, in order to reduce the danger of a security breach compromising its kernel processes. That said, the auth server is usually also configured as a cpu server, for remote access.

**Note:** The `cron(8)` service should be run only on the auth server, where it can authenticate itself to access any of the other machines on the network.

Read: *Security in Plan 9*, `auth(8)`

### 7.1.4 – What is the cpu server?

The cpu server is used for remote computation. A cpu server's kernel runs processes in isolation, on only that machine. The boot process of a cpu server (defined as such by setting `service=cpu` in the machine's `plan9.ini` or equivalent) may be examined by reading the `/rc/bin/cpurc` script, which is executed at boot time. Running as a cpu server causes the kernel to adjust certain resource values that ultimately determine the behavior of the machine. For example, the `cpurc` script starts certain programs only if the machine is recognized as a cpu server.

Common use cases for a separate cpu server are: To execute programs compiled for a different architecture than that of the terminal; To execute programs closer to the data they are operating upon (for example, if the terminal is running over a slow link but the cpu server is on the same ethernet segment as the file server); To execute processes in physical isolation from other processes. In the early days of Plan 9, a cpu server was often significantly more powerful than the (often, special-purpose) hardware used for diskless terminals. Today, terminals are typically powerful computers in their own right, and the need for a separate machine running only as a cpu server is less common. That said, it can be useful to execute unstable or unpredictable programs on a separate machine so that frequently crashing and/or rebooting does not affect one's immediate workspace environment—especially when testing new code. In the case of remote (mail, web, etc.) servers, it is also likely that cpu access would be desired.

In practice, the disk file server, the auth server, and even some terminals will often run their own cpu listeners, to enable remote access to the processes controlled by their kernels.

**Note:** Users seeking access to a cpu server must first be added on the file system of the cpu server's corresponding file server (for permission to access and modify files) as well as the user database of its designated auth server (for login authentication).

Read: *The Organization of Networks in Plan 9*, `cpu(1)`, `exportfs(4)`

### 7.1.5 – What is a terminal?

The terminal is the machine at which the Plan 9 user is most often physically located. Usually diskless, the terminal will almost always run with graphics enabled (for launching the `rio` GUI or other graphical programs). The boot process of a terminal (defined as such by setting `service=terminal` in the machine's `plan9.ini` or equivalent) may be examined by reading the `/rc/bin/termrc` script, which is executed at boot time.

**Note:** Many Plan 9 users run stand-alone systems that operate — effectively — as a combined terminal and file server. For example, inside a virtual machine such as `qemu`, or booted from hard disk on a laptop. In this case the Plan 9 network is entirely self-contained, running one kernel on one machine, which renders `auth` and `cpu` services superfluous. This configuration trades some of the inherent security of separate hardware and kernel boundaries for the convenience of combining the whole system into a single, bootable instance.

**Note:** Terminal users who do not run stand-alone machines or who wish to access Plan 9 network resources must first be added to the file system of the network's file server, and to the user database of the network's `auth` server.

## 7.2 – Kernel configuration and maintenance

### 7.2.1 – How do I mount the 9fat partition?

`9front` has done away with the scripts `9fat:`, `c:`, and so forth, that are found in the Bell Labs Plan 9 distribution. Instead, use the `9fs` script to mount the 9fat partition:

```
9fs 9fat
```

If you are not at the console, or if `#S` has not already been bound over `/dev`:

```
bind -b '#S' /dev # bind the local hard drive kernel device over /dev
9fs 9fat /dev/sdXX/9fat # specify the full path to the corresponding 9fat
```

**Note:** `9fs 9fat` posts a file descriptor in `/srv/dos`. If this file already exists and is already in use, `9fs 9fat` will fail. If no other process is using the file it is safe to simply remove it and run `9fs 9fat` again.

Read: `dosrv(4)`

### 7.2.2 – How do I modify plan9.ini?

Mount the 9fat partition and then edit the file `/n/9fat/plan9.ini`.

**Note:** The file must end with a newline.

Read: `plan9.ini(8)`

### 7.2.3 – Kernel configuration file

Kernel configuration files are stored in the kernel directory and share the name of the kernel to which they apply. For example, the configuration file for the pc kernel is `/sys/src/9/pc/pc`.

