

# HowTo-ASIX Systemd

Curs 2019 - 2020

---

<b>Preinstal·lació</b>	<b>3</b>
<b>Arrencada del sistema</b>	<b>7</b>
Visió global	7
Procediment d'arrencada	8
Systemd / runlevels	10
Targets d'arrencada	11
Targets / runlevels	11
Target per defecte	12
Iniciar el sistema a un target diferent	14
Emergency / Salvar els mobles! init=/bin/bash	15
Llistar units i dependències	16
Eines per monitorar l'arrencada	20
<b>Serveis / systemctl</b>	<b>21</b>
Gestió bàsica dels serveis	21
Status d'un servei	21
Activar / Desactivar un servei	22
Enable / Disable d'un servei	25
Restart / Reload d'un servei	27
Mask / Unmask Enmascarar un servei	27
Tipus de unit .service	29
Identificar el fitxer de servei / Contingut d'un paquet	29
Identificar l'enable / symlink manual	35
<b>El model de funcionament de systemd</b>	<b>38</b>
Arquitectura de systemd	38
Systemd Units	39
Systemd .target / Analitzem l'arrencada	40
Target dependències	44
Analitzem l'arrencada d'un servei	47
Arrencades de targets diferents	49
targets de recuperació i emergència	49
emergency.target	50
rescue.target	51

Altres targets de systemd	52
Gràfics de l'arrencada i de les relacions de les units	54
Esquemes d'arrencada en mode text	54
Systemctl list...	56
System-analyze	56
Systemd-analyze blame	57
Systemd-analyze time	58
Systemd-analyze plot	58
Systemd-analyze critical-chain	59
Systemd-analyze dot	60
<b>Crear units propies: targets i services</b>	<b>62</b>
<b>Under Construction</b>	<b>63</b>
Snapshot	64
Service WantedBy	64
Crear target propi	65
Crear serveis propis	65

---

# Preinstal·lació

Aquesta NO té a veure amb systemd sinó amb com utilitzar màquines virtuals per treballar. Està especialment pensada per usar amb les pràctiques de GRUB però s'ha inclòs també aquí per si voleu fer pràctiques de systemd més arriscades i preferiu no fer-ho al host de treball.

## Advertència:

- En aquest dossier es treballa amb imatges per a sistemes de 64bits, X86\_64.
- Alguns dels exercicis són perillos i poden espatllar el sistema si cometeu errors (si els executeu en el vostre propi host i no en màquines virtuals)
- `# virt-sysprep -a Fedora-Cloud-Base-27-1.6.x86_64.qcow2 --root-password file:passwd.txt`
- `# virt-sysprep -a Fedora-Cloud-Base-27-1.6.x86_64.raw --root-password file:passwd.txt`
- `cat passwd.txt`  
`secret`
- `# systemctl start libvirtd.service`
- `# systemctl start libvirt-guests.service`
- `# chown -R usuari.usuari <directori-imatges>`
- `$ qemu-kvm -hda Fedora-Cloud-Base-27-1.6.x86_64.qcow2`

Eines per treballar l'arrencada / instal·lació / grub de sistemes:

- Màquines virtuals a isard: <https://isard.escoladeltreball.org>
- Màquines virtuals al host.
- Hosts físics.










Per treballar amb màquines virtuals:

- \* **libvirt / virt-manager del grup de paquets @Virtualization.**
- Virtualbox.

Imatges per descarregar i treballar amb elles:

- Archive de Fedora de totes les seves releases i projectes:  
<https://archives.fedoraproject.org/pub/archive/fedora/>

## Index of /pub/archive/fedora/linux/releases/27






Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>		-	
 <a href="#">COMPOSE_ID</a>	2017-11-10 20:47	20	
 <a href="#">CloudImages/</a>	2017-11-10 19:04	-	
 <a href="#">Docker/</a>	2017-11-10 18:50	-	
 <a href="#">Everything/</a>	2017-11-10 18:52	-	
 <a href="#">Server/</a>	2017-11-10 19:04	-	
 <a href="#">Spins/</a>	2017-11-10 18:48	-	
 <a href="#">Workstation/</a>	2017-11-10 18:50	-	
 <a href="#">Workstation0stree/</a>	2017-11-10 21:18	-	

### [Fedora 27](#):

- ❑ **Server**: imatges de servidor Fedora. Recordeu que el servidor és per funcionar en mode text sense entorn gràfic.
- ❑ **Workstation**: imatges per a treballar amb estacions de treball com les de l'aula.
- ❑ **Docker**: imatges docker amb les que es treballa a segon de així.
- ❑ **CloudImages**: imatges preparades per treballar al cloud o com a imatges de màquines virtuals.

