UNIVERSIDADE DA CORUÑA

TRABAJO TUTELADO

ADMINISTRACIÓN DE SISTEMAS OPERATIVOS

Estudiantes: Cisneros Sande, Daniela.

Freire Blanco, Hugo.

Romero Zas, Iván.

A Coruña, abril de 2024.

INDICE

¿Qué es Systemd?

Es un conjunto de demonios (daemons), bibliotecas, y herramientas diseñadas para poder administrar y configurar el núcleo del sistema operativo. Es un elemento básico para el sistema operativo ya que es el sistema de inicio básico, con un primer proceso del que se generan todos los demás como una cadena, es decir, arranca todo lo que esta debajo del kernel permitiendo de esta forma ejecutar varios programas de forma simultánea.

Otra de las funcionalidades que ofrece systemd es la capacidad de implementar un sistema de gestión de dependencias que se basa en un control de los servicios, lo que le permite controlar las dependencias de forma más eficiente, optimizando el rendimiento del sistema operativo y mejorando su administración.

Además, systemd también permite el inicio de demonios bajo demanda, herramientas para la configuración del sistema y capacidad para realizar un seguimiento de los procesos del sistema con el uso de los **grupos de control** (hablamos más delante de esto en las funcionalidades).

El nombre de systemd es para poder distinguir los daemons fácilmente por tener la letra d como última letra del nombre (system daemons).

Evolución histórica

A principios de la década de los 2000, a medida que aparecían nuevas incorporaciones a la familia Linux empezó a manifestarse la necesidad de actualizar el sistema de inicio, Init, perteneciente a un System V que acumulaba más de 20 años de existencia para aquel entonces. La principal razón detrás de esta drástica medida fue la creciente necesidad de tiempos de arranque de servicios más rápidos, los cuales Init no lograba alcanzar, y por lo tanto distintas entidades y usuarios comenzaron a ofrecer sus puntos de vista para el nuevo inicializador de servicios.

Los principales contendientes a crear el sucesor definitivo a Init fueron por un lado Canonical, responsables de Ubuntu, con su Upstart (2006) y por otro Red Hat con Lennart Poettering, creador de avahi y pulseaudio, con su vision de Init, SystemD (2010). El debate sobre quien reemplazaria al iniciador agito foros de la comunidad de usuarios de Linux durante muchos meses, causando serias discrepancias además entre figuras y empresas relevantes. Finalmente la decantación por parte del Comité Técnico de Debian en febrero de 2014 a favor de adoptar definitivamente SystemD para el operativo Debian supuso el golpe final para el proyecto Upstart, además de una victoria para Red Hat y Poettering.

En el año siguiente, 2015, marco la fecha de consolidación de SystemD como el nuevo estándar para todos los sistemas operativos populares de Linux.

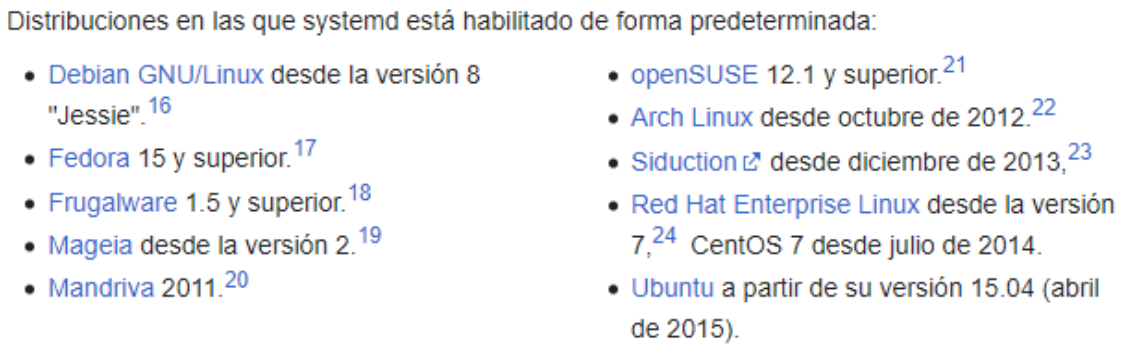


Figura1: Implementación de systemd en las principales distribuciones

Antes de explicar las diferentes funcionalidades que tiene systemd es preciso entender cuáles son los elementos con los que interactúa ya que nos ayudara a comprender mejor su funcionamiento. En los siguientes apartados explicaremos los diferentes elementos.

¿Qué son las units?

Las units de systemd son bloques básicos de configuración que representan recursos administrados por systemd, como servicios, sockets, dispositivos, puntos de montaje, entre otros. Cada unit contiene información sobre cómo systemd debe manejar y controlar ese recurso específico durante el arranque y el funcionamiento del sistema, esta información se encuentra en sus ficheros de configuración también llamados unit files.

Algunas características de las units son:

- **Activación basada en sockets:** Los servicios pueden esperar a ser iniciados hasta que se necesite acceder al socket asociado, permitiendo arrancar los servicios en paralelo al inicio del sistema.
- **Activación basada en bus:** Las units pueden empezar cuando se publique un bus asociado en D-Bus.
- **Activación basada en dispositivo:** Las units pueden arrancar cuando se conecte el hardware asociado.
- **Mapeo implícito de dependencias:** Systemd construye automáticamente la mayoría de las relaciones de dependencia, facilitando la configuración.
- **Instancias y plantillas:** Se pueden crear múltiples versiones de una unit básica para adaptarse a diferentes necesidades o funciones.
- **Facilidad para endurecer la seguridad:** Se pueden agregar medidas de seguridad fácilmente a las unit, como restringir el acceso a partes del sistema de archivos.
- **Drop-ins y fragmentos:** Es fácil personalizar units al anular partes específicas de los archivos de configuración estándar.

Localización de los ficheros de las units

Los ficheros asociados a las units de systemd generalmente están en el directorio /lib/systemd/system, siendo esta la localización por defecto cuando se instalan nuevas units en el sistema.

En caso de que queramos modificar cómo funciona una de las units podemos hacerlo en el directorio /etc/systemd/system. Los ficheros de las units que están en este directorio tienen preferencia sobre el resto de las localizaciones de nuestro sistema de archivos por lo que en caso de que queramos modificar una de las units podemos copiar su archivo asociado a este directorio y comenzar a modificarlo ya que va a tener preferencia sobre el original.

Por otro lado, si queremos modificar una parte del fichero o extender la funcionalidad, la forma correcta de hacer esto es creando un directorio terminado en .d como una extensión del fichero de unidad original y dentro de este directorio podemos crear los archivos de configuración (.conf) para poder sobrescribir o extender uno de los ficheros.

Ejemplo:

Ponemos un ejemplo para clarificar lo anteriormente explicado, lo que vamos a hacer es extender la funcionalidad de un servicio (miServicio.service) que ya tenemos creado:

Tenemos la unit concreta que queremos modificar en el directorio /lib/systemd/system.

Aplicando esta técnica vamos a crear un directorio que tenga como nombre el nombre del servicio, pero terminado en .d

$ mkdir /etc/systemd/system/miServicio.service.d

Una vez que estamos dentro de este directorio vamos a crear el archivo de configuración con el que vamos a extender la funcionalidad del servicio que teníamos inicialmente.

$ nano /etc/systemd/system/miServicio.service.d/custom.conf

Ahora vamos a añadir una variable de entorno:

[Service]

Environment="Mi_variable=Mi_valor_variable"

A continuación, guardamos el fichero y recargamos el servicio para añadir la nueva configuración al servicio:

$ systemctl restart miServicio

Con estos pasos hemos extendido la funcionalidad del servicio sin modificar directamente el archivo de configuración por lo que es una forma más segura de aplicar nuevos cambios ya que evitamos errores en el fichero original.

Tipos de units

Hay **diferentes tipos de categorías o units** de systemd en función del recurso que describen. Podemos distinguir el tipo de unit que es mediante el sufijo que lleva al final del archivo de esa unidad. Los tipos principales de unidades son los siguientes:

- **.service**: Una unit de servicio describe cómo manejar un servicio o aplicación, proporcionando información sobre cómo gestionarlo, como por ejemplo como arrancar y parar el servicio, en qué circunstancias tiene que ser parado, entre otras muchas funciones.

Figura2: Salida del comando para listar los servicios del sistema

- **.socket**: Una unit de tipo socket representa un punto de conexión para la comunicación de procesos tanto del sistema como a través de la red. Los archivos .socket siempre están asociados a una unit .service que se inicia cuando hay actividad en el socket definido por ese archivo de unit.

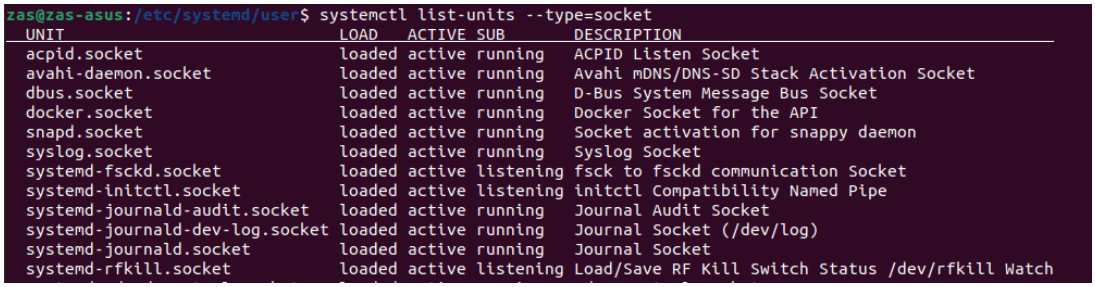


Figura3: Salida del comando para listar los sockets del sistema

- **.device**: Es una unit que describe un dispositivo que ha sido designado a systemd quien se encargara de su manejo a través por ejemplo de udev. No todos los dispositivos del sistema van a tener ficheros .device asociados.

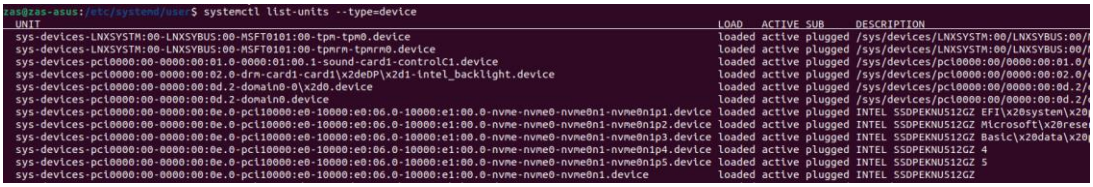


Figura4: Salida del comando para listar los devices del sistema

- **.mount**: Esta unit define un punto de montaje en el sistema que es gestionado por systemd. Cuando systemd detecta entradas en /etc/fstab (para el montaje

automático de directorios), puede generar automáticamente units de tipo .mount, lo que simplifica la gestión del sistema de archivos y el montaje de dispositivos, además de garantizar un comportamiento consistente durante el arranque.