### 7.2.4 – Kernel drivers

Kernel driver source files are located in the kernel source directory. For example, the pc kernel source is located in `/sys/src/9/pc`.

### 7.2.5 – How do I install a new kernel?

To build and install the new kernel(s) on the file system:

For 386:

```
cd /sys/src/9/pc
mk install # kernel is copied to /386/9pc
```

For amd64:

```
cd /sys/src/9/pc64
mk install # kernel is copied to /amd64/9pc64
```

For bcm (Raspberry Pi, etc.):

```
cd /sys/src/9/bcm
mk install # kernel is copied to /arm/9pif
```

For 386 and amd64 machines with local disk, it may be desired to install the new boot-loader and kernels onto the 9fat partition, in order to boot directly from disk. **Note:** The bootloader needs to be continuous on disk, so simply copying over the original file does not produce the desired effect. Instead:

```
9fs 9fat
rm /n/9fat/9bootfat
cp /386/9bootfat /n/9fat/
chmod +al /n/9fat/9bootfat # defrag magic
```

then copy the desired kernels:

For 386:

```
cp /386/9pc /n/9fat/
```

For amd64:

```
cp /amd64/9pc64 /n/9fat/
```

Finally, if a different kernel is being installed than the one currently running, edit `plan9.ini` and change `bootfile` to point to the new kernel.

Read: *FQA 7.2.2 – How do I modify plan9.ini?*

## 7.3 – Fileserver configuration and maintenance

### 7.3.1 – Adding users

Add a new user on the file server:

For cwfs:

```
echo newuser username >>/srv/cwfs.cmd
```

For hjfs:

```
echo newuser username >>/srv/hjfs.cmd
```

If needed, make the new user a member of another group (example: upas):

For cwfs:

```
echo newuser upas +username >>/srv/cwfs.cmd
```

For hjfs:

```
echo newuser upas +username >>/srv/hjfs.cmd
```

Both file servers store their user database in `/adm/users`. Examine this file, and the contents of the `/usr` directory, to evaluate success.

**Note:** It is also possible to access the control file interactively:

For cwfs:

```
con -C /srv/cwfs.cmd
```

For hjfs:

```
con -C /srv/hjfs.cmd
```

From here commands may be entered directly.

Type `Ctrl-\` to resume the `con` prompt, followed by `q` to quit.

**Note:** New users are created without a profile, mail directory, `tmp` directory (needed to edit files with `sam`) or other confections. To install a default profile for a new user, upon first login as that user, run:

```
/sys/lib/newuser
```

then edit `/usr/username/lib/profile` to your own specifications. The `newuser` file system command is described in the man pages `fs(8)` (for `cwfs`) and `hjfs(8)`. The default system `/lib/namespace` does the following:

```
bind -c /n/other/usr/$user/tmp /usr/$user/tmp
```

For cwfs users, it may be desirable to store the user's tmp directory on the other partition:

```
mkdir /n/other/usr/$user/tmp
```

### 7.3.2 – Configuring nvram

The cpu kernel checks the nvram file for valid auth credentials and attempts to copy them into factotum so that the machine may boot without manual intervention. To configure the nvram, run the command `auth/wrkey`, which will prompt for an `authid`, `authdom`, `secstore key`, and `password`. The `authid` is a synonym for the `hostowner` of the machine and should be a valid user that has already been (or will be) added to the corresponding auth server, in this case `glenda`. The `authdom` is the authentication domain for the machine, in this case `9front`. The `secstore key` and `password` are secret passwords of eight characters or more in length. The `password` is the password belonging to the `authid` user on the auth server responsible for the `authdom` entered above. The `secstore key` is the password of the user on the secure-store server (Read: *FQA 7.4.3 – secstored*). If the `secstore` client (Read: *FQA 8.4.7 – secstore*) is not being used on this machine (for example, if this is the auth server where `secstored` will run), just hit `enter` at the `secstore key:` prompt.

Run the command `auth/wrkey`:

```
bad nvram key
bad authentication id
bad authentication domain      # You may not see these errors.
authid: glenda
authdom: 9front
secstore key: [glenda's secstore password]
password: [glenda's password]
```

**Note:** Booting the file system with authentication enabled and an invalid nvram file will cause `auth/wrkey` to be run automatically at startup.

