dinit

dinit is a service supervisor with dependency support which can also act as the system "init" program. It was created with the intention of providing a portable init system with dependency management, that was functionally superior to many extant inits.

Contents

Installation

Install the `dinit` package.

Installation of services

For server or background software, Artix repositories normally provide a complementary `*-dinit` package that contain service files for dinit. For example, `clamav` has `clamav-dinit`. System services are stored in `/etc/dinit.d`, and enabled with the command (as root) `# dinitctl enable service_name`. User service files are in `/etc/dinit.d/user`, enabled (as user) with `$ dinitctl enable service_name`. Packages do not enable themselves, although there can be exceptions.

Some packages on Arch Linux's AUR have complementary dinit packages. If you can't find a service file for custom software, writing it is often as easy as in systemd.

Overview

Dinit will invoke strong feelings of déjà vu in those who are familiar with systemd, especially when using its main tool, `dinitctl`, or dealing with service files. Although dinit was inevitably influenced by other software, it should be treated as a completely different program. Its advantages are a pretty minimalistic feature set, simple service file format, ease of use and portability across *nix operating systems. You'll enjoy very good booting speed. As of disadvantages, there are few distros that support dinit officially.

Dinit can launch multiple services in parallel, with dependency management (i.e. if one service's operation depends on another, the latter service will be started first). It can monitor the process corresponding to a service, and re-start it if it dies, and it can do this in an intelligent way - first "rolling back" all dependent services, and restarting them when their dependencies are satisfied. The *dinitctl* tool can be used to start or stop services and check their state.

Dinit is designed to run as either as a system service manager (runs as root, uses system paths for configuration) or a user process (runs as a user, uses paths in the user's home directory for configuration).

Usage

Nearly every interaction with dinit are done with the `dinitctl` program.

- Start service: `# dinitctl start service_name`
- Stop service: `# dinitctl stop service_name`
- Reload service description: `# dinitctl reload service_name`
- Restart service: `# dinitctl restart service_name`
- Enable service: `# dinitctl enable service_name`
- chrooted: `# ln -sf /etc/dinit.d/service_name /etc/dinit.d/boot.d/`
- Disable service: `# dinitctl disable service_name`
- List services: `# dinitctl list`

Some usage may resemble commands of *systemctl* from systemd. Keep in mind that `dinitctl` commands have slighly different behavior. For example, `# dinitctl enable service_name` starts the service immediately instead of just putting it in auto-start.

`dinit-check` is usually used if to check if there is any problem with the dinit service directory, it will check for any syntax errors, invalid parameter values, and dependency cycles.

CLI tools

- `dinit-check` - check service configuration
- `dinitctl` - control services supervised by Dinit
- `dinit-monitor` - monitor services or environment supervised by Dinit

Files/Directories

- `/etc/dinit.d/` - user-installed dinit service files

- `/usr/lib/dinit.d/` - system dinit service files
- `/etc/dinit.d/config/` - configurations for dinit services
- `/etc/dinit.d/boot.d/` - dinit services that run on boot
- `/usr/lib/dinit/` - dinit wrapper scripts
- `/var/log/dinit/` - logs for dinit services
- `/etc/dinit.d/service_name-pre` - preparation service, in some cases, this is needed because the "preparation" must be run as root but the service itself must be run as its own user
- `/etc/dinit.d/user` - user services (see User services)

Service states

`dinitctl list` depicts current state of all services as ASCII toggles:

```
[[+]      ] boot
```

This is the first service in your list. The `[+]` indicates that the service is currently up.

```
[{+}      ] udevd (pid: 4)
```

All services except *boot* use `{+}` or `{-}` instead, as they are all ultimately dependent on *boot*. If active services are depicted as toggled "on" (on the left), then stopped services are toggled "off" (on the right):

```
[      {-}] mysql
```

Services in transition (or waiting for their dependencies to start/stop) have arrows:

```
[[ ]<<    ] mysql      # starting (and marked active)
[    >>{ }] mysql      # stopping
```

The brackets indicate the target state, which may not be the state to which the service is currently transitioning. For example:

```
[    <<{ }] mysql      # starting, but will stop after starting
[{ }>>    ] mysql      # stopping, but will restart once stopped
```

Service file structure

A typical service file looks like this:

```
type = process | bgprocess | scripted | internal
command = /path/to/servicename
restart = (boolean)
smooth-recovery = (boolean)
logfile = ...
```

```
depends-on = (service name)
waits-for = (service name)
```

There are five types of dinit services:

- `process`, for foreground daemons
- `bgprocess`, for forking daemons
- `scripted`, for oneshots
- `internal`, which is only useful inside dinit and is usually used to "gather" lots of dependencies into one big dependency, or similar to "bundle" in s6.
- `triggered`, which is similar to internal services, but an external trigger is required before they will start.