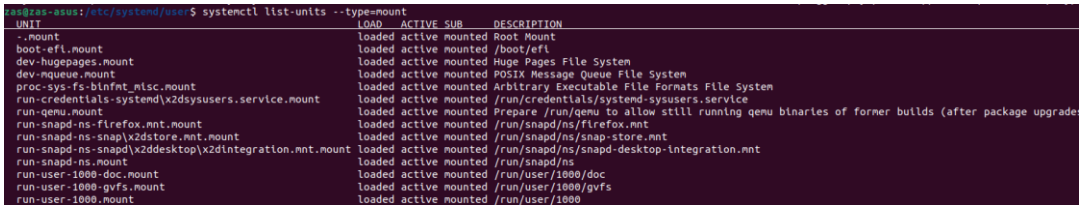


Figura5: Salida del comando para listar las unidades de tipo mount

- **.swap**: Estas units describen el espacio de swap en el sistema. El nombre de estas units tiene que referirse o reflejar ese espacio.

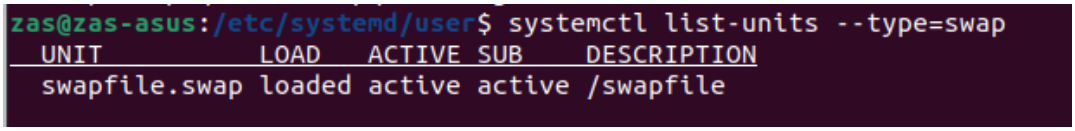


Figura6: Salida del comando para listar las unidades de tipo swap

- **.target**: Los targets representan los diferentes estados a los que el sistema puede llegar. Una vez que completemos uno de los targets el sistema pasa a un nuevo estado. También se consideran puntos de sincronización para otras units durante el proceso de arranque. Estos puntos de sincronización son importantes para poder asegurarse de que las units se activen en el lugar correcto y en el momento adecuando durante el arranque del sistema.

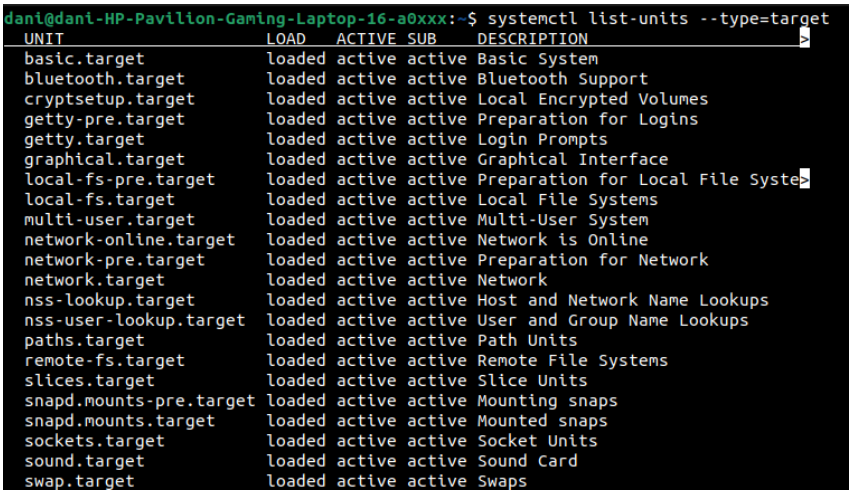


Figura7: Salida del comando para listar las unidades de tipo swap

- **.timer**: Define un temporizador, manejado por systemd, similar a cron para la activación y ejecución programada de comandos y scripts. Esto facilita la ejecución de tares programadas de forma eficiente dentro de systemd.

```
zas@zas-asus:/etc/systemd/user$ systemctl list-units --type=timer
  UNIT                          LOAD   ACTIVE SUB     DESCRIPTION
  anacron.timer                 loaded active waiting Trigger anacron every hour
  apt-daily-upgrade.timer       loaded active waiting Daily apt upgrade and clean activities
  apt-daily.timer               loaded active waiting Daily apt download activities
  dpkg-db-backup.timer          loaded active waiting Daily dpkg database backup timer
  e2scrub_all.timer             loaded active waiting Periodic ext4 Online Metadata Check for All Filesystems
  fstrim.timer                  loaded active waiting Discard unused blocks once a week
  fwupd-refresh.timer           loaded active waiting Refresh fwupd metadata regularly
  logrotate.timer               loaded active waiting Daily rotation of log files
  lynis.timer                   loaded active waiting Daily timer for the Lynis security audit and vulnerability scanner
  man-db.timer                  loaded active waiting Daily man-db regeneration
  motd-news.timer               loaded active waiting Message of the Day
  systemd-tmpfiles-clean.timer  loaded active waiting Daily Cleanup of Temporary Directories
  update-notifier-download.timer loaded active waiting Download data for packages that failed at package install time
  update-notifier-motd.timer    loaded active waiting Check to see whether there is a new version of Ubuntu available
```

Figura8: Salida del comando para mostar los timers

Formato de un fichero unit

La estructura interna de un fichero de unidad está **organizada en secciones** que son delimitadas con unos corchetes [ ] con el nombre de la sección dentro de ellos.

El nombre de las secciones es case-sensitive, lo que significa que, si no se respetan las mayúsculas y minúsculas, no se interpretará correctamente. Por tanto, la primera letra debe ser en mayúscula y el resto en minúscula.

Cada una de las secciones está formada por directivas que son elementos de tipo campo=valor.

```
[ Section ]
Directive1 = value
Directive2 = value
```

Secciones de los ficheros unit

-Unit: Se usa para definir metadatos de la unit y establecer la relación de esta con otras units. Esta sección se suele poner en la parte de arriba del fichero porque proporciona una visión general de la información de la unit.

Principales directivas que podemos poner dentro de la sección unit:

- **Description:** Descripción de la funcionalidad básica de la unit. Formada por un texto corto e informativo.
- **Documentation:** Esta directiva proporciona una localización para las listas de uris/url para la documentación de la unit. Puede ser la página del manual que podemos consultar o una url con la información. Esta información se va a exponer por ejemplo cuando hacemos el comando systemctl status “servicio”

- **Requires:** Este parámetro, lista las units de las que depende. Para que se active, las otras units de las que depende también tienen que estar activadas sino la activación falla.

- **Wants:** Es similar a requires pero menos restrictiva. Systemd intenta arrancar estas units que tenemos definidas, pero si alguna de estas falla, la unit actual continua su ejecución de forma normal no falla como en el caso de requires.
- **Before:** Las units que se listen aqui no se iniciarán hasta que esta unit haya sido marcada como iniciada, indicando así su prioridad de arranque.
- **After:** Las units que se listen aqui se iniciarán antes de que esta unit sea marcada como iniciada, lo que especifica su secuencia de arranque. Esto implica que todas las units de la lista deben haber iniciado antes de que esta empiece.
- **Conflicts:** Aquí indicaremos las units que no pueden empezar al mismo tiempo que la unit actual. Al comenzar esta unit puede hacer que otras units se paren para no empezar a la vez y generar conflictos.

-**Install:** Es opcional y se usa para definir el comportamiento de la unit en caso de que esté activada o desactivada. Solo las units que se puedan habilitar van a tener definida esta sección.

Consta de las siguientes directivas:

- **Wanted by:** Es la forma más común para indicar como la unit debería ser habilitada y para ello nos permite especificar una dependencia de forma similar al wants. Cuando usamos esta directiva en una unit systemd crea un enlace simbólico en el directorio /etc/systemd/system que indica la relación de dependencia. Por ejemplo, si tenemos wantedby=multi-user.target se va a crear un enlace simbólico en /etc/systemd/system/multiuser.target.wants que apunta a esta unidad.
- **Required by:** Similar a 'WantedBy', pero establece una dependencia de tipo 'required'. Esto significa que para que esta unit funcione correctamente, la unit a la que hace referencia en 'required' debe estar activa. Se crea un enlace en el directorio '/etc/systemd/system' con extensión '.requires'.
- **Alias:** Permite especificar un alias a la unit de forma que podemos interactuar con ella (activarla, pararla, ...) usando ese alias como nombre. (De esta forma tenemos un nombre alternativo para la unit).

-**Service:** Se usa para proporcionar una configuración que solo es aplicable para las units de tipo .service:

Los principales parámetros son:

- **Type:** tipo del servicio, esto es muy importante para poder categorizar/separar los diferentes servicios en función de sus procesos y su comportamiento. Esta información le dice a systemd como manejar estos servicios y como buscar su estado. Hay varios valores posibles en el campo type:

    1. **Simple:** Valor que toma por defecto si no se le indica otra cosa. Hace referencia a un servicio que consiste en un solo proceso que se indica en la línea execstart y donde cualquier otro proceso o comunicación tiene que ser gestionada por otra unit fuera de ésta (dentro de la unit solo tenemos ese proceso).

2. **Forking:** Este es el tipo que vamos a tener cuando el servicio crea un proceso hijo mediante fork. Es util para que systemd sepa que el proceso hijo tiene que seguir corriendo, aunque el padre haya terminado.
3. **Notify:** Indica que el servicio va a emitir una notificación cuando haya terminado de arrancar y esté listo para funcionar. Systemd va a esperar a recibir esta notificación para poder continuar con las otras units del sistema.

- **Execstart:** Aquí es donde especificamos la ruta completa y los argumentos que le vamos a pasar al comando que va a ser ejecutado en el service. Si la ruta al comando está precedida por un guión (-), systemd no marcará la unidad como fallida en caso de que el comando ejecutado en 'ExecStart' devuelva un código de error distinto de 0(es decir que no se ejecute exitosamente).
- **ExecStop:** Con esto vamos a indicar el comando necesario para el servicio. Si no le ponemos esto, el proceso es eliminado cuando se pare el servicio.
- **Restart:** Con esto indicamos la situación en la que systemd va a intentar reiniciar automáticamente el servicio. Puede tomar valores como *always, on-success, on-failure, on-abort*, entre otros.
- **Timeout-sec:** Permite configurar el tiempo que systemd tiene que esperar cuando se para o inicia el servicio para considerarlo como fallido o para hacerle un kill directamente y eliminarlo. (Esto puede ser util por si el servicio tarda más de lo esperado en arrancar o no se da acabado de detener).

-Socket: Esta sección es común en las configuraciones de systemd (permitiendo la comunicación entre procesos) porque muchos servicios utilizan los sockets para aumentar la paralelización y la flexibilidad. Algunas directivas que podemos poner son:

- **ListenStream:** Define la dirección del socket para las comunicaciones.
- **ListenDatagram:** Define la dirección del datagram socket que permite entre otras cosas la comunicación rápida de paquetes. Los servicios que usan udp emplean este tipo de socket.
- **ListenFifo:** Le indicamos un buffer de tipo FIFO para la comunicación en lugar de un socket.

