

# **Configuración del sistema de arranque de Mac OS X**

## Acerca de este documento

---

En este reportaje vamos a comentar el proceso de arranque que sigue Mac OS X, y como podemos configurar servicios de Mac OS X para que se ejecutan automáticamente al arrancar la máquina.

Este documento está escrito para Mac OS X 10.4 o posteriores. Si quiere configurar una versión anterior debe consultar el reportaje "SystemStarter".

El objetivo de este reportaje no es aprender a lanzar aplicaciones de usuario al arrancar el ordenador, sino otro tipo de aplicaciones conocidas como servicios o procesos de background, que son aplicaciones no visuales (normalmente procedentes del mundo de UNIX) que permanecen lanzadas todo el tiempo, y que dan servicios (p.e. servicios de red) como puedan ser: Apache, NFS, SSH, cron, etc.

Este reportaje está dirigido a usuarios familiarizados con sistemas UNIX, con lo que no vamos a explicar los detalles más comunes o conocidos de estos sistemas.

## Nota Legal

---

Este reportaje ha sido escrito por Fernando López Hernández para MacProgramadores, y de acuerdo a los derechos que le concede la legislación española e internacional el autor prohíbe la publicación de este documento en cualquier otro servidor web, así como su venta, o difusión en cualquier otro medio sin autorización previa.

Sin embargo el autor anima a todos los servidores web a colocar enlaces a este documento. El autor también anima a cualquier persona interesada en conocer el proceso de arranque de Mac OS X y su configuración a bajarse o imprimirse este reportaje.

Madrid, Mayo del 2007

Para cualquier aclaración contacte con:

[fernando@DELITmacprogramadores.org](mailto:fernando@DELITmacprogramadores.org)

## Tabla de contenido

---

Cambios introducidos por Mac OS X 10.4.....	4
El programa <code>launchd</code> .....	4
¿Por qué el cambio en el sistema de arranque? .....	4
El proceso de arranque.....	6
Configurar demonios <code>launchd</code> .....	7
Crear el fichero de propiedades del demonio .....	9
Ejemplo de demonio <code>launchd</code> .....	11
Testear un demonio.....	12
Usar <code>StartupItems</code> .....	13
Anatomía de los <code>StartupItems</code> .....	13
Crear el script de arranque / parada .....	13
Especificar los parámetros del <code>StartupItem</code> .....	15
Lanzar y parar un <code>StartupItem</code> desde la consola .....	16
Configurar el login y logout.....	18
Instalar un script usando <code>defaults</code> .....	18
Pasar opciones a <code>loginwindow</code> .....	19

## Cambios introducidos por Mac OS X 10.4

---

### El programa `launchd`

En Mac OS X 10.4 Apple introdujo un nuevo sistema de arranque coordinado por el programa `launchd`. Este programa asume las responsabilidades de las que hasta ahora se habían encargado otros programas más clásicos de UNIX como son `mach_init`, `init`, `xinetd` o `cron`. Antes de Mac OS X 10.4 el proceso de arranque estaba coordinado por los procesos `mach_init` e `init` los cuales ejecutaban el script `/etc/rc` que preparaba el sistema para el usuario. Aunque este script todavía existe, y se sigue usando para configurar el proceso de arranque, los comandos `mach_init` e `init` han sido eliminados y sustituidos por el comando `launchd` (situado en `/sbin/launchd`) que es el nuevo proceso raíz del sistema.

### ¿Por qué el cambio en el sistema de arranque?

Tradicionalmente en UNIX la responsabilidad de arrancar el sistema y configurar sus distintos procesos de background ha estado delegado al menos a tres programas distintos:

- 1) `init` que es el primer proceso que se ejecuta, y que se encarga de arrancar y configurar los servicios más básicos.
- 2) `xinetd` que es el encargado de lanzar los procesos de background que dan servicios de red. Aunque inicialmente los sistemas UNIX no usaban TCP/IP para comunicarse, con el tiempo se popularizó este protocolo, y fue cuando se creó `xinetd`. Una peculiaridad característica de `xinetd` es que puede lanzar los procesos de background bajo demanda, es decir, los procesos de background que dan servicios de red pueden estar siempre lanzados, pero también pueden lanzarse sólo cuando se conecta un cliente al puerto del servicio.
- 3) `cron` es un proceso que una vez lanzado se encarga de ejecutar otros procesos periódicamente, o en una fecha determinada, de acuerdo a la configuración indicada en el fichero `/etc/crontab`. El proceso `launchd` ha asumido ahora estas responsabilidades.