Read: `auth(8)`

### 7.3.3 – Setting up a listener for network connections

In order for remote machines to mount the file system of the file server, the file server must first be running a network listener. This section details the steps required to transform a terminal with disk (the result of a default install of `9front`) into a disk file server for other machines.

The first step is to switch from the terminal service to the cpu service by editing the service line in `/n/9fat/plan9.ini`:

```
service=cpu
```

Read: *FQA 7.2.2 – How do I modify plan9.ini?*

Before rebooting, configure the nvram: *FQA 7.3.2 – Configuring nvram*. This allows the machine to load auth credentials from the nvram file into factotum, so that it can

continue to boot without manual intervention.

Reboot:

```
fshalt -r
```

The next step (on `cwfs`; not needed on `hjfs`) is to enable authentication on the file server, to prevent unauthorized users from accessing the disk over the network. At the `bootargs` prompt, retype the default and add the `-c` flag to enter the file server's config mode. At the `config` prompt, type `noauth` twice to toggle authentication on the file server. Finally, type `end` to continue with the boot process:

```
bootargs is (tcp, local!device) [local!/dev/sdXX/fscache] local!/dev/sdXX/fscach
config: noauth
auth is now disabled
config: noauth
auth is now enabled
config: end
```

The machine will now continue to boot.

Once booted, the next step is to configure the file server to listen for connections from remote hosts. Modify the `bootargs` of the file server in `/n/9fat/plan9.ini`:

For `cwfs`:

```
bootargs=local!/dev/sdXX/fscache -a tcp!*!564
```

For `hjfs`:

```
bootargs=local!/dev/sdXX/fs -m 702 -A -a tcp!*!564
```

**Note:** The `-m 702` flag for `hjfs` allocates 702 megabytes of memory to be used as a cache. This value is typically automatically calculated by the 9front installer, and may differ on your system. There is no need to change whatever default was already configured.

Read: *FQA 7.2.2 – How do I modify plan9.ini?*

Reboot the file server:

```
fshalt -r
```

When the system finishes booting it should now be listening for network connections to the file system. Users who have been added to the file server and the auth server should now be able to authenticate and mount the file server (`tcp boot`, etc.).

Read: `cwfs(4)`, `hjfs(4)`, *FQA 6.7.1 – How do I tcp boot?*

### 7.3.4 – Mounting a file system from userspace

For `cwfs`:



```
% cwfs64x -n fs -f /dev/sdE0/fscache      # use the correct path to your fscache
% mount /srv/fs /n/fs
```

**Note:**

Running the above commands will post the file systems's console in /srv/fs.cmd.

For hjfs:

*to be announced*

## 7.4 – Auth server configuration and maintenance

### 7.4.1 – Configuring an auth server

The auth server should be booted with `service=cpu` in `plan9.ini`, and `ndb` modified to associate the new auth server with the desired `authdom`.

If the `cpu` server machine boots from a local disk, edit the `service` line in in `/n/9fat/plan9.ini`:

```
service=cpu
```

Read: *FQA 7.2.2 – How do I modify plan9.ini?*

If the machine boots via PXE, edit the `service` line in in the file under `/cfg/pxe/` that correspondes to its MAC address. In this case, `/cfg/pxe/000c292fd30c`:

```
service=cpu
```

**Note:** The contents of `/cfg/pxe/000c292fd30c` serves as the equivalent of `plan9.ini` for the PXE booted machine. Any other settings that would normally be configured in `plan9.ini` may also be entered there.

Next, `ndb` must be modified to associate the new auth server with the desired `authdom`. Assuming the auth server has a MAC address of `00:0c:29:2f:d3:0c`, an IP address of `192.168.0.2`, and a default gateway/DNS server of `192.168.0.1` that are all on the Class C network `192.168.0.0/24`, and that the `authdom` is `9front`, edit `/lib/ndb/local` and add the `authdom` and the auth server's IP under the corresponding `ipnet`:

```
ipnet=9front ip=192.168.0.0 ipmask=255.255.255.0
    ipgw=192.168.0.1
    auth=192.168.0.2 # add auth server's ip
    authdom=9front # add authdom
    fs=192.168.0.3
    cpu=192.168.0.4
    dns=192.168.0.1
    dnsdomain=9front
    smtp=192.168.0.4
```

Read: `ndb(6)`

Before rebooting, configure the nvram: *FQA 7.3.2 – Configuring nvram*. This allows the machine to load auth credentials from the nvram file into factotum, so that it can continue to boot without manual intervention.