-**Mount**: Permite gestionar los puntos de montaje dentro de systemd. Estas unidades de montaje a menudo se traducen directamente desde los archivos /etc/fstab durante el proceso de arranque.

Se pueden especificar:

- **What:** El path del recurso que vamos a montar.
- **Where:** El path donde el recurso va a ser montado.
- **Type:** El tipo de la unidad que vamos a montar.
- **Options:** Otras opciones adicionales.

-Swap: Esta sección la incluiremos en las unidades tipo swap y se usa para poder configurar el espacio de intercambio en el sistema junto con las opciones específicas que le pongamos.

Parámetros:

- **What:** El path de la ubicación del espacio de swap (el path del fichero/device).
- **Priority:** Entero con el que indicamos la prioridad del swap.
- **Options:** Conjunto de opciones adicionales que también podemos indicar en el archivo /etc/fstab.

-**Path**: Permite definir entre otras cosas una ruta de un sistema de archivos que systemd puede monitorizar en busca de cambios. Cuando se detecte algún tipo de actividad en esa ruta systemd va a activar otra unidad que está asociada a esa ruta.

La actividad en esta ruta que hemos puesto se determina mediante evento de tipo inotify que es un mecanismo que tiene el kernel de linux para que las aplicaciones puedan detectar los cambios.

¿Qué son los targets?

Los targets son un conjunto de unidades que se van a ejecutar en unas determinadas circunstancias. Durante el arranque del sistema se ejecutan muchos targets diferentes como poweroff.target, multiuser.target, reboot.target, grafical.target, entre otros.

Estos sistemas heredan muchos aspectos de los niveles de ejecución (runlevels) de SystemV. Aunque en su esencia son similares, es importante destacar la complejidad adicional que introduce systemd con sus targets, un elemento muy importante de esta herramienta.

TABLE 1: SYSTEM V RUNLEVELS AND `systemd` TARGET UNITS

System V runlevel	`systemd` target	Purpose
0	`runlevel0.target`, `halt.target`, `poweroff.target`	System shutdown
1, S	`runlevel1.target`, `rescue.target`,	Single-user mode
2	`runlevel2.target`, `multi-user.target`,	Local multiuser without remote network
3	`runlevel3.target`, `multi-user.target`,	Full multiuser with network
4	`runlevel4.target`	Unused/User-defined
5	`runlevel5.target`, `graphical.target`,	Full multiuser with network and display manager
6	`runlevel6.target`, `reboot.target`,	System reboot

Figura9: Comparación de los runlevels de systemV y los targets de systemd

Representan estados específicos del sistema y para alcanzar esos estados particulares, debemos ejecutar todas las unidades que forman el target. Para averiguar cuáles son estas unidades, podemos utilizar el comando mencionado anteriormente ***systemctl list-dependencies unit.tipoUnit***.

Una de las particularidades de los targets es que no están vinculados a una secuencia de arranque especifica, lo que significa que no siempre se ejecutan en el mismo orden. Además, algunos de ellos pueden ejecutarse de forma paralela, lo que contribuye a acelerar el proceso de arranque del sistema.

La transición entre los targets se puede hacer manualmente por el usuario, con el comando ***systemctl isolate***, o de forma automática durante el arranque del sistema. En este último tendrá que recorrer todos los niveles hasta alcanzar el deseado.

Funcionamiento de los targets en relación con Systemd

Cuando arranca la maquina va a comenzar el primer proceso systemd, que de forma similar a su predecesor corresponde al proceso de pid 1. Este proceso lo que se realiza es la lectura del archivo /etc/systemd/system/default.target para determinar cuál es el target que tiene que lanzar el proceso de inicio, el cual es un enlace simbólico al verdadero target que vamos a lanzar.

De forma general el target default será para la gran mayoría de usuarios el gráfico.

En el caso de un servidor o de una maquina donde no tengamos interfaz gráfica el target apuntado por el default.target es el multiuser.target, que se encarga de todos los servicios corriendo en modo CLI por línea de comandos.

Podemos cambiar el target por defecto del sistema con el comando:

systemctl set-default "nombre target"

Una vez que se ha establecido el target por defecto, comienzan a ejecutarse los demás targets y unidades del sistema. El orden de ejecución está determinado por las dependencias entre estos targets, lo que asegura que se lleven a cabo de manera coordinada y eficiente.

Las dependencias entre los targets vienen definidas por los ficheros de configuración de las units donde para cada unit/target le decimos antes y después de que targets quiere ejecutarse formando una secuencia ordenada de ejecución.

¿Cómo se configuran los targets en Systemd?

Los targets están ubicados dentro del directorio /etc/systemd/system y podemos identificarlos fácilmente porque son archivos que tienen la extensión .target

Podemos consultar la lista completa de los targets disponible con:

systemctl list-units --type target

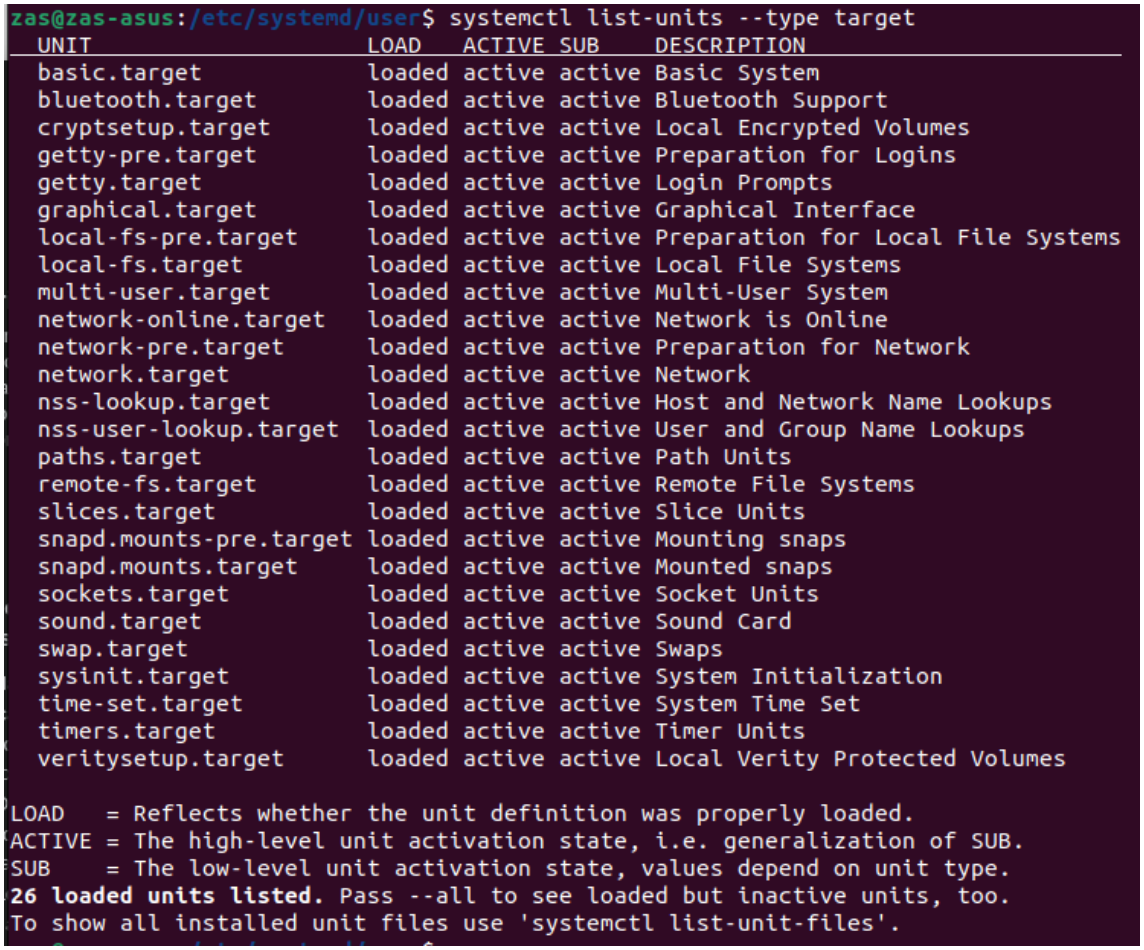


Figura10: Salida del comando para listar los targets

Para crear un target podríamos hacer lo siguiente:

Primero vamos a crear un archivo de configuración para el target en el directorio /etc/systemd/system. Este archivo lo vamos a definir con una extensión .target.

Dentro de este archivo de configuración del target vamos a definir todas las unidades que queremos que se vayan a activar o desactivar cuando se active ese target. Podemos hacer referencia dentro del fichero del target a diferentes tipos de units del sistema como por ejemplo sockets, targets, services, entre otros.

Un posible ejemplo de un fichero de un target puede ser algo así:

```
[Unit]
Description=Mi propio target

[Install]
WantedBy=multi-user.target

[Targets]
# Lista de unidades que deben activarse cuando se active este target
Wants=servicio1.service
Wants=servicio2.service
```

Con el wantedBy le estamos diciendo que este target se va a activar cuando el multiuser-target se active correctamente.

Cuando nuestro target se active también se van a activar todas las units que hemos puesto en el fichero de configuración del target: en este caso servicio1.service y servicio2.service.

Después de terminar con la configuración del target tenemos que recargar systemd para que incorpore la configuración de este nuevo target. Para poder hacer eso hacemos el comando **systemctl daemon-reload** (Con ello systemd va a recargar todos los ficheros de configuración sin tener que reiniciar el sistema)

Una vez hecho esto ya podemos comenzar a administrar el target con los comandos de start para activarlo, stop para pararlo, enable para que se active automáticamente, entre otras cosas.

Todo esto es lo que nos permite configurar de forma básica un nuevo target en el sistema.

¿Qué nos permite hacer Systemd?

Systemd proporciona una serie de comandos básicos con los que podemos examinar el estado del sistema y gestionar todo el sistema de servicios. Uno de los principales comandos es ***systemctl*** que tiene muchas funcionalidades diferentes entre las que están:

- ***Systemctl****:* es un comando que permite listar las unidades en ejecución del sistema, hace lo mismo que el comando ***systemctl list-units***. Entre las unidades que lista el comando se incluyen servicios, sockets, montajes, dispositivos y otras unidades que maneja el sistema. La información que muestra es la siguiente:

  Nombre de la unidad: siempre terminan con un punto y seguido de esto indica el tipo de unidad que es, por ejemplo: NetworkManager.service.

  Estado de la unidad: Indica si la unidad ha sido cargada (loaded) correctamente. La siguiente columna también nos da información acerca de la unidad

indicándonos si esta activa, inactiva, fallando. Y la columna SUB no da un estado más detallado dentro del estado activo.