`launchd` ofrece principalmente tres ventajas respecto al mecanismo tradicional de arranque de Mac OS X. La primera ventaja es que con `launchd` se centraliza y homogeniza toda la gestión de los procesos de background. También `launchd` se encarga de gestionar los posibles límites de recursos que asignamos a los programas, y de decidir con qué cuenta de usuario se ejecuta cada uno.

La segunda ventaja es que la configuración de todos los procesos de background `launchd` compatibles se hace en un fichero XML con la extensión `.plist`. Este fichero podemos editarlo con la herramienta Property List Editor<sup>1</sup>. Hasta ahora, cada comando tenía su propio fichero de configuración, y cada uno usaba su propio formato.

La tercera ventaja es que los procesos de background `launchd` compatibles se pueden lanzar bajo demanda, lo cual permite no tener cargados en memoria procesos de background que no se están usando. Además `launchd` puede descargar de memoria un proceso de background si éste lleva mucho tiempo inactivo y se está necesitando memoria.

---

<sup>1</sup> Este programa se distribuye junto con las Developers Tools, y podemos encontrarlo en la carpeta `/Developer/Applications/Utilities/`

## El proceso de arranque

---

El proceso de arranque de una máquina Mac OS X se puede resumir en cuatro pasos:

El primer paso es el *arranque de la BIOS*, durante este proceso se comprueba el hardware disponible y se elige el sistema operativo a ejecutar. Este primer paso se puede dividir en:

- POST (Power-On Self Test) donde se determinan el hardware disponible y se comprueba que exista suficiente memoria, así como que el hardware está en buen estado.
- Open Firmware construye un árbol de dispositivos hardware existentes (una representación jerárquica del ordenador), y elige el sistema operativo a ejecutar.

El segundo paso sería la *carga del núcleo* del sistema operativo. Se carga una imagen del núcleo en memoria. Durante este proceso la máquina suele mostrar el icono con la manzana de Apple. Al acabar la carga del núcleo se lanza el proceso de usuario root, que es `launchd`.

Durante el tercer paso `launchd` lanza los procesos necesarios para configurar el sistema. Para ello `launchd` lanza los procesos indicados en el script `/etc/rc` (que pueden, o no, ser de background), así como otros procesos que comentaremos más adelante. Esta es la parte que ha sido optimizada con el uso de `launchd`.

En el cuarto paso `launchd` lanza `loginwindow` que es el proceso encargado de autenticar a los usuarios y controlar su sesión.

# Configurar launchd

---

## Demonios y agentes

En el mundo UNIX tradicionalmente se ha usado el término **demonio** (daemon) para referirse a procesos que se desprenden tanto de su padre como de la entrada y salida de la consulta. Para ello, típicamente, el proceso ejecuta la llamada al sistema `daemon()`. Estos procesos son procesos que no mueren cuando muere el proceso padre que los creó, pero a cambio no tienen acceso a la consola. En el mundo UNIX los demonios representan procesos de background.

Apple utiliza el término **proceso de background** para referirse a cualquier proceso que ejecute en background. Y los subdivide en dos tipos:

1. Los **demonios**, que son procesos que ejecutan en background a nivel de sistema, es decir, que no dependen de un usuario. Por el propio modelo de seguridad de Mac OS X, los demonios no tienen acceso ni a la consola ni al window server (no pueden mostrar ventanas)<sup>2</sup>. Un servidor web es un buen ejemplo de demonio.
2. Los **agentes**, que son procesos que ejecutan en background, pero que dependen de un usuario, y que sí que tienen acceso al window server. Un calendario que muestra avisos cuando llega una determinada fecha sería un buen ejemplo de agente.

## Los demonios

Los demonios `launchd` deben de colocarse como ficheros `.plist` bajo el directorio `/Library/LaunchDaemons`.