Exemple d'imatges de Fedora 27 [Workstation](#):

## Index of /pub/archive/fedora/linux/releases/27/Workstation/x86\_64/iso

Name	Last modified	Size	Description
 <a href="#">Parent Directory</a>		-	
 <a href="#">Fedora-Workstation-27-1.6-x86_64-CHECKSUM</a>	2017-11-10 18:47	1.2K	
 <a href="#">Fedora-Workstation-Live-x86_64-27-1.6.iso</a>	2017-11-05 07:45	1.5G	
 <a href="#">Fedora-Workstation-netinst-x86_64-27-1.6.iso</a>	2017-11-05 04:59	508M	
 <a href="#">Fedora-Workstation-netinst-x86_64-27-1.6.iso.manifest</a>	2017-11-05 06:48	700	

Podem observar que es proporcionen dos tipus d'imatges (el mateix passa amb Server):

- ❑ **Live**: una imatge de Fedora-Workstation\_live en format ISO que permet tant l'arrancada del sistema operatiu en una Live com la instal·lació al disc dur.
- ❑ **Netinst**: Imatge lleugera (500M) d'instal·lació d'un sistema operatiu Fedora27-Workstation al disc dur usant una connexió a internet. La imatge és lleugera (un DVD sencer d'instal·lació són +3GB) perquè conté el mínim, la resta de la instal·lació es fa descarregant els paquets via xarxa. Per això es diu [netinst](#), instal·lació via xarxa.

L'avantatge respecta a un DVD clàssic d'instal·lació és que la instal·lació final ja està actualitzada, no cal fer el update. Cal tenir present que quan es fabrica un DVD d'instal·lació es fa una foto 'fixa' de com és el sistema en aquella data. Si per exemple el DVD es fabrica al febrer i instal·lem el sistema al juny, del febrer al juny el sistema a anat variant amb actualitzacions que no estan al DVD. En canvi netinst instal·larà sempre la última versió disponible dels paquets segons la versió de sistema.

- ❑ DVD: en el cas de workstation no hi ha versió DVD però si navegeu per mirar les ISO que es proporcionen pel Fedora 27 Server veureu que hi ha una ISO de 2.3GB. No es recomana descarregar-la, és preferible usar netinst.

Exemple d'imatges de Fedora 27 [CloudImages](#):

Index of /pub/archive/fedora/linux/releases/27/Cloud			
Name	Last modified	Size	
 <a href="#">Parent Directory</a>		-	
 <a href="#">Fedora-Atomic-27-1.6.x86_64.qcow2</a>	2017-11-05 07:32	638M	
 <a href="#">Fedora-Atomic-27-1.6.x86_64.raw.xz</a>	2017-11-05 07:32	489M	
 <a href="#">Fedora-Atomic-Vagrant-27-1.6.x86_64.vagrant-libvirt.box</a>	2017-11-05 07:31	607M	
 <a href="#">Fedora-Atomic-Vagrant-27-1.6.x86_64.vagrant-virtualbox.box</a>	2017-11-05 07:29	621M	
 <a href="#">Fedora-Cloud-Base-27-1.6.x86_64.qcow2</a>	2017-11-05 07:29	222M	
 <a href="#">Fedora-Cloud-Base-27-1.6.x86_64.raw.xz</a>	2017-11-05 07:30	134M	
 <a href="#">Fedora-Cloud-Base-Vagrant-27-1.6.x86_64.vagrant-libvirt.box</a>	2017-11-05 07:32	243M	
 <a href="#">Fedora-Cloud-Base-Vagrant-27-1.6.x86_64.vagrant-virtualbox.box</a>	2017-11-05 07:32	253M	
 <a href="#">Fedora-CloudImages-27-1.6-x86_64-CHECKSUM</a>	2017-11-10 18:47	2.3K	