Tipo de unidad: Identifica el tipo de unidad, pueden ser servicios, sockets, montajes, dispositivos, temporizadores, targets.

Los comandos que dan información detalla de un servicio (como systemctl status) incluyen diferentes datos sobre dicha unidad o servicio entre los que se incluyen:

Información sobre el servicio: nos muestra el nombre, una descripción del servicio, el directorio desde donde se carga el archivo de configuración, si este servicio está habilitado para iniciarse automáticamente al arrancar el sistema.

Estado del servicio: nos indica si el servicio esta activado o desactivado y en el caso de estar activo si está en ejecución dándonos además información del tiempo desde el que está corriendo. También nos mostraría si ha ocurrido algún error e información sobre el mismo.

Registro de eventos del servicio: se muestra un registro de los eventos recientes relacionados con el servicio, lo que puede ser útil para diagnosticar problemas o también ayuda para entender el comportamiento del servicio.

Indicación de errores: si hay un problema con el servicio, el comando indica mensajes de error y dará ayudas para tratar de averiguar el origen del problema.

Dependencias del servicio: este comando también muestra las dependencias del servicio, es decir, otros servicios del sistema que tienen que estar activos para que este funcione.

Información sobre el proceso: Muestra información sobre el proceso asociado en caso de que este en estado de ejecución como el pid del proceso, el tiempo de ejecución, el uso de recursos.

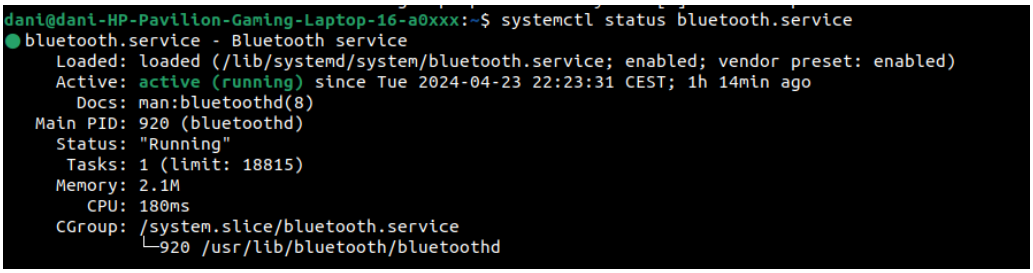


Figura13: Ejemplo de la informacion de un servicio

Systemd gestiona las siguientes funciones

- Administración de servicios: Gestiona los servicios del sistema, controlando el inicio de estos servicios, el estado, el reinicio, ….

- Gestiona las unidades: Además de los servicios del sistema systemd también se va a encargar de otras unidades como sockets, dispositivos (Para la gestión de dispositivos usa udev que es un gestor en Linux que se encarga de controlar los ficheros de dispositivo que hay en /dev, siendo el sucesor de devfs), montajes, …
- Controla las dependencias: Systemd también controla las dependencias entre los servicios de forma que para un servicio concreto podemos ver cuáles son los servicios dependientes que también tienen que estar activos para que funcione correctamente.
- Registro de los eventos del inicio y la gestión del sistema: Este registro de eventos facilita la resolución de problemas y la auditoria del sistema.
- Control de los recursos: Systemd puede limitar y controlar los recursos asignados a los servicios como la memoria, la CPU, … Estos límites se pueden controlar en unos ficheros .service (Los explico más adelante en el funcionamiento detallado)

Funciones que no están dentro de su ámbito

- Es un conjunto de herramientas para la administración del sistema, pero no es un kernel, por lo que depende del kernel para funcionar.
- No es un gestor de paquetes; aunque systemd interactúe con gestores de paquetes para la administración de los servicios.
- Si bien systemd realiza acciones relacionadas con el sistema de archivos, como crear enlaces simbólicos al habilitar y deshabilitar servicios, no se clasifica como un sistema de archivos en sí mismo.

Ventajas e inconvenientes de Systemd

Ventajas:

- Hace que los procesos de arranque sean mucho más sencillos eliminando la necesidad de satisfacer dependencias (sysvinit[1]) gracias a dbus y los sockets.
- Es relativamente rápido, evitando cualquier retraso posible durante el proceso de arranque.
- Realiza una buena gestión de los servicios, pudiendo obtener mucha información sobre cada uno de los servicios del sistema y realizar acciones sobre los servicios.
- Los cgroups se utilizan para realizar un seguimiento de los procesos de servicio, en lugar de los PID.
- Incorpora un sistema de registro de información sobre los servicios que se maneja con la herramienta journald (sistema de registro de systemd). Este sistema de registros guarda datos adicionales sobre lo servicios, permite buscar

[1] Es un proceso estándar de Init para controlar que programas lanza o detiene cuando se inicializa a nivel de ejecución.

de forma eficiente, tiene su sistema de rotación de logs para eliminar los más antiguos, ….

Desventajas:

- No respeta la filosofía Unix de “haz una cosa y hazla bien” dado que este no solo se ocupa de la gestión de servicios del sistema, sino que también tiene otros componentes como journald para la gestión de los registros, udev para la gestión de dispositivos, entre otras utilidades complementarias.
- El software de systemd es incompatible con los operativos BSD, a pesar de compartir el parentesco histórico de Unix, y por consiguiente Init.
- El tamaño de systemd y la gran cantidad de componentes que maneja hace que sea un objetivo potencial para explotar vulnerabilidades y buscar nuevos ataques (desde 2010 se han detectado 9 vulnerabilidades en este software).
- Su estandarización como sistema de arranque y administración de servicios, concebida para tales funciones, ha trascendido su papel inicial, generando una notable dependencia de gran parte del software en Linux. Por ejemplo, el entorno gráfico GNOME requiere la presencia de systemd para un funcionamiento óptimo. Este nivel de integración se extiende a otras herramientas fundamentales como dbus y udev, lo que impone una carga adicional a los desarrolladores del ecosistema Linux.
- Su naturaleza es muy compleja lo que hace que sea muy complicado mejorar el software e incorporar nuevas funcionalidades, además de que el manejo y la resolución de errores se vuelven más difíciles por su extensión. Cabe destacar que los demonios tienen muchas dependencias lo que hace que sea complicado sustituirlos por otros que no generen problemas.
- Para muchos principiantes en Linux, el tamaño de systemd hace que sea considerado como algo dificil de administrar y de comprender.

Explicación en detalle de los componentes de Systemd

Systemd no solo está formado por el proceso de inicio principal del sistema (pid 1) sino que también abarca todo un conjunto de paquetes de software alrededor de él, incluyendo los daemons, journald, logind, networkd junto a otros componentes del sistema de bajo nivel, es decir, es un conjunto de aplicaciones que está formado por un gran número de archivos binarios.

Systemd también integra a muchos otros servicios que son comunes en sistemas de Linux manejando entradas de usuario, la interfaz de línea de comandos, la conexión en caliente con los dispositivos (controlado por udev), registro de ejecución programada (reemplazando a cron), …

Todas estas funcionalidades y esta estructura se pueden resumir en la siguiente imagen:

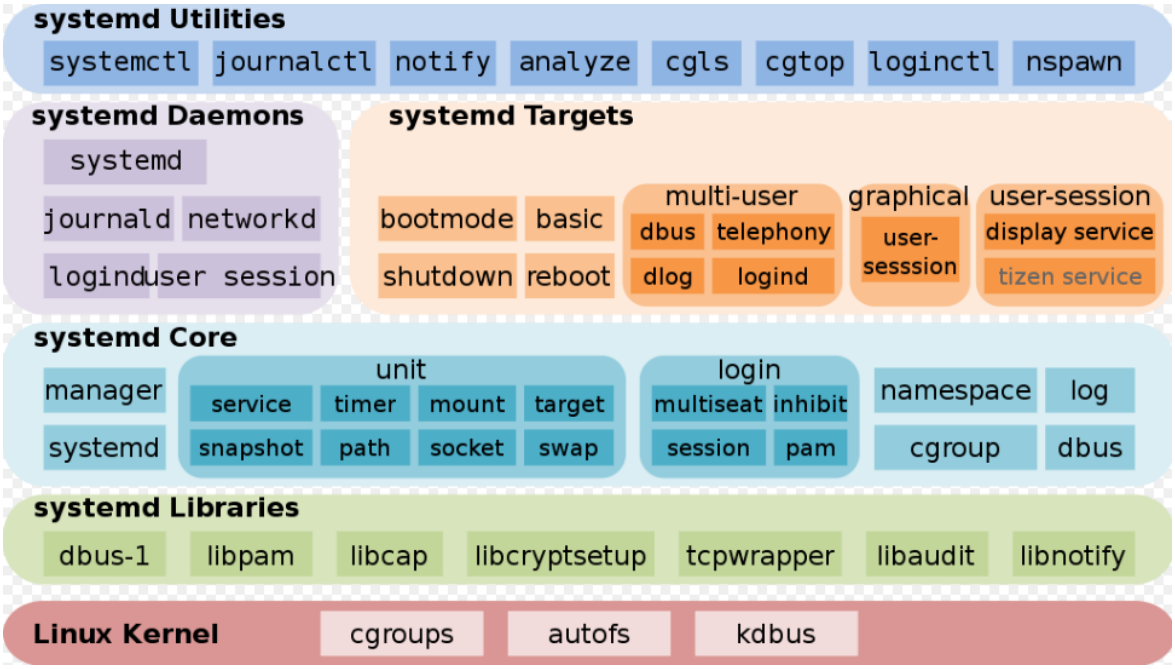


Figura14: Ejemplo de systemd en Tizen (S.O Linux para telefonos)

En las ***librerías de systemd*** tenemos algunos ejemplos:

- **Dbus-1:** librería que contiene todo lo necesario para poder implementar el dbus que es un sistema de comunicaciones que permite que diferentes aplicaciones se comuniquen entre sí, esto es fundamental para el sistema ya que generalmente las aplicaciones no funcionan de forma aislada y se comunican con otras aplicaciones (por temas de dependencias o para compartir datos).
- **Libpam** (Pluggable Authentication Modules): Biblioteca que proporciona la infraestructura para la autenticación en sistemas Linux. Systemd interactúa con libpam para gestionar el inicio de sesión de los usuarios y la autenticación durante el inicio del sistema (según las reglas que tengamos en los diferentes ficheros de configuración de pam).
- **Libcap**: Biblioteca que permite a los programas de Linux realizar operaciones privilegiadas de forma segura y controlada. Systemd puede usar libcap para asignar capacidades especificas a los servicios, lo que permite que los servicios se ejecuten de forma más segura con privilegios reducidos. (Las capabilities en Linux dividen los privilegios de root en unidades más pequeñas y diferenciadas, permitiendo que los procesos tengan un subconjunto de privilegios haciendo que el servicio pueda estar expuesto a menos riesgos porque se está ejecutando con una parte de los privilegios totales del root).
- **Libcryptsetup**: Biblioteca utilizada para administrar y configurar el cifrado de dispositivos en Linux, principalmente a través de herramientas como LUKS (Linux Unified Key setup). Systemd puede interactuar con esta librería para manejar el cifrado de dispositivos durante el proceso de inicio del sistema. (Por ejemplo, puede haber una de las áreas o particiones del disco que este cifrada durante el arranque del sistema).