También existen otros dos directorios que Apple usa para configurar demonios, pero que no debemos de usarlos, ya que están reservados para los demonios de Apple:

- `/etc/mach_init.d` Reservado por Apple para lanzar demonios que se debían lanzar durante el arranque del sistema. El directorio estuvo en vigor hasta Mac OS X 10.3, y actualmente está deprecado, aunque Apple lo sigue usando.
- `/System/Library/LaunchDaemons` Éste es el directorio que actualmente usa Apple para colocar sus demonios. Este directorio no debemos de usarlo, ya que está reservado para los demonios de Apple.

---

<sup>2</sup> Este comportamiento es independiente de si el proceso ejecuta, o no, a la llamada al sistema `daemon()`. Aun así `daemon()` sigue siendo útil cuando un proceso que lanzamos desde el terminal quiere desprenderse de su padre y de la consola del terminal.

Los demonios pueden ser de dos tipos:

- `RunAtLoad`, los cuales se ejecutan una vez al arrancar la máquina, y permanecen cargados durante toda la ejecución.
- `OnDemand`, los cuales se ejecutan sólo cuando algún cliente intenta usar el servicio.

En principio, es mejor que los procesos se carguen `OnDemand`, pero esto no siempre es posible, ya que para que un proceso se cargue de esta forma tiene que haber seguido una serie de recomendaciones que da Apple a los desarrolladores, como son el capturar la señal `SIGTERM` para terminarlo, o el que el proceso demonio no llame a la primitiva `daemon()` para desligarse del proceso padre que lo creó (lo cual haría creer a `launchd` que el proceso ha fallado y automáticamente volvería a lanzarlo). Si está intentando añadir un proceso ya existente a la lista de demonios de su máquina, use la opción `RunAtLoad`, a no ser que en la documentación del programa indique que es `launchd` compatible.

## Los agentes

Los agentes se ejecutan después que los demonios cada vez que nos logamos con un usuario, y se cierran al abandonar la cuenta de usuario.

Los agentes `launchd` deben de colocarse como ficheros `.plist` bajo el directorio `/Library/LaunchAgents`.

También existen otros dos directorios que Apple usa para configurar agentes, pero que no debemos de usarlos, ya que están reservados para los agentes de Apple:

- `/etc/mach_init_per_user.d` Reservado por Apple para lanzar agentes que se debían de lanzar una vez por cada usuario. El directorio estuvo en vigor hasta Mac OS X 10.3, y actualmente está deprecado, aunque Apple lo sigue usando.
- `Library/LaunchAgents` Es el directorio que actualmente usa Apple para colocar sus agentes. Este directorio no debemos de usarlo, ya que está reservado para los agentes de Apple.

## Crear el fichero de propiedades de un demonio o agente

Como hemos dicho, para crear un demonio o agente debe de crear un fichero `.plist` en la carpeta `/Library/LaunchDaemons`, o `/Library/LaunchAgents` respectivamente, indicando a `launchd` cómo lanzar el demonio o agente. Este fichero debe de contener una serie de propiedades que se resumen en la Tabla 1. Puede consultar una documentación más detallada ejecutando `man launchd.plist`.

Clave	Tipo	Req.	Descripción
Label	string	Sí	Cadena única que identifica al job de <code>launchd</code> .
Disabled	boolean	No	Permite deshabilitar un job. Por defecto es <code>false</code> .
UserName	string	No	Nombre de usuario con el que ejecutar el job. Por defecto es <code>root</code> .
GroupName	string	No	Grupo con el que ejecutar el job. Por defecto es <code>wheel</code> .
inetdCompatibility	dictionary	No	La presencia de esta clave indica que el proceso espera lanzarse como si estuviera configurado para <code>inetd</code> .
ProgramArguments	array of strings	Sí	Argumentos de línea de comandos del proceso de background.
Program	string	No	Indica el comando a ejecutar. Si se omite se usa el primer argumento de <code>ProgramArguments</code> .
OnDemand	boolean	No	Indica si el proceso se lanza bajo demanda. Por defecto es <code>true</code> .
RunAtLoad	boolean	No	Indica si el proceso se lanza siempre al arrancar. Por defecto es <code>false</code> .
RootDirectory	string	No	Indica el directorio al que hacer <code>chroot</code> antes de ejecutar el proceso.
WorkingDirectory	string	No	Indica el directorio al que