Podem observar que hi ha imatges amb format:

- ❑ **RAW**: imatges en cru, raw' que poden ser usades pels gestors de màquines virtuals com a imatges directes de disc (com si fossin fetes amb dd). Observeu que la imatge en realitat és [raw.xz](#), està comprimida. Comprimida ocupa 489M i un cop es descomprimeix ocupa 4B. És sobre la imatge raw un cop descomprimida que podem treballar.
- ❑ **Qcow2**: format estàndard d'imatges de disc per a màquines virtuals usat per libvirt, Virtualbox, etc.
- ❑ **Vagrant**: imatges per a Vagrant.

Les imatges Cloud de Fedora venen amb un password de root que es desconeix de manera que si s'engegen NO és possible iniciar sessió. Cal preparar les imatges abans per establir-hi el password de root.

S'utilitza la utilitat **virt-sysprep**, que permet, entre altres:

- Modificar el passwd de root de les imatges
- Crear i configurar usuaris a les imatges
- Crear i eliminar cclaues SSH.

Virt-sysprep can reset or unconfigure a virtual machine so that clones can be made from it. Steps in this process include removing SSH host keys, removing persistent network MAC configuration, and removing user accounts. Virt-sysprep can also customize a virtual machine, for instance by adding SSH keys, users or logos. Each step can be enabled or disabled as required.

Virt-sysprep modifies the guest or disk image in place. The guest must be shut down. If you want to preserve the existing contents of the guest, you must snapshot, copy or clone the disk first.

Modificar el passwd de root d'una imatge Cloud:

```
virt-sysprep -a Fedora-Cloud-Base-27-1.6.x86_64.qcow2 --root-password file:passwd.txt  
virt-sysprep -a Fedora-Cloud-Base-27-1.6.x86_64.raw --root-password file:passwd.txt  
  
cat passwd.txt  
root jupiter
```

Convertir formats d'imatge:

```
$ qemu-img convert -f raw -O qcow2 centos7.img centos7.qcow2  
$ qemu-img convert -f vmdk -O raw centos7.vmdk centos7.img  
$ qemu-img convert -f vmdk -O raw centos7.vmdk centos7.img  
$ VBoxManage clonehd ~/VirtualBox\ VMs/fedora21.vdi fedora21.img --format raw
```

```
qemu-system-x86_64 -m 512 -smp 4 --enable-kvm -boot order=dc -hda /root/howl/vm1.qcow2  
-cdrom /root/howl/SLES_SP1_x86_64.iso
```

---

# Arrencada del sistema

---

## Visió global

Primerament abans de descriure detalladament tot el procediment d'arrencada aclarim els conceptes de com s'organitzen aquests apunts i les explicacions del professor a classe. Hi ha moltes coses a explicar que englobarem en tres categories:

- ☐ Grub / Instal·lació de sistemes.
- ☐ Systemd: Arrancada del sistema.
- ☐ Systemctl / systemd: Gestió de serveis.

### **Grub / Instal·lació de sistemes**

En aquesta àrea de coneixement es treballa la configuració del grub, el gestor d'arrencada. Crear menús, submenús, establir opcions d'arrencada i posar passwords.

Per practicar 'facile e divertente' configuracions diferents de grub és xulo instal·lar diversos sistemes operatius a l'aula en particions diferents. Això permet no només configurar arrencades sinó també jugar a clonar particions, moure-les de lloc, etc.

Aquesta part és fàcil de practicar a l'aula però perillosa de practicar a casa...

### **Systemd: Arrancada del sistema**

En aquest apartat el concepte d'arrencada fa referència a tot allò que passa un cop s'ha seleccionat en el grub quin sistema operatiu es vol iniciar. Tot el procediment necessari per carregar el sistema operariu que podem observar quan premem una tecla i a la consola veiem el registre dels serveis que es van engegat.