Luego tenemos ***la parte del core*** de systemd que está formado por diferentes módulos:

- **Unit**: Son los bloques fundamentales de configuración que representan los distintos tipos de objetos que gestiona el sistema y hay diferentes tipos como servicie, mount, path, entre otros. Mas adelante entraremos en detalle.
- **Login**: Es el administrador de sesiones de systemd. Gestiona las sesiones del usuario, el inicio de sesión y la autenticación entre otras cosas.[2]
- **Dbus**: Indispensable para el correcto funcionamiento de systemd ya que es lo que permite comunicar las aplicaciones entre sí y esto es esencial para el funcionamiento de la mayoría de las aplicaciones.
- **Log:** El sistema de log de systemd gestionado por journalctl.
- **Cgroup** : Función de Linux para limitar, priorizar y asignar recursos para la ejecución de los procesos como por ejemplo porcentaje de CPU, memoria, tiempo de CPU, entre otros.

Por otra parte, nos encontramos con los ***daemons*** de systemd que son procesos del sistema que se inician durante el arranque del sistema y cuya ejecución no termina hasta que se paren manualmente o hasta que se apague la máquina. Son procesos no iterativos que no disponen de una terminal de control con la que pueda interactuar el usuario. Además, su salida de error y el propio resultado de su funcionamiento será registrado en forma de logs en /var/logs en sistemas modernos.

Por ejemplo, el demonio de login (logind), registra entre otras cosas los accesos al sistema en /var/log/auth.log

También contamos con los ***targets*** de systemd, que son un conjunto de unidades/units que se van a ejecutar en unas determinadas circunstancias y en un orden concreto, anteriormente conocidos como runlevels.

Por último, tenemos la parte de las ***utilidades*** de systemd entre las que tenemos:

- **Systemctl**: Herramienta para administrar y analizar información de los servicios del sistema.
- **Journalctl:** Herramienta para analizar los logs de systemd desde un único lugar.
- **Analyze:** Herramienta de análisis del proceso de arranque del sistema y que además proporciona información útil sobre los tiempos de inicio de los servicios y otros componentes del sistema, a través de *blame*.
- **Loginctl:** Herramienta para administrar usuarios y sus sesiones con sus respectivos procesos, por ejemplo, con el flag *list-sessions* podemos averiguar que tty está empleando un determinado usuario.

Ejemplos del uso de las utilidades de Systemd

- ***systemctl***: muestra una lista de todos las units del sistema, entre ellas dispositivos servicios, targets, sockets, entre otros.

[2] Está estrechamente relacionado con PAM ya que controla quien puede iniciar sesión durante el arranque del sistema (Modulo de login de PAM).

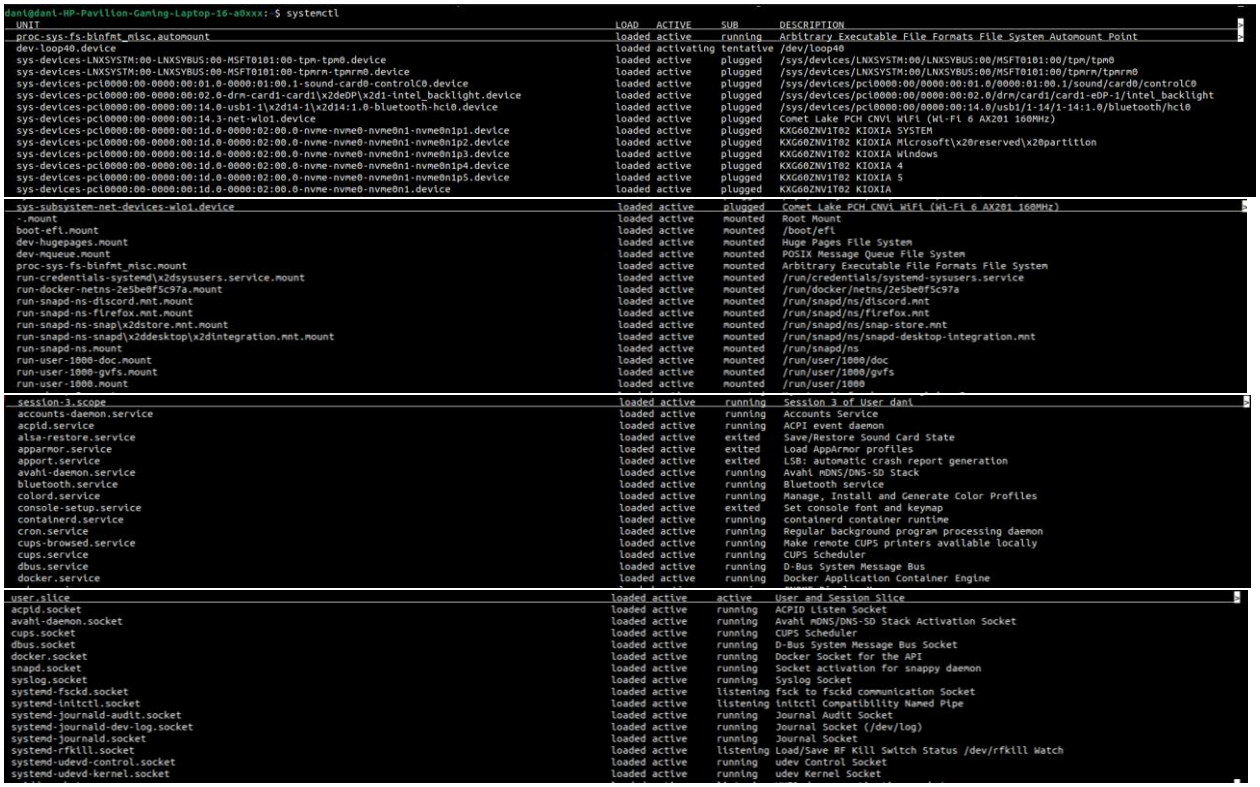


Figura15: Salida del comando systemctl

- ***Systemctl list-unit-files:*** permite tener un listado de las unidades instaladas en el sistema junto con su estado.

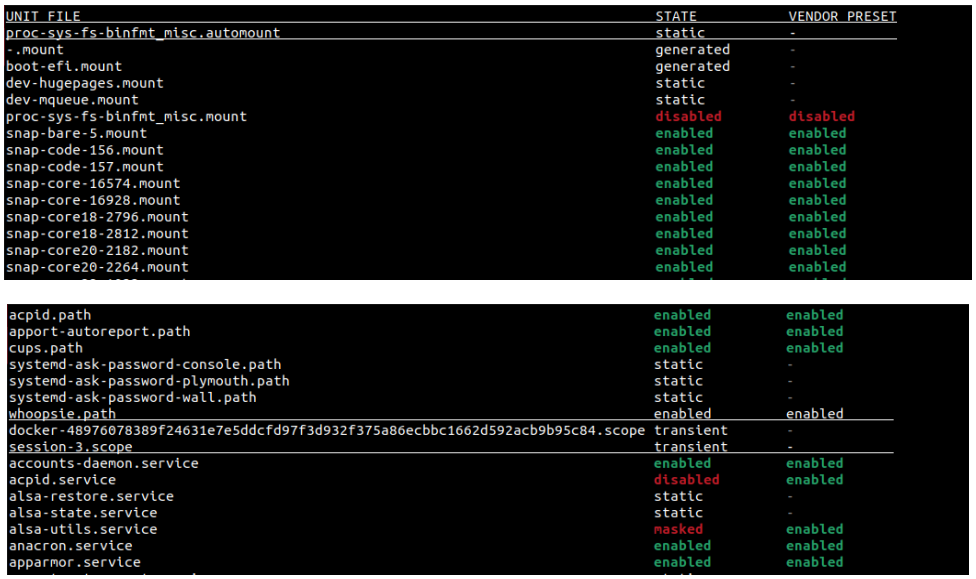


Figura16: Unidades instaladas en el sistema junto con su estado

- ***systemctl list-dependencies unit.tipoUnit***: muestra las dependencias de la unit. Útil para investigar que dependencia tiene la responsabilidad de una caída de una unit determinada.

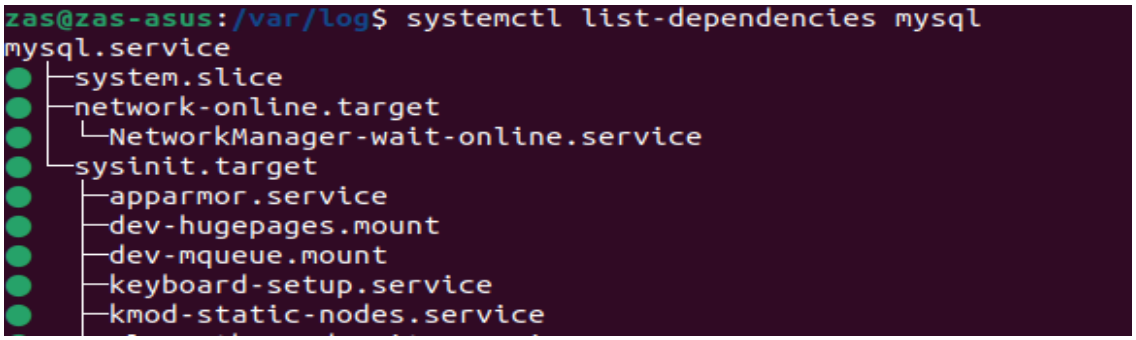
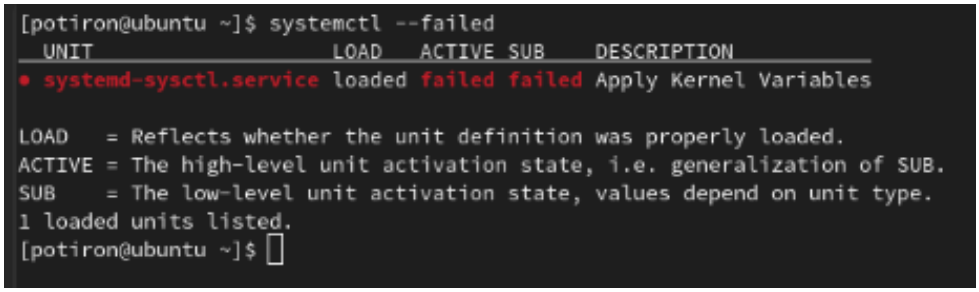


Figura16: Comando para ver las dependencias de una unit

- ***Systemctl –failed****:* permite listas las unidades que han fallado o no se han desplegado correctamente.