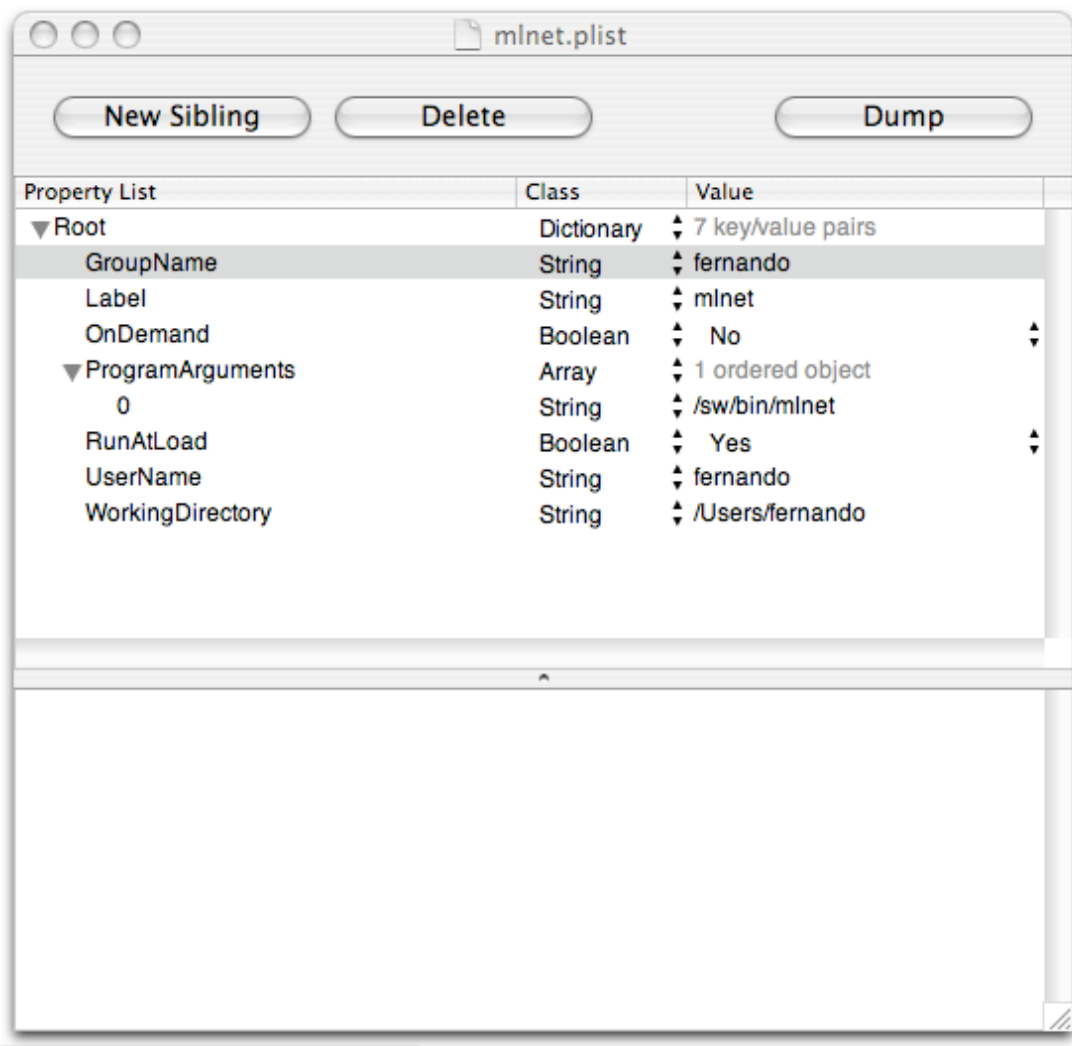
			hacer <code>chdir</code> antes de ejecutar el proceso.
<code>ServiceDescription</code>	<code>string</code>	No	Permite dar una descripción del proceso
<code>EnvironmentVariables</code>	<code>dictionary of strings</code>	No	Variables de entorno a fijar antes de ejecutar el proceso
<code>Umask</code>	<code>integer</code>	No	Indica que debe de pasarse a <code>umask</code> antes de ejecutar el proceso.
<code>ServiceIPC</code>	<code>boolean</code>	No	Indica si el proceso se comunica con <code>launchd</code> usando técnicas de IPC. El flag es incompatible con <code>inetdCompatibility</code> .
<code>TimeOut</code>	<code>integer</code>	No	El tiempo máximo que tardará el job en arrancar.
<code>StartInterval</code>	<code>integer</code>	No	Indica cada cuanto tiempo debe de lanzarse el job. Usado para jobs periódicos.
<code>StartCalendarInterval</code>	<code>dictionary of integer</code>	No	Permite indicar fechas periódicas a las que lanzar el programa. Vea <code>man launchd.plist</code>
<code>StandardOutPath</code>	<code>string</code>	No	Fichero a usar como <code>stdout</code> .
<code>StandardErrorPath</code>	<code>string</code>	No	Fichero a usar como <code>stderr</code> .
<code>SoftResourceLimits</code> <code>HardResourceLimits</code>	<code>dictionary of integers</code>	No	Indica límites a los recursos del proceso como se indica en <code>man launchd.plist</code> .
<code>Nice</code>	<code>integer</code>	No	Indica un valor a pasar a <code>nice</code> para ejecutar el comando con menos prioridad.
<code>Sockets</code>	<code>dictionary of strings</code>	No	Permite indicar qué sockets lanzan este proceso al llegar un cliente como se indica en <code>man launchd.plist</code> .