**Note:** If the auth server's hostowner (referred to as authid in the auth/wrkey dialogue) will be any other user than the default glenda, that user must be authorized (in the auth context) to "speak for" other users. Assuming a hostowner of sl, add a rule to /lib/ndb/auth:

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

This rule allows the user sl to speak for all users *except for* sys and adm.

Read: auth(8)

Reboot:

```
fshalt -r
```

At boot time, the shell script /rc/bin/cpurc consults ndb to determine if the machine is an auth server. If it is, the script will launch the keyfs process and start listeners for auth connections. If, after booting, keyfs is not running, something went wrong.

Finally, create an auth user and configure an auth password for the hostowner of the machine. This auth user should be the same name as the authid that was entered at boot time during the auth/wrkey dialogue. Likewise, set the password to match the password that was entered during the auth/wrkey dialogue. **Note:** If the user and password do not match what was entered during the auth/wrkey dialogue, users will not be able to authenticate using this auth server.

Read: *FQA 7.4.2 – Adding users*

#### 7.4.1.1 – Avoiding an ndb entry for the auth server

If an auth server for a given authdom is not found in the local ndb, then the authdial() function from the libauthsrv library (used for resolving auth servers) will default to the dns host name p9auth.example.com, where p9auth is the sub-domain, and example.com is the authdom. This convention (where followed) is useful to avoid having to manually add auth server information for arbitrary remote networks to the local ndb.

#### 7.4.2 – Adding users

To add a new user to the auth server, login as the auth server's hostowner, make sure auth/keyfs is running in your namespace, and then set an auth password for the user:

```
% auth/keyfs
% auth/changeuser username
Password: # type password here, will not echo
Confirm password: # confirm password here, will not echo
assign Inferno/POP secret? (y/n) n
Expiration date (YYYYMMDD or never)[return = never]:
2 keys read
Post id:
User's full name:
Department #:
User's email address:
Sponsor's email address:
user username installed for Plan 9
```

**Note:** Questions that appear after the `keys read` notice are optional. Hit Enter for each one to leave them blank.

Read: `auth(8)`, `keyfs(4)`

### 7.4.3 – secstored

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.

To set up `secstored`, login to the auth server as `hostowner` and:

```
mkdir /adm/secstore
chmod 770 /adm/secstore
```

Start `secstored` at boot time by adding the following to `/cfg/$sysname/cpurc` on the auth server:

```
auth/secstored
```

Read: `secstore(1)`, `secstore(8)`

#### 7.4.3.1 – Adding users to secstore

`secuser` is an administrative command that runs on the secstore machine, normally the auth server, to create new accounts and to change status on existing accounts. It prompts for account information such as password and expiration date, writing to `/adm/secstore/who/user` for a given secstore user.

Login to the auth server as `hostowner` and:

```
auth/secuser username
```

and answer the prompts.

By default, `secstored` warns the client if no account exists. If you prefer to obscure this information, use `secuser` to create an account `FICTITIOUS`.

Read: *FQA 8.4.7 – secstore* for more information on using the secstore client.

### 7.4.3.2 – Converting from ps9k1 to dp9ik

Date: Wed, 6 Jan 2016 03:54:08 +0100  
From: cinap\_lenrek@felloff.net  
To: 9front@9front.org  
Subject: [9front] new factotum/authsrv/keyfs  
Reply-To: 9front@9front.org

i just pushed the new code which adds dp9ik authentication support.