- ***Systemctl status [unit]:*** nos muestra información detallada sobre el estado de un servicio del sistema:

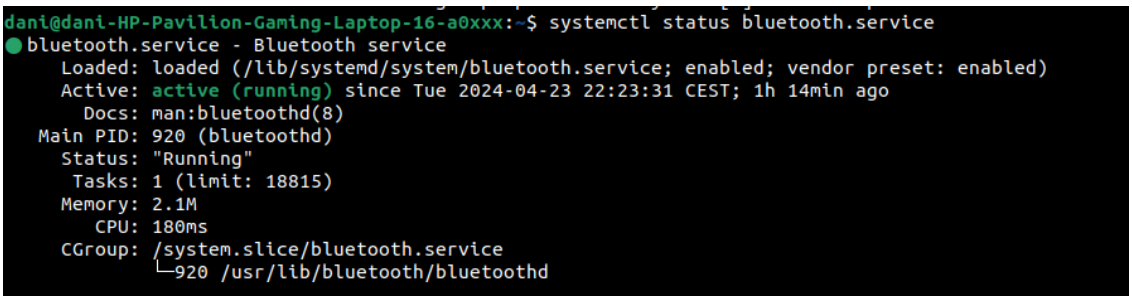


Figura17: Comando para ver los detalles de un servicio concreto

- ***systemctl start/stopunit.tipoUnit***: permite arrancar y parar un servicio respectivamente.

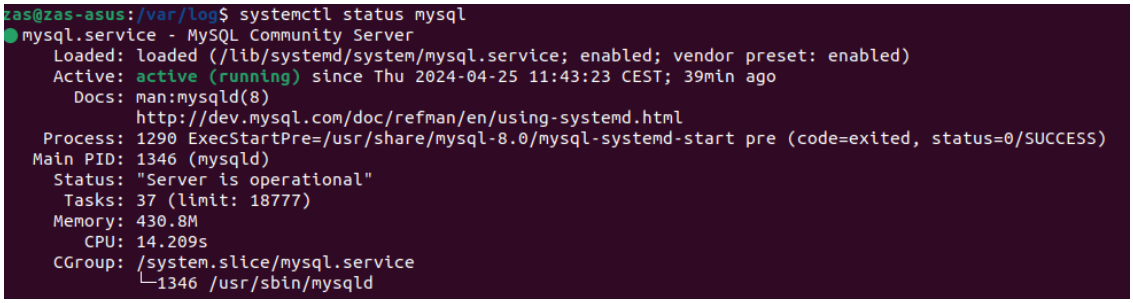


Figura17: Salida del comando systemctl

- ***systemctl enable/disable unit.tipoUnit*:** habilita o deshabilita una unit en el arranque del sistema. Para asegurarnos de que este no se reactiva otra vez podemos usar la opción *mask.*

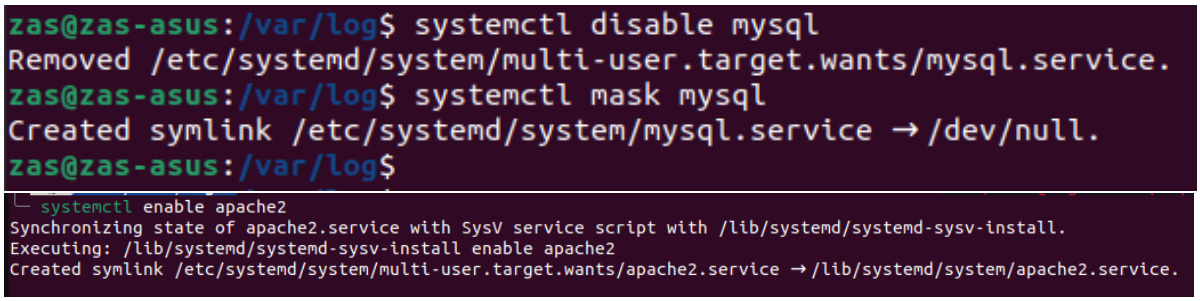


Figura18: Comandos para habilitar y hacer un mask al servicio

- ***journalctl***: muestra todos los registros desde el inicio hasta el momento actual.

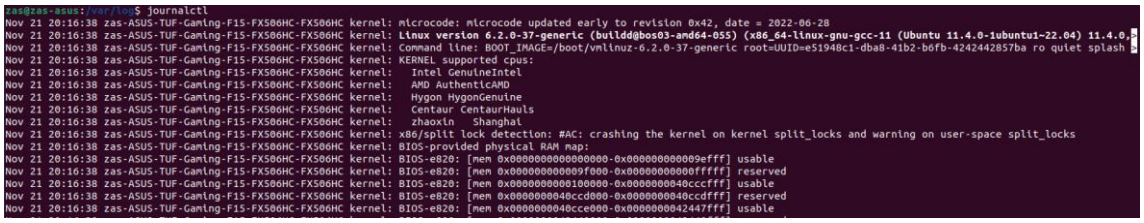


Figura19: Salida del comando journalctl

- ***journalctl –b:*** muestra los registros del arranque actual; podemos acceder a logs de anteriores arranques modificando el flag –b.

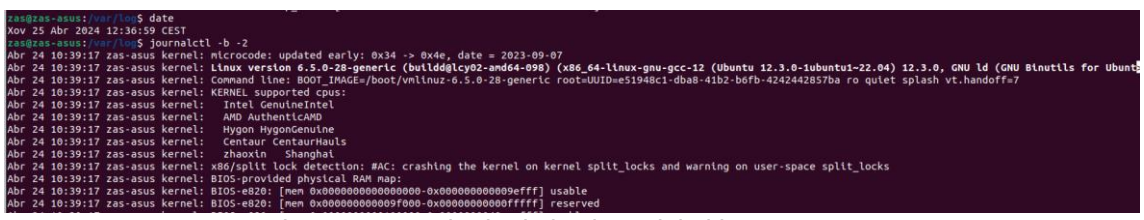


Figura19: Resultado de lso logs del ultimo arranque

- ***journalctl --user usuario:*** muestra los registros de la sesión del usuario concreto.

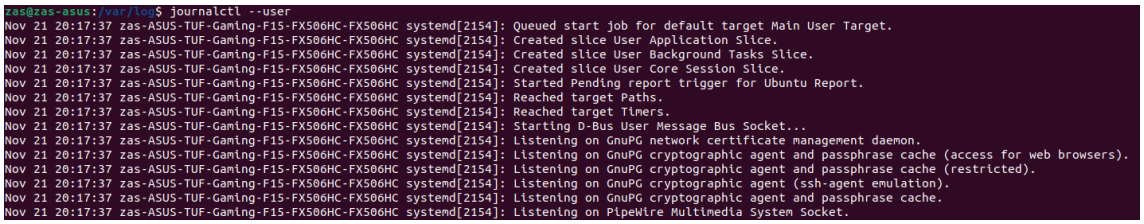


Figura20: Comando para mostar los logs de un usuario

- ***journalctl –u unit.tipoUnit:*** muestra los registros de una unit específica.

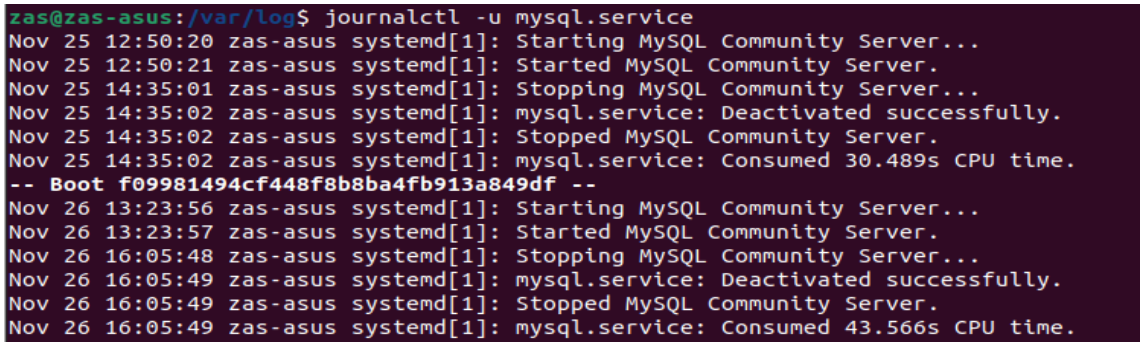


Figura21: Logs de una unidad especifica

- ***journalctl –u unit.tipoUnit >> fichero****:* exportamos los registros a un fichero.

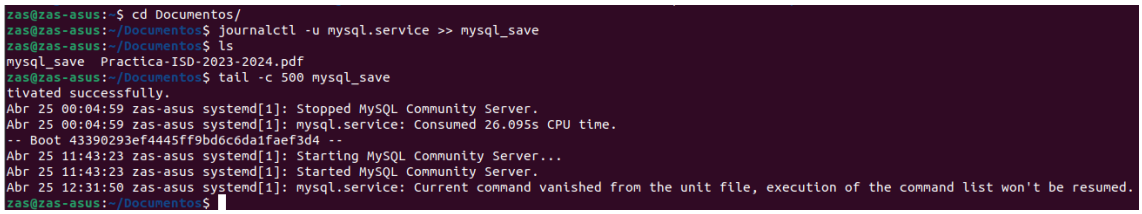


Figura22: Comando y resutado de la exportacion de los logs a un fichero

- ***systemd-analyze blame***: muestra la lista de todas las units del sistema ordenados por tiempo de ejecución, ordenados de mayor a menor.

```
zas@zas-asus:~/Documentos$ systemd-analyze blame
40.013s apt-daily-upgrade.service
10.130s gpu-manager.service
 6.791s NetworkManager-wait-online.service
 5.638s plymouth-quit-wait.service
  842ms snapd.service
  624ms mysql.service
  603ms systemd-resolved.service
  596ms docker.service
  511ms fwupd.service
  485ms qemu-kvm.service
  452ms apparmor.service
  371ms systemd-oomd.service
  357ms snapd.seeded.service
```

Figura23: Units del sistema y su tiempo total de ejecucion

- ***systemd-analyze critical-chain***: muestra la cadena critica de unidades, que es la secuencia de units (mayormente servicios) que más influye en el tiempo de inicio del sistema. Esto nos permite identificar los responsables de posibles retrasos en el arranque del sistema.

```
zas@zas-asus:~/Documentos$ systemd-analyze critical-chain
The time when unit became active or started is printed after the "@" character.
The time the unit took to start is printed after the "+" character.

graphical.target @13.913s
└─multi-user.target @13.913s
  └─plymouth-quit-wait.service @8.274s +5.638s
    └─systemd-user-sessions.service @8.270s +2ms
      └─network.target @1.386s
        └─systemd-resolved.service @782ms +603ms
          └─systemd-tmpfiles-setup.service @593ms +31ms
            └─local-fs.target @582ms
              └─run-snapd-ns-snapd\x2ddesktop\x2dintegration.mnt.mount @12.538s
                └─run-snapd-ns.mount @11.688s
                  └─local-fs-pre.target @290ms
                    └─systemd-tmpfiles-setup-dev.service @277ms +13ms
                      └─systemd-sysusers.service @266ms +10ms
                        └─systemd-remount-fs.service @221ms +42ms
                          └─systemd-journald.socket @205ms
                            └─system.slice @170ms
                              └─-.slice @170ms
```