**Tabla 1:** Propiedades del fichero `.plist` de `launchd`

## Ejemplo de demonio launchd

En la carpeta `/System/Library/LaunchDaemons` tiene varios demonios de ejemplo, pero aquí vamos a ver cómo se crearía un demonio para `mlnet`, la conocida herramienta de intercambio de ficheros<sup>3</sup>.

La Figura 1 muestra el contenido del fichero `.plist` propuesto. La herramienta se lanza usando el comando `mlnet`. Una vez ejecutado este comando, en el directorio actual se crea una serie de carpetas con la configuración del programa, así como con las carpetas donde depositar los ficheros a intercambiar. El comando se queda esperando a que nos conectemos a él: O bien haciendo un telnet al puerto 4000, o bien conectando vía web al puerto 4080.



**Figura 1:** Fichero de configuración de `mlnet`

<sup>3</sup> Actualmente este fichero puede encontrarse en Internet buscando el nombre `mlnet`.

Este es un buen ejemplo de comando que podemos tener ejecutando desde que arrancamos la máquina hasta que la paramos. Luego usaremos la entrada `RunAtLoad`.

Para indicar dónde está el directorio con los ficheros compartidos podemos usar la entrada `WorkingDirectory`, y si queremos dar un usuario con el que lanzar el comando podemos usar la entrada `UserName` y `GroupName`.

El fichero `mlnet.plist` debe de tener permisos de sólo lectura para los usuarios que no sean administradores, con el fin de evitar que un usuario sin permiso de administración modifique estos ficheros. Apple pide que tengan permiso de lectura para el grupo `wheel`, con el fin de poder ser lanzados. El usuario que tengan como dueño debe tener permiso de lectura y escritura (p.e. en mi caso el dueño es el usuario `fernando`). Los ficheros del sistema (los de la carpeta `/System/Library/LaunchDaemons`) tienen como dueño a `system`, que es el dueño de todos los ficheros bajo la carpeta `/System`.

## Testear un demonio

Antes de poner el demonio en la carpeta `/Library/LaunchDaemons` y reiniciar su máquina puede probar a ver si arranca correctamente. Para ello puede usar el comando `launchctl` y pedirle que cargue el demonio.

```
$ launchctl  
launchd% load /Library/LaunchDaemons/mlnet.plist
```

Por la salida que produzca este programa al lanzarle podrá saber si algo está mal. Si todo va bien ya puede colocar el fichero de arranque en el directorio `/Library/LaunchDaemons` y reiniciar su máquina.

## Usar StartupItems

---

Antes de Mac OS X 10.4 ya existían los StartupItems, los cuales nos permitían indicar demonios que queríamos ejecutar al arrancar la máquina. Los StartupItems sólo nos permiten lanzar demonios al arrancar la máquina (no bajo demanda como `launchd`), pero a cambio tienen un sistema de gestión de dependencias entre demonios que nos permiten indicar que un demonio no se debe lanzar hasta que se haya lanzado otro. Actualmente los StartupItems se han conservado, y es la forma que está usando Apple para gestionar estas dependencias entre sus propios demonios. En caso de que no existan dependencias entre demonios se recomienda usar `launchd`, pero la decisión final queda en sus manos.