Aquest apartat es pot practicar perfectament en el nostre ordinador sempre i quant tinguem una dosis apropiada de seny. Es tracta de veure què són els [targets](#), quins hi ha, com els podem configurar i fins i tot crear-ne de nous.

### **Systemctl / Systemd: Gestió de serveis**

Un cop el sistema ja està en marxa i estem en un shell podem modificar els serveis que hi ha actius aturant-los, pausant-los, engenant-ne de nous, etc. També examinar tota la configuració i la relació que hi ha entre ells. L'eina principal és la comanda de

systemd anomenada systemctl amb la qual gestionarem les [units](#), en especial les de [service](#).

L'arquitectura d'arrencada de systemd és molt simple però costa una mica de digerir al principi. Per això seguirem l'estratègia d'aprenentatge següent: primer veurem com són els targets d'arrencada; després la gestió de serveis amb systemctl; i finalment la part complicada que és com funciona tot plegat!

## Procediment d'arrencada

El procediment d'arrencada clàssic d'un sistema és més que conegut i usualment comporta els següents passos:

### Engegar

El host prement el botó d'engegada.

### POST Power On self Test

El test que realitza per verificar el hardware i per detectar-lo. Cada vegada que el sistema engega pot tenir elements afegits de nou i li cal fer-ne la detecció.

### Bios Basic Input Output System:

Hi ha el manual de com funciona el hardware perquè el sistema operatiu el sàpiga fer anar. Tothom sap fer anar un televisor (canviar de canal, modificar el volum, etc), però cada televisor té el seu manual de com es fan les coses en aquell televisor en concret. Aquest manual és la bios.

### Setup

A la bios hi ha emmagatzemada la configuració o setup de l'ordinador. Usualment es mantenen les dades amb una pila. Sovint s'accedeix a la configuració de la bios o setup amb les tecles [del](#) o [F2](#).

### Boot Options

A la bios es determina quin és el mecanisme d'arrencada a través d'una llista de preferències (fallback). Generalment en l'arrencada es pot accedir a un menú d'opcions d'arrencada amb les tecles [F8](#) o [F10](#) o directament es pot arrencar de xarxa prement [F12](#).

L'opció clàssica d'arrencada és carregar un sistema operatiu que hi ha al disc dur de l'ordinador. Aquest procés en realitat requereix de varis passos. Com a boot option s'ha d'indicar que es vol iniciar el sistema des del disc dur, indicant quin dels dispositius de disc es vol usar.

### MBR Master Boot Record



Quan s'ha seleccionat arrencar el sistema carregant-lo del disc dur s'ha indicat quin dels dispositius de disc cal usar. Aquest disc conté a l'inici del tot una zona anomenada MBR que resideix usualment al primer o primers sectors abans de les particions.

El MBR conté usualment una descripció del disc (una firma), la taula de particions i un **LOADER** o carregador. És un petit programet (de fet un salt) que 'estira' del procediment d'arrencada. Si no hi ha un loader s'intenta engegar la partició que estigui marcada com a activa.

## LOADER

En el MBR hi ha el mecanisme necessari per carregar a memòria un gestor d'arrencada, en el cas de GNU/Linux aquest és usualment el Grub. Aquest primer pas de càrrega s'anomena Stage1 i després es pot passar directament al Stage2 o a vegades cal un pas previ anomenat Stage1.5.

## Grub

Grand Unified Bootloader és el gestor d'arrencada fet per GNU que actualment s'utilitza en la majoria de sistemes GNU/Linux. En el grub es poden definir menús, submenús i opcions d'arrencada.

Un exemple clàssic de mení pot ser:

```
linux16 /boot/vmlinuz-4.18.19-100.fc27.x86_64 root=/dev/sda5 ro rhgb quiet LANG=en_US.UTF-8
initrd16 /boot/initramfs-4.18.19-100.fc27.x86_64.img
```

En una entrada de Grub d'un sistema operatiu GNU/Linux es defineix un **kernel** i un **initramfs** que corresponen al sistema operatiu a engegar.