Figura24: Cadena critica de unidades del sistema

- ***systemd-anaylze time***: proporciona el tiempo total que lleva iniciar el sistema, desde el arranque hasta el momento en que el usuario obtiene el control del sistema al llegar a la pantalla de inicio de sesión. Separa los tiempos en diferentes grupos o categorías:
  - Firmware: este es el tiempo que el firmware de la computadora tarda en cargar antes de cargar el propio cargador de arranque.
  - Loader: Es el tiempo que el cargador de arranque (bootloader) tarda en cargar el kernel del sistema operativo.
  - Kernel: Tiempo que tarda el kernel/nucleo del sistema operativo en iniciarse completamente y cargar todo aquello que sea necesario
  - Userspace: Tiempo que va a llevar iniciar todos los servicios y procesos del espacio del usuario una vez que el kernel se ha cargado correctamente (configuración de la red, montaje del sistema de archivos, ...)

```
zas@zas-asus:~/Documentos$ systemd-analyze time
Startup finished in 4.131s (firmware) + 2.442s (loader) + 1.917s (kernel) + 13.955s (userspace) = 22.447s
graphical.target reached after 13.913s in userspace
```

Figura25: Información sobre el tiempo de arranque del sistema

Ejemplo práctico basado en las units

Realizaremos un ejemplo donde pondremos a prueba algunos de los puntos mencionados anteriormente. Este ejercicio no solo nos ayudará a entender mejor dichos puntos, sino que también demostrará cómo se pueden realizar acciones que simplifican diversas tareas para los usuarios.

El ejemplo se basa en la realización de un script que hará una copia de seguridad del directorio donde se encuentran los registros del sistema.

En primer lugar, vamos a crear el directorio donde vamos a almacenar los backups, en este caso en el directorio /home/usuario/backup.

```
usuario@prueba:~$ mkdir /home/usuario/backup
usuario@prueba:~$ ls /home/usuario/backup/
usuario@prueba:~$
```

Después de eso vamos a crear el fichero .sh que va a ejecutar nuestro service. Este script lo vamos a meter en el directorio /usr/local/bin que es un directorio que contiene programas ejecutables instalados manualmente por el usuario o administrador del sistema entre otras cosas.

```
usuario@prueba:~$ nano /usr/local/bin/backup.sh
```

Dentro del fichero vamos a tener el siguiente contenido:

```
# What to backup.
backup_files="/etc/hosts.allow /etc/hosts.deny /etc/resolv.conf  /etc/nsswitch.conf /etc/apt/sources.list "

# Where to backup to.
dest="/home/usuario/backup"


# Create archive filename.
day=$(date +%A)
hostname=$(hostname -s)
archive_file="$hostname-$day.tgz"
scriptsaludo=$dest/saludo.sh

# Capitalizar la primera letra del día de la semana
dia=$(date +%A)
dia_minusculas=$(echo $dia | tr '[:upper:]' '[:lower:]')
dia_capitalizado="$(tr '[:lower:]' '[:upper:]' <<< ${dia_minusculas:0:1})${dia_minusculas:1}"
# Crear la línea con la fecha actual
nuevotexto="echo \"[${dia_capitalizado^}]      [Ultima Copia: \'$(date)']\""
# Usar sed para editar la línea correspondiente en saludo.sh
sed -i "/\[${dia_capitalizado^}\]/c\\$nuevotexto" $scriptsaludo

nuevotexto="echo \"[Ultima Copia: \'$(date)']\""
# Usar sed para sobrescribir la última línea en saludo.sh con el nuevo texto
sed -i "\$c\\$nuevotexto" $scriptsaludo

# Print start status message.
echo "Backing up $backup_files to $dest/$archive_file"
date
echo

# Backup the files using tar.
tar czf $dest/$archive_file $backup_files
# Print end status message.
echo
echo "Backup finished"
echo
# Long listing of files in $dest to check file sizes.
ls -lh $dest
$scriptsaludo 2>&1
```

Fichero de referencia donde guarda las cosas: saludo.sh:

```
  GNU nano 6.2                 /home/usuario/backup/saludo.sh
echo "[Monday]       [Ultima Copia:     ]"
echo "[Tuesday]      [Ultima Copia:     ]"
echo "[Wednesday]    [Ultima Copia:     ]"
echo "[Thursday]     [Ultima Copia:     ]"
echo "[Friday]       [Ultima Copia:     ]"
echo "[Saturday]     [Ultima Copia:     ]"
echo "[Sunday]       [Ultima Copia:     ]"
echo "[Ultima Copia: ]"
```

El funcionamiento de este fichero es el siguiente:

En la primera parte del fichero vamos a definir cuáles son los ficheros de los que vamos a hacer el backup y cuál va a ser la ruta donde vamos a guardarlos; esa ruta va a ser donde esté el saludo.sh para modificar el fichero con los nuevos datos.

Luego, vamos a transformar el formato del día de la semana para poder obtener el nombre con la primera letramayúsculaa y así poder hacer que cuadre con el formato del día que hay en el fichero.

Después vamos a usar sed y una expresión regular para buscar la fila del fichero que tenga ese día de la semana y reemplazar el contenido de esa fila por lo nuevo que vamos a meter que va a ser lo mismo, pero además con el valor de la variable date que es la hora en la que hemos hecho el backup.

Por último, vamos a reemplazar la última línea del fichero por la hora del último backup que en este caso también es el valor de la variable date. Es algo muy parecido a la fila de arriba.

Vemos que nos da esta salida mostrando ya el contenido del fichero modificado:

```
Backup finished

total 8.0K
-rw-r--r-- 1 root root 1.5K Apr 30 20:28 aso2-Tuesday.tgz
-rwxrwxrwx 1 root root  381 Apr 30 20:28 saludo.sh
[Monday]       [Ultima Copia:     ]
[Tuesday]      [Ultima Copia: 'Tue Apr 30 20:28:22 UTC 2024']
[Wednesday]    [Ultima Copia:     ]
[Thursday]     [Ultima Copia:     ]
[Friday]       [Ultima Copia:     ]
[Saturday]     [Ultima Copia:     ]
[Sunday]       [Ultima Copia:     ]
[Ultima Copia: 'Tue Apr 30 20:28:22 UTC 2024']
root@aso2:/usr/local/bin#
```

Si vamos al fichero también podemos ver que se ha modificado con éxito:

```
usuario@aso2:~$ cat /home/usuario/backup/saludo.sh
echo "[Monday]       [Ultima Copia:     ]"
echo "[Tuesday]      [Ultima Copia: 'Tue Apr 30 20:28:22 UTC 2024']"
echo "[Wednesday]    [Ultima Copia:     ]"
echo "[Thursday]     [Ultima Copia:     ]"
echo "[Friday]       [Ultima Copia:     ]"
echo "[Saturday]     [Ultima Copia:     ]"
echo "[Sunday]       [Ultima Copia:     ]"
echo "[Ultima Copia: 'Tue Apr 30 20:28:22 UTC 2024']"
usuario@aso2:~$
```

Ahora sabiendo que está bien ya podemos hacer el service:

Vamos a crear un fichero .service que vamos a meter en /etc/systemd/system y que va a tener el siguiente contenido:

```
root@aso2:/usr/local/bin# cat /etc/systemd/system/backup.service
[Unit]
Description = "Efectua el backup de ciertos archivos"
After = "network.target"

[Service]
ExecStart = "/usr/local/bin/backup.sh"

[Install]
WantedBy = "default.target"
root@aso2:/usr/local/bin#
```

Dentro de este fichero le estamos diciendo que se ejecute después del network.target y además le estamos diciendo que en la parte del service ejecute el .sh que hemos creado anteriormente.

También vamos a tener una dependencia con el default.target con lo que estamos especificando las condiciones en la que el servicio es activado. (Es decir el servicio se inicia cuando el sistema alcance ese estado del target).

Una vez hecho esto ya tenemos todo preparado para poder iniciar el servicio en el sistema.

Vamos a hacer un enable y ver su estado:

```
root@aso2:/usr/local/bin# systemctl enable backup.service
Created symlink /etc/systemd/system/default.target.wants/backup.service → /etc/systemd/system/backup.service.
root@aso2:/usr/local/bin#
```

Con el enable ya podemos ver que se ha creado un enlace desde el default target al service debido al wanted by que hemos puesto.

```
root@aso2:/usr/local/bin# systemctl start backup.service
root@aso2:/usr/local/bin# systemctl status backup.service
○ backup.service - "Efectua el backup de ciertos archivos"
     Loaded: loaded (/etc/systemd/system/backup.service; enabled; vendor preset: enabled)
     Active: inactive (dead) since Tue 2024-04-30 20:42:34 UTC; 1s ago
    Process: 30803 ExecStart=/usr/local/bin/backup.sh (code=exited, status=0/SUCCESS)
   Main PID: 30803 (code=exited, status=0/SUCCESS)
        CPU: 21ms

Apr 30 20:42:34 aso2 backup.sh[30820]: -rwxrwxrwx 1 root root  381 Apr 30 20:42 saludo.sh
Apr 30 20:42:34 aso2 backup.sh[30821]: [Monday]       [Ultima Copia:     ]
Apr 30 20:42:34 aso2 backup.sh[30821]: [Tuesday]      [Ultima Copia: 'Tue Apr 30 20:42:34 UTC 2024']
Apr 30 20:42:34 aso2 backup.sh[30821]: [Wednesday]    [Ultima Copia:     ]
Apr 30 20:42:34 aso2 backup.sh[30821]: [Thursday]     [Ultima Copia:     ]
Apr 30 20:42:34 aso2 backup.sh[30821]: [Friday]       [Ultima Copia:     ]
Apr 30 20:42:34 aso2 backup.sh[30821]: [Saturday]     [Ultima Copia:     ]
Apr 30 20:42:34 aso2 backup.sh[30821]: [Sunday]       [Ultima Copia:     ]
Apr 30 20:42:34 aso2 backup.sh[30821]: [Ultima Copia: 'Tue Apr 30 20:42:34 UTC 2024']
Apr 30 20:42:34 aso2 systemd[1]: backup.service: Deactivated successfully.
root@aso2:/usr/local/bin#
```

Al hacer el enable vemos que el servicio ya está activo y en este caso pone que no está en ejecución porque el servicio es hace una copia de seguridad y ya termina no es algo continuo.