to update a system, the following things need to be done:

```
# make sure you have the latest libmp/libsec
cd /sys/src/libmp; mk install
cd /sys/src/libsec; mk install

# rebuild mpc (required for libauthsrv)
cd /sys/src/cmd; mk mpc.install

# rebuild libauthsrv / libauth
cd /sys/src/libauthsrv; mk install
cd /sys/src/libauth; mk install

# rebuild factotum/keyfs/authsrv
cd /sys/src/cmd/auth; mk install

# then rebuild kernel to include the new factotum,
# but dont reboot your authserver just yet...
cd /sys/src/9/pc; mk install

# if your /adm/keydb is still in DES format (cat it to see
# if the keyfile starts with the AES signature), you need to
# convert it to use the new dp9ik protocol:

# make backup
cp /adm/keys /adm/keys.old
auth/convkeys -ap /adm/keys

# now set the aes key in nvram (so authserver can decrypt
# the keydb when it boots)
auth/wrkey

# now you can reboot the AS and once its up, you have to
# set new passwords for the users. logging in with the
# old p9sk1 plan9 password should continue to work if
# you skip this.
passwd [username]

# if there are issues logging in with dp9ik because keydb
# doesnt have the new key yet, you can use delkey(1) to
# remove the dp9ik key from factotum as a work arround.

--
cinap
```

## 7.5 – Cpu server configuration and maintenance

### 7.5.1 – Configuring a cpu server

**Note:** Operating a cpu server requires auth services. Read: *FQA 7.4 – Auth server configuration and maintenance*

The first step in converting a terminal to a cpu server is to switch from the terminal service to the cpu service.

If the cpu server machine boots from a local disk, edit the service line in in `/n/9fat/plan9.ini`:

```
service=cpu
```

Read: *FQA 7.2.2 – How do I modify plan9.ini?*

If the machine boots via PXE, edit the service line in in the file under `/cfg/pxe/` that correspondes to its MAC address. In this case, `/cfg/pxe/000c292fd30c`:

```
service=cpu
```

**Note:** The contents of `/cfg/pxe/000c292fd30c` serves as the equivalent of `plan9.ini` for the PXE booted machine. Any other settings that would normally be configured in `plan9.ini` may also be entered here.

Setting `service=cpu` causes the shell script `/rc/bin/cpurc` to be run at boot time, which in turn launches a listener that scans the `/rc/bin/service` directory for scripts corresponding to various network ports. Read: `listen(8)`. The script `tcp17019` handles incoming cpu connections. Authentication for incoming cpu connections is performed by the auth server associated with the authdom by `ndb`. Read: *FQA 7.4.1 – Configuring an auth server*

Before rebooting, configure the nvram: *FQA 7.3.2 – Configuring nvram*. This allows the machine to load auth credentials from the nvram file into `factotum`, so that it can continue to boot without manual intervention.

Reboot:

```
fshalt -r
```

## 7.6 – Terminal configuration and maintenance

### 7.6.1 – Configuring a terminal

The 9front ISO boots into a `livedd` running the `9pc` kernel, resulting in the simplest form of terminal running on the 386 architecture. A terminal may also be network booted (the preferred method) or installed to its own stand-alone file system on a local storage device.

Read: *FQA 6.7 – How do I boot from the network?*

## 7.6.2 – Configuring a Terminal to Accept cpu Connections

If the `hostowner` `factotum` has been loaded with the appropriate key and the system is listening for `cpu` connections, a user may `cpu` into a terminal that is not running auth services. To configure a terminal to accept `cpu` connections in this fashion, substitute your choice of `dom` (this refers to the `authdom`), `user` and `password`, below:

```
echo 'key proto=dp9ik dom=9front user=glenda !password=p@ssw0rd' \  
>/mnt/factotum/ctl  
aux/listen1 -t 'tcp!*!rcpu' /rc/bin/service/tcp17019
```

## 7.6.3 – UTC Timesync

By default, `/rc/bin/termrc` sets `TIMESYNCARGS=(-rLa1000000)`, to synchronize `9front` time with the real time clock. On many systems this time is saved as UTC, whereas Windows keeps the local time there. If your time is in UTC you should omit the `-L`: Put `TIMESYNCARGS=(-ra1000000)` into `/rc/bin/termrc.local`, which is executed by `/rc/bin/termrc`.

## 7.7 – Mail server configuration and maintenance



Incoming and outgoing mail is handled by `upas` and its related suite of programs. Configuration is handled by a number of files found in `/mail/lib/`, while many of `upas`' common functions are carried out by shell scripts that are (relatively) easy to modify.