There are three types of dependencies:

- `depends-on` is hard dependency, so if any service is named here, it must run no matter what before the service is started
- `depends-ms` is a milestone dependency, so while the rules of depends-on apply when starting service, if the dependency stopped, the dependent will still run
- `waits-for` is soft dependency, while the service will wait for its dependency to run, it will still run if the dependency fails

For more details, see `dinit-service(5)`.

User services

Some essential services, such as D-Bus and PipeWire, run as user-level processes. To automatically run these processes for your users, you must use either dinit-user-spawn or turnstile, which will spawn user instances of dinit. Avoid using both simultaneously. To check that you run a user instance of dinit, run (as user):

```
$ dinitctl list
[[+]     ] boot
[{+}     ] pipewire-pulse (pid: 1211)
[{+}     ] pipewire (pid: 1209)
[{+}     ] dbus (pid: 1204)
[{+}     ] wireplumber (pid: 1210)
```

Using dinit-user-spawn

Install the `dinit-user-spawn` package:

```
sudo pacman -S dinit-user-spawn
```

If the service is not already enabled, then run the following as root:

```
dinitctl enable dinit-user-spawn
```

You should now immediately have a dinit instance running as your user.

Using turnstile

Install the `turnstile turnstile-dinit` packages:

```
sudo pacman -S turnstile turnstile-dinit
```

Enable `turnstile`

```
dinitctl enable turnstiled
```

Relogin and check `$ dinitctl list`.

Manual alternative

There is a third manual way. You can instead manually launch dinit or other service managers yourself, for example by having the following in your bashrc:

```
if ! pgrep -u "$USER" dinit > /dev/null; then
   dinit --user
fi
```

However, you will need to manually manage dinit, and create the relevant folders it expects.

tty handling

dinit handles tty through `getty` service. You can configure active ttys through `/etc/dinit.d/config/console.conf`

To modify individual ttys, you can copy (do **NOT** delete) `/etc/dinit.d/config/agetty-default.conf` to your desired tty (e.g. `/etc/dinit.d/config/agetty-tty1.conf`)

Inside, you'll see the contents like

```
# DO NOT REMOVE THIS FILE!
# Note: You can copy and rename this file to the name of the tty
you
#  want (e.g.: /etc/dinit.d/config/agetty-tty1.conf will make a
#  configuration specific to tty1)
GETTY_BAUD=38400
GETTY_TERM=linux
GETTY_ARGS=
```

You can modify the files according to `agetty(1)`, which allows one to autologin.

User Services needing "Display"

Generally make sure

```
$:cat ~/.config/dinit.d/environment
DISPLAY=:0
```

Explicitly reference the environment file in the service, where <USER> is the user path

```
env-file = /home/<USER>/.config/dinit.d/environment
```

Example

For example we create a service for installed `volumeicon`:

We make sure ~/.config/dinit.d/environment contains `DISPLAY=:0`

Edit as root

```
/etc/dinit.d/user/volumeicon
```

to

```
type = process
command = /usr/bin/volumeicon
logfile = $HOME/.local/state/dinit/volumeicon.log
env-file = /home/<USER>/.config/dinit.d/environment
restart = true
```

Make sure /.local/state/dinit/volumeicon.log exists.

```
mkdir ~/.local/state/dinit/
touch ~/.local/state/dinit/volumeicon.log
```

Check User service

```
$: dinit-check volumeicon
Checking service: volumeicon...
dinit-check: warning: variable substitution performed by dinit-check for file paths may not match dinit daemon (environment may differ); use --online to avoid this warning
Performing secondary checks...
Secondary checks complete.
No problems found.
```

Enable

```
$:dinitctl enable volumeicon
Service 'volumeicon' has been enabled.
```

```
Service 'volumeicon' started.
```

Check list

```
$:dinitctl list
[[+]     ] boot
[{+}     ] system
[{+}     ] dbus (pid: 875)
[{+}     ] pipewire (pid: 879)
[{+}     ] pipewire-pulse (pid: 881)
[{+}     ] wireplumber (pid: 880)
[{+}     ] volumeicon (pid: 16945)
```

Now if you restart pipewire `dinitctl restart pipewire --force` volumeicon will automatically restart

---