Ahora para probar si reiniciamos el sistema podemos ver como se hace la copia correctamente.

Ejemplo de cómo librarse de systemd:

Para conseguir esto usaremos de ejemplo un software bastante común en las máquinas de todos los estudiantes universitarios españoles, el cliente vpn pulse-secure (linux) que permite el acceso a las redes internas de nuestra facultad, y en un contexto más nacional, el acceso a *eduroam*.

Lo primero que hemos hecho es analizar el contenido del paquete de pulse secure para un sistema de systemd y nos podemos encontrar con el siguiente contenido:

```
root@aso1:/home/usuario/Desktop/PulseSecure# ls
DEBIAN  lib  opt  usr  var
root@aso1:/home/usuario/Desktop/PulseSecure#
```

Vemos que tenemos 5 carpetas. Algunas de las importantes que vamos a usar son lib, opt y usr.

La carpeta DEBIAN no la vamos a usar y la vamos a eliminar ya que tiene ficheros que son de systemd que no los vamos a usar.

Dentro de la carpeta lib vamos a tener el pulsesecure.service que es la unit que se usaría en systemd y que nosotros vamos a modificar para que se pueda usar en otro operativo que no tenga systemd.

```
root@aso1:/home/usuario/Desktop/PulseSecure/lib/
e.service
[Unit]
Description=pulsesecure service Daemon
After=network.target
[Service]
Type=forking
Restart=always
RestartSec=1
User=root
ExecStart=/opt/pulsesecure/bin/startup.sh start
ExecStop=/opt/pulsesecure/bin/startup.sh stop

[Install]
WantedBy=multi-user.target
```

Luego tenemos el directorio opt que tiene los binarios y las librerías que se van a usar para poder gestionar el pulse secure y eso sí que lo vamos a necesitar:

```
root@aso1:/home/usuario/Desktop/PulseSecure/opt/pulsesecure# cd bin/
root@aso1:/home/usuario/Desktop/PulseSecure/opt/pulsesecure/bin# ls
cefBrowser               jamCommand             pulsesecure   startup.sh
cefSubProcess            LogsAndDiagnostics.sh  pulseUI
certificate_installer.sh pulselauncher          setup_cef.sh
root@aso1:/home/usuario/Desktop/PulseSecure/opt/pulsesecure/bin#
```

Por ejemplo, en el bin tenemos el startup.sh que es para poder arrancar el pulse (le pasamos start como argumento para arrancar y stop para parar)

Por último en el directorio usr vamos a tener, entre otras cosas un .desktop que es el que permite cargarlo como entrono grafico:

```
root@aso1:/home/usuario/Desktop/PulseSecure/usr/share/applications# ls
pulse.desktop
root@aso1:/home/usuario/Desktop/PulseSecure/usr/share/applications#
```

En primer lugar, vamos a mover el directorio pulseSecure de opt a la carpeta /opt/ para tener ahí todos los binarios y las librerías;

```
root@aso1:/home/usuario/Desktop/PulseSecure/opt#
           mv pulsesecure/ /opt/
```

Ahora vamos a tener que crear nuestro servicio para el systemd tomando como referencia el .service que nos venia en el paquete. Para ver el formato del servico que tenemos que crear y como es la estructura del archivo podemos ver algunos en /etc/init.d/. Este es el lugar donde vamos a meter el nuestro.

Ejemplo de un fragmento de un servicio ya hecho en /etc/init.d

```
#! /bin/sh
### BEGIN INIT INFO
# Provides: bluetooth
# Required-Start:    $local_fs $syslog $remote_fs dbus
# Required-Stop:     $local_fs $syslog $remote_fs
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start bluetooth daemons
### END INIT INFO
#
# bluez    Bluetooth subsystem starting and stopping
#
# originally from bluez's scripts/bluetooth.init
#
# Edd Dumbill <ejad@debian.org>
# LSB 3.0 compilance and enhancements by Filippo Giunchedi <filippo@debian.o
case $1 in
  start)
        log_daemon_msg "Starting $DESC"

        if test "$BLUETOOTH_ENABLED" = 0; then
                log_progress_msg "disabled. see /etc/default/bluetooth"
                log_end_msg 0
                exit 0
        fi
```

Servicio que hemos hecho nosotros para el pulse secure:

```
root@aso1:/home/usuario/Desktop/PulseSecure/lib/systemd# cat /etc/init.d/pulse
secure
#!/bin/bash
#
# pulsesecure       This starts and stops the pulsesecure service
#
# chkconfig: 345 20 80
# description: pulsesecure service
# processname: pulsesecure
### BEGIN INIT INFO
# Provides:       pulsesecure
# Required-Start: $network
# Required-Stop:  $network
# Default-Start:  3 4 5
# Default-Stop:   0 1 2 6
# Short-Description: pulsesecure service
# Description:    This script manages the pulsesecure service.
### END INIT INFO
# Source function library.
```

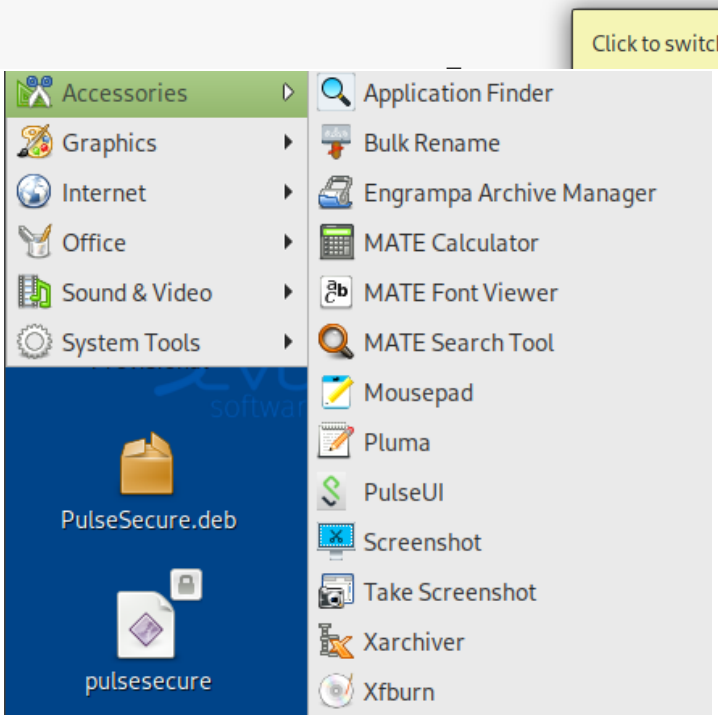
```
# Check if pulsesecure is installed
if [ ! -x "$PULSESTART" ]; then
    echo "Error: pulsesecure script not found or not executable."
    exit 1
fi
start() {
    echo -n "Starting pulsesecure:"
    exec $PULSESTART start

}
stop() {
    echo -n "Stopping pulsesecure:"
    exec $PULSESTART stop

}

# Función para verificar el estado del servicio
status_service() {
    echo -n "Checking status of servicio: "
```

```
case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    restart)
        stop
        start
        ;;
    status)
        status_service
        ;;
    *)
        echo "Usage: $0 {start|stop|restart|status}"
        exit 1
        ;;
esac
```

Este script lo vamos a colocar en /etc/init.d junto con el resto de los archivos.

Ahora para tener también la versión grafica vamos a mover el .desktop que hemos dicho antes y lo colocamos en /usr/share/applications junto con el resto de las aplicaciones:

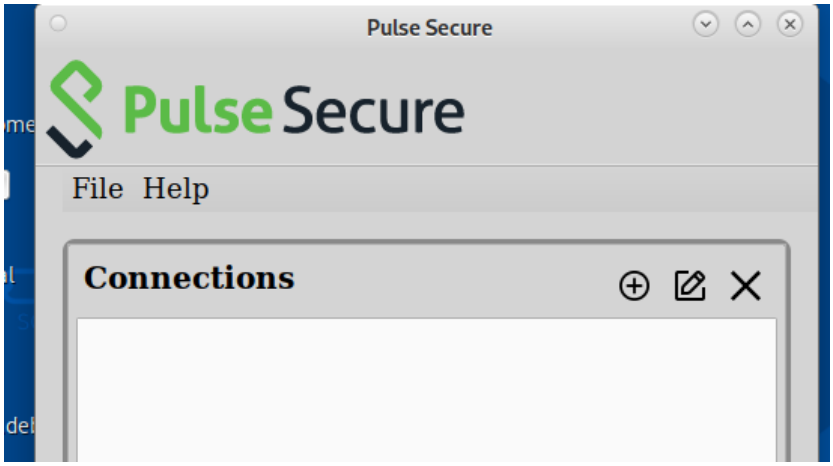
```
root@aso1:/home/usuario/Desktop/PulseSecure# mv usr/share/applications/pulse.d
esktop  /usr/share/applications/
root@aso1:/home/usuario/Desktop/PulseSecure# ls /usr/share/applications/ | gre
p pulse
pulse.desktop
```



Ahora para ver que funciona ya podemos probar con el comando service para gestionar el servico y podemos hacer start, stop, status.

```
root@aso1:~# service pulsesecure status
Checking status of servicio: Stopped
root@aso1:~# service pulsesecure start
Starting pulsesecure:root@aso1:~#
root@aso1:~# service pulsesecure status
Checking status of servicio: Running
root@aso1:~# pgrep -x pulsesecure
2847
root@aso1:~# service pulsesecure stop
Stopping pulsesecure:root@aso1:~#
root@aso1:~# service pulsesecure status
Checking status of servicio: Stopped
root@aso1:~# pgrep -x pulsesecure
root@aso1:~#
```

Además, también va con el .desktop si lo probamos



Con esto conseguimos librarnos de systemd y adaptar un servicio para no depender del mismo.

Bibliografía:

- pulsecure.deb
- https://www.zdnet.com/article/debian-init-decision-further-isolates-ubuntu/
- https://es.wikipedia.org/wiki/Systemd
- https://keepcoding.io/blog/que-es-systemd/
- https://wiki.archlinux.org/title/systemd_(Espa%C3%B1ol)
- https://blog.alcancelibre.org/staticpages/index.php/introduccion-systemd
- https://www.redeszone.net/2014/09/13/razones-para-tener-una-distribucion-linux-con-systemd/
- https://debiandulce.blogspot.com/2017/07/systemd-la-rosa-con-espinas.html
- https://geekland.eu/systemctl-administrar-servicios-linux/
- https://wiki.archlinux.org/title/Dm-crypt_(Espa%C3%B1ol)/Specialties_(Espa%C3%B1ol)
- https://juncotic.com/proceso-de-inicio-de-gnu-linux-systemd/
- https://es.opensuse.org/SDB:Systemd_target
- https://www.digitalocean.com/community/tutorials/understanding-systemd-units-and-unit-files