**Note:** The user who runs the assorted `upas` programs needs read and write permissions on `/mail/queue` and `/mail/tmp`, as well as write permissions for any mailboxes where mail will be delivered.

**Note:** Be sure to configure proper DNS entries for your domains. If Plan 9 will host your DNS, see: *FQA 6.2.5.2 – DNS authoritative name server*

Read: *Upas – A Simpler Approach to Network Mail*, mail(1)

The following sections describe configuration of basic Internet mail services.

### 7.7.1 – smtpd.conf

Some changes to the default `smtpd.conf` are required to accept mail *for* Internet domain names, and to relay mail *for* remote hosts (most commonly, your own machines). The following lines should be changed to correspond to your network:

```
# outgoing mail will be sent from this domain by default
defaultdomain          9front.org

# do not be an open relay
norelay                on

# disable dns verification of sender domain
verifysenderdom       off

# do not save blocked messages
saveblockedmsg        off

# if norelay is on, you need to set the
# networks allowed to relay through
# as well as the domains to accept mail for
ournets 199.191.58.37/32 199.191.58.42/32 192.168.4.0/24

# domain names for which incoming mail is accepted
ourdomains 9front.org, bell-labs.co, cat-v.org
```

Read: `smtpd(6)`, `smtp(8)`

### 7.7.2 – rewrite

To act as an Internet mail server, copy `rewrite.direct` to `rewrite` and modify to reflect your site's Internet domain name(s):

```
# case conversion for postmaster
pOsTmAsTeR alias postmaster

# local mail
\l!(.*) alias \1
(ttr|9front.org|bell-labs.co|cat-v.org)!(.*) alias \2
[^\!@]+ translate "/bin/upas/aliasmail '&'"
local!(.*) >> /mail/box/1/mbox

# we can be just as complicated as BSD sendmail...
# convert source domain address to a chain a@b@c@d...
@[^\!@,]*):([^\!@]*)@([^\!]* ) alias \2@\3@\1
@[^\!@]*),([^\!@,]*):([^\!@]*)@([^\!]* ) alias @\1:\3@\4@\2

# convert a chain a@b@c@d... to ...d!c!b!a
([^\!@]+)@([^\!@]+)@(.+) alias \2!\1@\3
([^\!@]+)@([^\!@]+) alias \2!\1

# /mail/lib/remotemail will take care of gating to systems we don't know
([^\!]*)!(.*) | "/mail/lib/qmail '\s' 'net!\1'" "\2"
```

Read: rewrite(6)

### 7.7.3 – names.local

To map incoming e-mail addresses to local usernames, edit `names.local` accordingly:

```
# postmaster goes to glenda
postmaster glenda
```

**Note:** `postmaster@[any domain]` will be delivered to local user `glenda`.

### 7.7.4 – remotemail

Finally, `upas` needs to know what to do with mail that cannot be delivered locally. Edit `remotemail` and enter the desired behavior.

To deliver mail directly to the remote server responsible for the Internet domain name in question:

```
#!/bin/rc
shift
sender=$1
shift
addr=$1
shift
exec /bin/upas/smtp $addr $sender $*
```

Read: smtp(8)



### 7.7.5 – SMTP over TLS

First, make sure you have already created TLS certificates for your server.

Next, create a file `/rc/bin/service/tcp587:`

```
#!/bin/rc
user='{cat /dev/user}
exec /bin/upas/smtpd -c /sys/lib/tls/cert -n $3
# to use with listen1, change $3 to $net
```

### 7.7.6 – IMAP4 over TLS

First, make sure you have already created TLS certificates for your server.

Next, create a file `/rc/bin/service/tcp993:`

```
#!/bin/rc
exec tlssrv -c/sys/lib/tls/cert -limap4d \
    -r'{cat $3/remote} /bin/ip/imap4d -p \
    -r'{cat $3/remote} >>[2]/sys/log/imap4d
# to use with listen1, change $3 to $net
```

### 7.7.7 – Spam Filtering

#### 7.7.7.1 – ratfs

From `ratfs(4)`:

Ratfs starts a process that mounts itself (see `bind(2)`) on mountpoint (default `/mail/ratify`). Ratfs is a persistent representation of the local network configuration and spam blocking list. Without it each instance of `smtpd(6)` would need to reread and parse a multimegabyte list of addresses and accounts.