## Kernel / initramfs

El Grub carrega el Kernel més el disc virtual de memòria amb els drivers necessaris per iniciar-lo. En carregar-se el Kernel li caldran drivers per el tecat, la targeta de so, de vídeo, de xarxa, etc de manera que necessita els drivers apropiats. Però els ordinadors tenen hardware diferent i per tant ha de seleccionar el driver apropiat entre infinitat de drivers. Si tots els drivers es carreguessin al kernel aquest seria molt gran i ocuparia molta memòria de manera que no és viable. Els drivers i una estructura inicial de sistema de fitxers que els continguin es troben en el initramfs.

En aquest pas es carrega el kernel, el nucli del sistema operatiu, però al kernel li falta tota la resta del sistema!

## Systemd

Un cop carregat el Kernel es hi genera el primer procés de d'espai d'usuari, de fora del Kernel. Aquest procés és el procés PID 1 (amb el número 1 de PID). I és l'encarregat de generar tots els altres processos, de 'estirar' tota la arrencada.

Hi ha dues famílies d'arrencada:

- ❑ Amb [init](#) i [runlevels](#).
- ❑ Amb [systemd](#) i [targets](#).

Els sistemes Red Hat / Centos / Fedora / Ubuntu utilitzen Systemd.

El systemd és el pare de tots els processos i té el PID 1. Ell estira de tots els altres processos i realitza la arrancada típica que veiem a pantalla fent scroll de serveis que es van carregant.

El systemd no és només el procés pare de tots els altres sinó que és tot un mecanisme dissenyat per gestionar l'arrencada i la gestió de serveis amb [systemctl](#) a través de [units](#).

L'aprenentatge de systemd és complex de manera que es recomana molt llegir, practicar, rellegir i re practicar. No s'entendrà a la primera!

Cal tenir en ment que systemd és qui gestiona tot el procés d'arrencada de serveis del sistema operatiu i tota la seva gestió amb systemctl, per tant tinguem present que són dues àrees de coneixements diferents però relacionades:

- ❑ Systemd / Systemctl per definir l'arrencada a través de targets. Quins serveis cal engegar.
- ❑ Systemd / Systemctl per gestionar en mode comanda com a administrador l'inici / aturada / etc de serveis.

Exemple de llistat de l'arbre de processos on es pot observar systemd amb el PID 1

```
# pstree -psa | head
systemd,1 --switched-root --system --deserialize 24
|-ModemManager,739
|  |-{ModemManager},746
|  `--{ModemManager},756
|-NetworkManager,825 --no-daemon
|  |-dhclient,1053 -d -q -sf /usr/libexec/nm-dhcp-helper -pf /var/run/dhclient-wlp2s0.pid -lf...
|  |-{NetworkManager},843
|  `--{NetworkManager},845
|-abrt-dbus,2002 -t133
|  |-{abrt-dbus},2005
```

## Systemd / runlevels

El mecanisme anterior a systemd d'arrencada dels serveis consistia a fer una llista exhaustiva dels serveis i en quin ordre calia engegar-los. Per iniciar una màquina en mode 3, multiusuari en consola de text (sense entorn gràfic) es configurava el llistat numerat de tots els serveis necessaris, que s'anaven engegant un a un.

És com dir als alumnes de la classe anem a fer una llista per ordre alfabètic per poder sortir a fer una cerveseta al bar del davant... Abans del runlevel-cerveseta s'ha de descriure la llista ordenada.

Amb systemd el planteig és totalment diferent, no es determina l'ordre de res ni s'engegen les coses una després de l'altra (això ralentitzava molt l'arrencada). Systemd determina l'objectiu i engega tots els serveis de cop, tots. Hi ha serveis que depenen d'altres i no es poden engegar fins que aquests altres no ho han fet, o no han finalitzat o no han fet tal tasca. És igual. Systemd hoi engega tot i els serveis ja s'aniran espavilat.

Així els serveis que no depenen de cap altre servei es poden engegar de bon principi sense esperar-se a res. Els que tenen dependències es posen en cua, a esperar que finalitzi allò que esperen. A la que poden s'executen.