### Anatomía de los StartupItems

Los StartupItems se guardan en la carpeta `/Library/StartupItems`, o en el caso del sistema operativo, éste guarda los StartupItems en la carpeta `/System/Library/StartupItems`. Por cada elemento a lanzar debe crearse una subcarpeta con el nombre del elemento, y dentro de él debe aparecer un script con el mismo nombre que la carpeta, y que veremos que es el encargado de lanzar y parar el demonio. Además dentro de la subcarpeta debe de aparecer un fichero con el nombre `StartupParameters.plist` que contiene información de configuración del demonio.

Al igual que pasaba con los elementos de `launchd`, estos ficheros deben de tener permiso de sólo lectura para cualquier usuario que no sea administrador, y deben de pertenecer al grupo `wheel` con permiso de lectura para este grupo, con el fin de que el sistema los pueda ejecutar.

Por ejemplo, si queremos ahora lanzar `m1net` usando un StartupItem y no `launchd`, podemos crear la subcarpeta `/Library/StartupItems/m1donkey`, y crear dentro de ella dos ficheros: Un script con permiso de ejecución con el nombre `m1donkey`, y el fichero de parámetros `StartupParameters.plist`. A continuación detallaremos cómo se haría esto.

### Crear el script de arranque / parada

El script de arranque debería de tener la forma que muestra el Listado 1. El script empieza cargando una serie de rutinas comunes localizadas en el fichero `/etc/rc.common`. Después debemos de implementar las funciones

`StartService()`, `StopService()` y `RestartService()` que lanzan, paran y reinician el demonio.

```
#!/bin/sh
. /etc/rc.common

StartService ()
{
    ....
    ....
}

StopService ()
{
    ....
    ....
}

RestartService ()
{
    ....
    ....
}

RunService "$1"
```

**Listado 1:** Esqueleto del script de arranque

Dentro del fichero `/etc/common` se encuentra la función `RunService()` que se encarga de llamar a una de las anteriores funciones según proceda. Si el código de arranque de un demonio lleva mucho tiempo se recomienda lanzar el demonio en background con el fin de que el arranque del sistema operativo entero no se vea penalizado.

El Listado 2 muestra un script (llamado `mldonkey`) que podríamos hacer para lanzar `mlnet` usando los `StartupItems`.

```
#!/bin/sh
. /etc/rc.common

StartService ()
{
    cd /Users/fernando/donkey/
    echo $PWD
    su fernando -c 'nohup /sw/bin/mlnet &'
}

StopService ()
{
    killall mlnet
}
```

```

}

RestartService ()
{
  StopService
  StartService
}

RunService "$1"