To configure the spam blocking list, edit `/mail/lib/blocked` as desired, according to the rules laid out in the man page. Example:

```
# allow messages from any user at 9front.org
*allow 9front.org!*

# block messages from any user at bell-labs.com
*block bell-labs.com!*

# block messages from ip block of aol modems
block 152.166.0.0/15
```

If `ratfs` is already running, cause it to reload the modified `/mail/lib/blocked`:

```
echo reload >/mail/ratify/ctl
```

For more details, read: `ratfs(4)`, `smtpd(6)`

To launch `ratfs` at boot time, add the following line to `/cfg/$sysname/cpustart`:

upas/ratfs

and add the following line to `/lib/namespace`:

```
mount -c #s/ratify /mail/ratify
```

**Note:** The directory served by `ratfs` must be visible from the `upas` listener's namespace. Usually, this is accomplished by starting `ratfs` *before* the `upas` listeners.

### 7.7.7.2 – scanmail

Read: `scanmail(8)`

## 7.7.8 – Troubleshooting the mail server

An online tool that evaluates the configuration of a given mail server is available at: <https://www.mail-tester.com>

## 7.8 – Web server configuration and maintenance

If you must.

### 7.8.1 – ip/httpd

No.

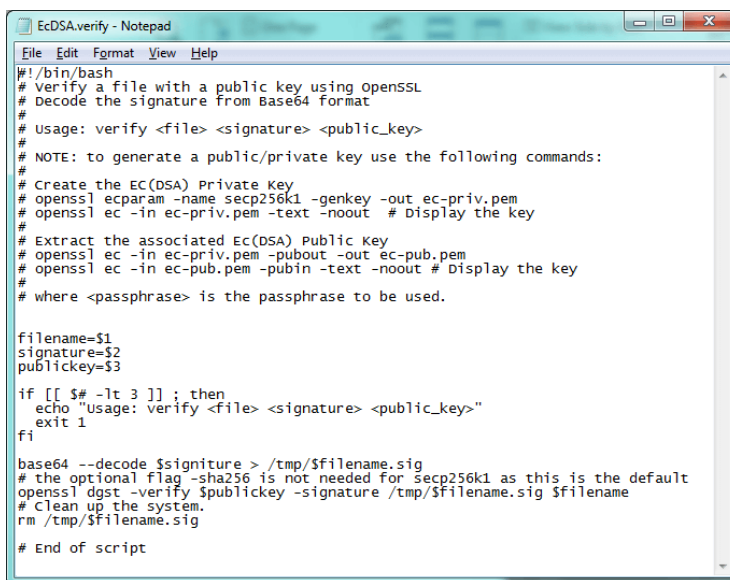
### 7.8.2 – rc-httpd

The `rc-httpd` web server is a simple shell script that handles static files, directory listings and drop-in CGI programs such as the `werc` anti-framework. `rc-httpd` is run from a file in the directory scanned by `listen(8)`, or called as an argument to `listen1(8)`.

Read: `rc-httpd(8)`

**Note:** `rc-httpd` is employed to serve the `9front.org` family of websites.

## 7.9 – TLS certificates



```
EcDSA.verify - Notepad
File Edit Format View Help
#!/bin/bash
# Verify a file with a public key using OpenSSL
# Decode the signature from Base64 format
#
# Usage: verify <file> <signature> <public_key>
#
# NOTE: to generate a public/private key use the following commands:
#
# Create the EC(DSA) Private Key
# openssl ecparam -name secp256k1 -genkey -out ec-priv.pem
# openssl ec -in ec-priv.pem -text -noout # Display the key
#
# Extract the associated Ec(DSA) Public key
# openssl ec -in ec-priv.pem -pubout -out ec-pub.pem
# openssl ec -in ec-pub.pem -pubin -text -noout # Display the key
#
# where <passphrase> is the passphrase to be used.

filename=$1
signature=$2
publickey=$3

if [[ $# -lt 3 ]]; then
    echo "usage: verify <file> <signature> <public_key>"
    exit 1
fi

base64 --decode $signature > /tmp/$filename.sig
# the optional flag -sha256 is not needed for secp256k1 as this is the default
openssl dgst -verify $publickey -signature /tmp/$filename.sig $filename
# Clean up the system.
rm /tmp/$filename.sig

# End of script
```