Observeu la diferència. Amb runlevels calia fer la llista i executar un a un els serveis, el que li tocava el torn 65 no es podia executar fins que no s'havien engegat els 64 anteriors. Amb systemd s'engeguen tots! Cada servei avança fins on pot i s'espera a que es compleixin els seus prerequisits per continuar avançant.

Amb aquest mecanisme de paral·lelisme l'arrencada és més ràpida.

Amb systemd el profe diu, anem per ordre alfabètic a fer una cerveseta, però no mana res més, els alumnes ja s'ordenen sols, el primer de la llista (o els primers) ja van tirant (així es poden demanar olivetes i patatetes...) sense esperar a que tota la fila estigui ordenada.

## Targets d'arrencada

### Targets / runlevels

El sistema tradicional d'arrencada de GNU/Linux era basat en el procés init i en runlevels. Actualment treballem amb systemd i targets, però la nomenclatura usada durant tants anys amb runlevels encara persisteix i és important identificar-la.

Els targets actuals són:

- ☐ emergency.target
- ☐ rescue.target
- ☐ multi-user.target
- ☐ graphical.target
  
- ☐ default.target

La correspondència entre els antics **runlevels** i els **targets** actuals és:

- 0 poweroff.target
- 1 single rescue rescue.target

- 2 multi sense xarxa
- 3 multi-user.target
- 4 custom.target
- 5 graphical.target
- 6 reboot.target

Podeu consultar la taula amb man runlevel

Table 1. Mapping between runlevels and systemd targets

Runlevel	Target
0	poweroff.target
1	rescue.target
2, 3, 4	multi-user.target
5	graphical.target
6	reboot.target

Una descripció breu dels targets és la següent:

- [poweroff.target](#) apaga la màquina, correspon a l'antic runlevel 0.
- [recue.target](#) permet iniciar la màquina en mode de rescat (antigament directament en mode root monousuari). Pràcticament no es carrega cap altre servei. Correspon a l'antic runlevel 1.
- [multi-user.target](#) permet iniciar la màquina en mode consola amb tots els serveis (definits per defecte) activats, excepte l'entorn gràfic. Correspon a l'antic mode 3.
- [graphical.target](#) és l'arrencada completa de la màquina a mode gràfic, incorpora els mateixos serveis que multi-user.target (per defecte) més l'entorn gràfic. Correspon a l'antic mode 5. Actualment és el target per defecte a les aules i segurament a casa vostra.
- [reboot.target](#) fa reiniciar la màquina. Correspon a l'antic runlevel 6.

Per defecte el systema a l'aula arrenca en mode graphical.target i carrega tots els serveis que hi ha definits per aquest mode, podem veure la llista de serveis que va carregant prememtn una tecla mentre es fa l'arrencada del sistema i també amb l'ordre systemctl.

## Target per defecte

A quin target hem iniciat el sistema? Com ho podem verificar? Quin són els serveis associats a aquest target? Podem canviar el target per defecte? Anem a respondre a totes aquestes preguntes.

Primerament comentem el concepte de target. Amb systemd els antics nivells d'arrencada (runlevels) es transformen en targets, objectius. El planteig és que hi ha un objectiu, aconseguir arribar a un determinat 'destí'. Si mai heu jugat a videojocs... us sonarà que per arribar a passar determinat nivell has d'haver aconseguit prèviament certes coses. La idea de systemd és definir on volem arribar no els passos seguir per arribar-hi sinó què cal tenir.

Consultar / establir el target per defecte:

- `systemctl get-default`
- `systemctl set-default <target-name>`
- `runlevel`
- `ln -s` (manualment)
- `Systemctl is-active <target-name>`
- `Systemctl is-enabled <target / servei>`

Consultar / establir el target per defecte

```
# systemctl get-default
graphical.target

# systemctl set-default multi-user.target
Removed /etc/systemd/system/default.target.
Created symlink /etc/systemd/system/default.target → /usr/lib/systemd/system/multi-user.target.

# systemctl get-default
multi-user.target

# systemctl set-default graphical.target
Removed /etc/systemd/system/default.target.
Created symlink /etc/systemd/system/default.target → /usr/lib/systemd/system/graphical.target.

# systemctl get-default
graphical.target
# runlevel
N 5

# ls -l /etc/systemd/system/default.target
lrwxrwxrwx 1 root root 40 Apr 21 20:47 /etc/systemd/system/default.target -> /usr/lib/systemd/system/graphical.target
```

Observeu que establir el `default.target` és simplement crear un symbolic link a [/etc/systemd/system](#) amb aquest nom que apunti al target desitjat dins de [/usr/lib/systemd/system](#).