```

**Listado 2:** StartupItem que lanza y para `mlnet`

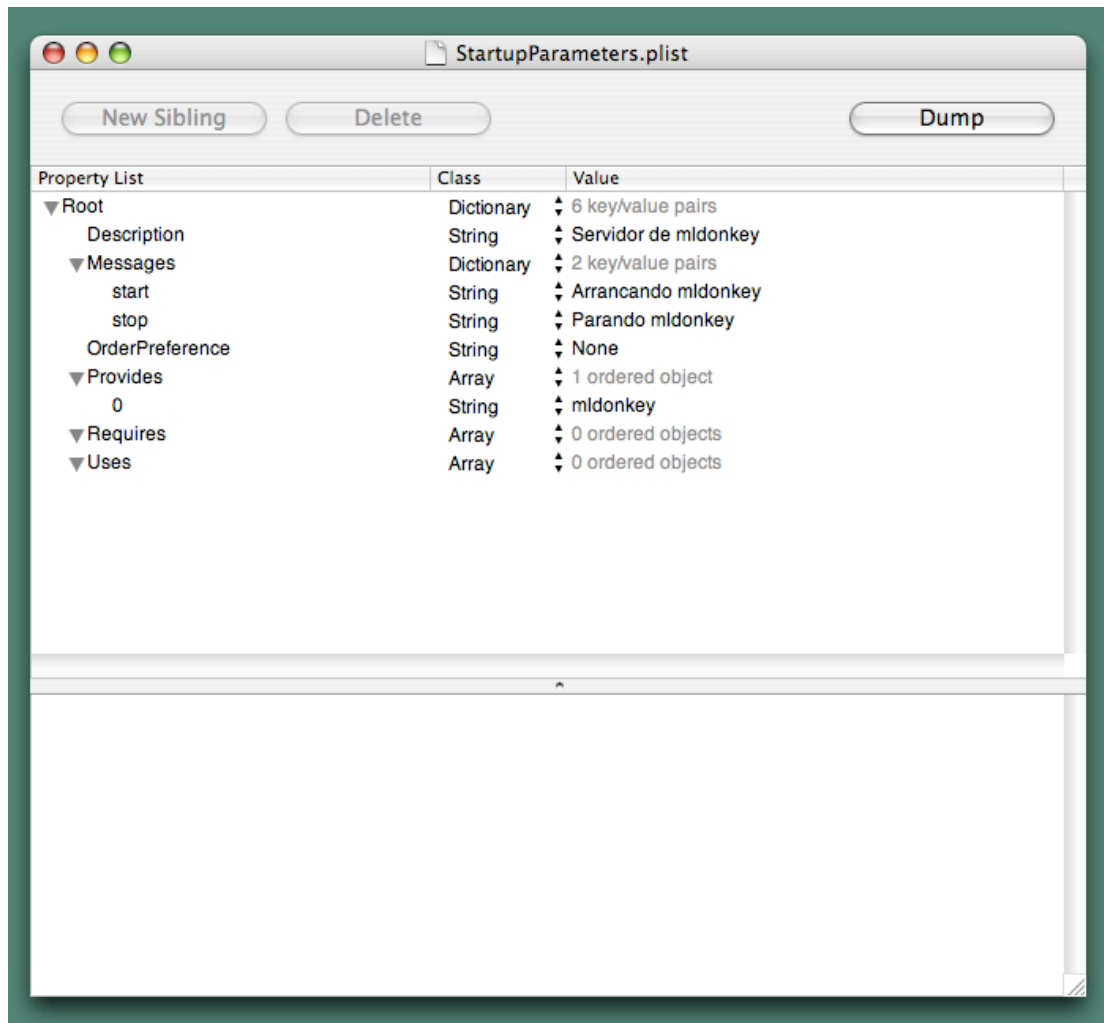
## Especificar los parámetros del StartupItem

Los parámetros del StartupItem se indican en un fichero con el nombre `StartupParameters.plist`, el cual no es un fichero de propiedades XML (como en el caso de `launchd`), sino un fichero de propiedades ASCII, que también podemos crear con la herramienta Property List Editor, pero al guardarlo debemos de decirle que lo guarde como un fichero de propiedades ASCII.

La Tabla 2 muestra los pares clave-valor que puede contener este fichero, y la Figura 2 muestra el contenido del fichero `StatupParameters.plist` para el ejemplo de lanzar `mlnet`.

Clave	Tipo	Valor
Description	string	Una descripción corta del elemento.
Provides	array	Nombre de los servicios que proporciona este elemento. Aunque un demonio puede proporcionar muchos servicios se recomienda indicar sólo uno.
Requires	array	Nombre los servicios que tienen que estar ya ejecutándose para poder lanzar este servicio. Si alguno de los servicios no está disponible, nuestro servicio no se ejecuta.
Uses	array	Nombre los servicios que deberían de estar ya ejecutándose para poder lanzar este servicio. Si alguno de los servicios no está disponible, nuestro servicio debería de poder ejecutar igualmente.
Messages	dictionary	El diccionario contendrá dos claves llamadas <code>start</code> y <code>stop</code> , cuyo valor será el mensaje que queremos dar al lanzar o parar el demonio. Este mensaje lo muestra Mac OS X al arrancar.

**Tabla 2:** Posibles clave-valor del fichero `StartupParameters.plist`



**Figura 2:** Fichero StartupParameters.plist para mldonkey

## Lanzar y parar un StartupItem desde la consola

Podemos lanzar StartupItems desde el terminal usando el comando:

```
$ /sbin/SystemStarter start mldonkey
```

Obsérvese que usamos como nombre `mldonkey` que es el valor que aparece en el campo `Provides` del fichero de parámetros, y no el nombre del comando que es `mldonkey`.