To use TLS-enabled services on a Plan 9 mail server (poptls, apoptls, imaps, etc.) you need to generate a certificate and key for your mail server and tell the factotum of the server about that key. The following example creates a self-signed certificate:

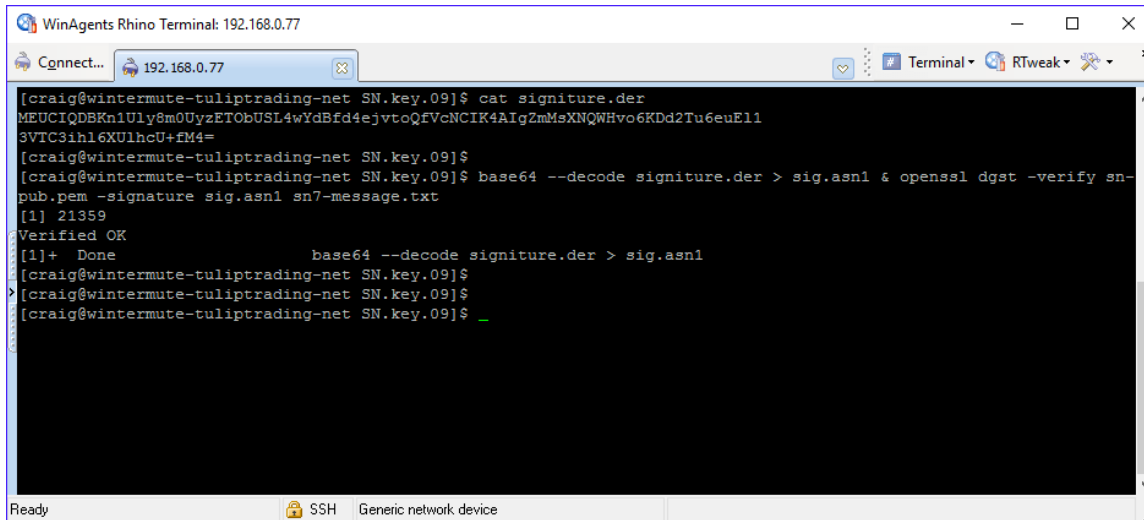
```
ramfs -p
cd /tmp
auth/rsagen -t 'service=tls role=client owner=*' > key
chmod 600 key
cp key /sys/lib/tls/key # or: store key in secstore
auth/rsa2x509 .C=US CN=fakedom.dom' /sys/lib/tls/key | \
    auth/pemencode CERTIFICATE > /sys/lib/tls/cert
```

**Note:** Here, US is the two-digit country code, and fakedom.dom is the fully qualified domain name.

To load the key into the server's factotum at boot time, add the following line to /cfg/\$sysname/cpustart:

```
cat /sys/lib/tls/key >>/mnt/factotum/ctl
```

Read: rsa(8)



```
WinAgents Rhino Terminal: 192.168.0.77
Connect... 192.168.0.77 Terminal RTweak
[craig@wintermute-tuliptrading-net SN.key.09]$ cat signiture.der
MEUCIQDBKn1Uly8mOUyzETObUSL4wYdBfd4ejvtoQfVcNCIK4AIgZmMsXNQWHvo6KDd2Tu6euE11
SVTC3ihl6XUlhU+fM4=
[craig@wintermute-tuliptrading-net SN.key.09]$
[craig@wintermute-tuliptrading-net SN.key.09]$ base64 --decode signiture.der > sig.asn1 & openssl dgst -verify sn-
pub.pem -signature sig.asn1 sn7-message.txt
[1] 21359
Verified OK
[1]+ Done base64 --decode signiture.der > sig.asn1
[craig@wintermute-tuliptrading-net SN.key.09]$
[craig@wintermute-tuliptrading-net SN.key.09]$
[craig@wintermute-tuliptrading-net SN.key.09]$ _
```

Ready SSH Generic network device