Tota la gestió de targets es realitza a través de symlinks, es pot fer manualment amb l'ordre `ln -s` però és perillosa... a l'escola ho fem però a casa es recomanable ser prudent.

```
# systemctl is-active multi-user.target
active
#
# systemctl is-active default.target
inactive

# systemctl is-active rescue.target
inactive
```

```
# systemctl is-enabled default.target
enabled
# systemctl is-enabled multi-user.target
static
# systemctl is-enabled rescue.target
disabled
```

## Iniciar el sistema a un target diferent

Hem vist com indicar quin és el target per defecte però podem canviar d'un target a un altre? Un cop iniciada la màquina per exemple en mode gràfic podem passar a només mode text? O només mode rescue? Podem engegar el sistema a un mode diferent del mode per defecte?

Els mecanismes per modificar el target d'arrancada són:

- ❑ En el grub passar-li al kernel el target a arrancar com a opció.
- ❑ Modificar el valor de `default.target`
- ❑ Usar `systemctl isolate`

### Paràmetre del kernel al Grub

Quan s'engega el sistema en el menú del grub es poden modificar els valors de les opcions que es passen al kernel. Al grub podem premer `e` i editar l'entrada del grub. Les opcions es poden passar amb la nova sintaxi de systemd però molt sovint s'utilitzen els antics valors de `runlevel`.

- ❑ **runlevel**: afegir al final de la línia del kernel un espai i un número corresponent al `runlevel` al que es vol iniciar el sistema. Per exemple posar un 1 engega en mode `rescue`, posar un 3 en mode text i un 5 en mode gràfic.

```
linux16 /boot/vmlinuz-4.18.19-100.fc27.x86_64 root=/dev/sda5 ro rhgb quiet LANG=en_US.UTF-8 1
```

- ❑ `systemd.unit=<target-name>` indicar aquesta opció en les opcions del kernel, escrivint tot sencer el nom del target, per exemple:

*`systemd.unit=multi-user.target`*

```
linux16 /boot/vmlinuz-4.18.19-100.fc27.x86_64 root=/dev/sda5 ro rhgb quiet LANG=en_US.UTF-8  
systemd.unit=rescue.target
```

### Modificar el valor del `default.target`

Tal i com s'ha vist en l'apartat target per defecte es pot modificar el target al que arrenca el sistema per defecte amb les ordres `systemctl set-default` i `systemctl get-default`.

### Amb l'ordre `systemctl isolate`

Un cop un sistema ja està engegat i a un target concret es pot anar a un altre target amb l'ordre:

`systemctl isolate <target-name>`

Així per exemple si hem engegat en mode consola (mode multi-user.target) i volem anar a mode gràfic podem fer `systemctl isolate graphical.target`.

Les antigues ordres de init per passar d'un runlevel a un altre encara funcionen (a vegades): *init runlevel / tellinit runlevel*.

**Pràctica proposada:**

1. En tots els casos verificar el target actual. Llisteu els processos i els serveis.
2. Configurar el sistema establint `default.target` a mode multi-user.target.
3. Manualment canviar a `graphical.target` amb `isolate`.
4. Reiniciar el sistema i al grub establir l'opció de iniciar en mode `emergency.target`.
5. Reiniciar i indicar al grub l'opció del kernel per iniciar a `rescue.target`.
6. Canviar de target amb l'ordre `isolate` activant multi-user.target.
7. Restablir per defecte `graphical.target` i reiniciar el sistema.
8. Amb `isolate` indicar que es vol accedir al target `poweroff.target`.

## Emergency / Salvar els mobles! `init=/bin/bash`

En apartats anteriors s'ha indicat quins són els targets que utilitza systemd. Anem-los a repassar.