También podemos usar la opción `stop` o `restart` para ejecutar estas operaciones. Realmente este comando lo que hace es ejecutar la función `RunService()` del fichero de script correspondiente pasándole como argumento `start`, `stop` o `restart`.

Este comando también se puede usar para depurar un StartupItem si éste no está funcionando bien. Para ello usamos la opción `-d` de la forma:

```
$ /sbin/SystemStarter -d start mldonkey
```

Ahora obtendrá mucha información de depuración indicando que es lo que está sucediendo.

## Configurar el login y logout

---

Para lanzar una aplicación cada vez que el usuario hace un login en su cuenta se puede usar los **login items**, que son programas que `loginwindow` ejecuta cada vez que el usuario entra en su cuenta. A diferencia de los agentes, no se ejecutan cuando nos logamos con una cuenta sin GUI. Estos elementos se pueden configurar, o bien desde el panel de control en una de las opciones de cuenta de usuario, o bien programáticamente desde el fichero de propiedades `~/Library/Preferences/loginwindow.plist`.

Los login items son la forma recomendada de lanzar aplicaciones cuando el usuario entra en su cuenta, pero existen otras técnicas "más oscuras" que vamos a comentar también.

### Instalar un script usando defaults

Desde Mac OS X 10.3, si lo que queremos es lanzar un script podemos ponerlo en defaults de `loginwindow` así:

```
$ sudo defaults write com.apple.loginwindow LoginHook  
/path/al/script
```

Donde `/path/al/script` es el path al script que queremos lanzar al logarnos.

Cuando se lanzan estos script, se lanzan con la cuenta de root. Normalmente es recomendable reducir su peligrosidad pasando desde dentro del script a la cuenta del usuario. Para ello podemos crear un script como muestra el Listado 3.

```
#!/bin/bash  
  
su $1  
# Comandos menos peligrosos  
.....  
.....
```

**Listado 3:** Script para login item

El script siempre recibe como primer argumento el nombre del usuario que se está logando.

También hay que tener en cuenta que el usuario no tendrá acceso a su cuenta hasta que el script termine, con lo que si el proceso es largo, es mejor lanzarlo en background.

## Pasar opciones a loginwindow

En la carpeta `/System/Library/CoreServices/` encontramos un conjunto de aplicaciones del sistema, entre ellas `loginwindow` la cual es lanzada por `launchd` cuando termina de lanzar los demonios. En concreto, desde Mac OS X 10.2 en el fichero `/etc/ttys` encontramos la entrada:

```
console "/System/Library/CoreServices/loginwindow.app/
Contents/MacOS/loginwindow" vt100 on secure
onoption="/usr/libexec/getty std.9600"
```

La cual indica a `launchd` con que argumentos debe de lanzar la consola.

La Tabla 3 muestra las opciones que podemos pasar a `loginwindow` (precedidas por guión) para personalizar su comportamiento.

Opción	Descripción
<code>-LoginHook</code>	Permite indicar el path de un script que queremos ejecutar al lanzar <code>loginwindow</code>
<code>-LogoutHook</code>	Permite indicar el path de un script que queremos ejecutar al acabar <code>loginwindow</code>
<code>-PowerOffDisabled</code>	Si recibe un <code>YES</code> las opciones de menú de restart y shutdown estarán deshabilitadas.

**Tabla 3:** Opciones de `loginwindow`

Por ejemplo, podemos modificar esta entrada así:

```
console "/System/Library/CoreServices/loginwindow.app
/Contents/MacOS/loginwindow" -LoginHook /path/al/script -
PowerOffDisabled YES vt100 on secure
onoption="/usr/libexec/getty std.9600"
```

Con lo que se ejecutará el script `/path/al/script` al logarnos en una cuenta, y estarán deshabilitadas las opciones de restart y shutdown.

Obsérvese que este comportamiento no es muy recomendable ya que estamos personalizando opciones para todas las cuentas. Apple recomienda usar `defaults` o los login item, en vez de esta última forma de personalizar `loginwindow`.