`Graphical.target` i `multi-user.target` són bastant clars d'entendre. Arrancada en entorn gràfic o en entorn de consola, tots dos casos amb els mateixos serveis (generalment).

`Rescue.target` pel nom també s'enten però pot portar a algunes confusions. Inicia la màquina carregant pràcticament cap servei, molt pocs, de manera que permet recuperar-se de problemes d'arrencada quan un dels serveis penja l'arrencada. Es diferencia del runlevel 1 en que aquí no s'entra directament en mode root, cal posar el password de root per poder iniciar sessió en mode `rescue.target`.

`Emergency.target` és un mode d'emergència en que simplement s'engega systemd.

### Salvar els mobles! `init=/bin/bash`

En cas de catàstrofe total es pot usar el [trick](#) d'escriure a les opcions del kernel del Grub el següent:

```
init=/bin/bash rw
```

Afegint aquest text li diem que el primer programa que ha de carregar el kernel no és systemd sinó `/bin/bash`. Només es carregarà això, serà un sistema totalment despulat de

processos. El paràmetre `rw` és perquè munti el sistema de fitxers en mode *read/write*, sinó el munta en mode *read only*.

Aquest trick és molt útil si per exemple els alumnes han espatllat el `/etc/fstab` en editar-lo matusserament...

#### Pràctica proposada:

9. Iniciar el sistema en mode `emergency.target`. Llistar els processos, l'arbre de processos, els units i les dependències. Cal indicar el password de root? Hi ha múltiples sessions de consola?
10. Iniciar el sistema en mode `init=/bin/bash`. Llistar els processos, l'arbre de processos, els units i les dependències. Cal indicar el password de root? Hi ha múltiples sessions de consola?

## Llistar units i dependències

El funcionament de `systemd` amb els `targets` i els `units` és molt senzill però també complicat d'entendre de bones a primeres, de manera que abans d'exposar a fons els `targets` i la llista de serveis associats ([wants](#)) hem d'agafar pràctica treballant amb `systemd` i `systemctl`.

L'explicació més a fons queda per a més endavant en aquest [HowTo](#), de moment ens conformem a aprendre a llistar units i dependències.

`Systemd` processa units que poden ser de diversos tipus (n'hi ha varis). Per simplificar de moment treballem amb [targets](#) i [services](#). `Targets` correspon als antics `runlevels` o nivells d'arrencada del sistema. `Service` són serveis del sistema com per exemple `httpd`, `sshd`, `gpm`, `libvirt`, etc.

Els conceptes i les ordres a practicar són:

- ☐ `systemctl list-units`
- ☐ `systemctl list-dependencies`
- ☐ `systemctl list-dependencies <basic.target> | grep ".target$"`
- ☐ `systemctl list-units --type=<tipus>`

Llistat de totes les [units](#) (de tots els tipus):

```
# systemctl list-units
UNIT                                LOAD    ACTIVE SUB    DESCRIPTION
proc-sys-fs-binfmt_misc.automount  loaded active waiting Arbitrary Executable File Formats F
sys-devices-pci0000:00-0000:00:02.0-drm-card0-card0\lx2deDP\lx2d1-intel_backlight.device loaded active plugged /sys/devices/pci0000:00/0000:00:02.
sys-devices-pci0000:00-0000:00:14.0-usb1-1\lx2d8-1\lx2d8:1.0-bluetooth-hci0.device loaded active plugged /sys/devices/pci0000:00/0000:00:14.
sys-devices-pci0000:00-0000:00:17.0-ata2-host1-target1:0-0-1:0-0-0-block-sda-sda1.device loaded active plugged ST500LM021-1KJ152 Reservado_para_el
sys-devices-pci0000:00-0000:00:17.0-ata2-host1-target1:0-0-1:0-0-0-block-sda-sda2.device loaded active plugged ST500LM021-1KJ152 SISTEMA
sys-devices-pci0000:00-0000:00:17.0-ata2-host1-target1:0-0-1:0-0-0-block-sda-sda3.device loaded active plugged ST500LM021-1KJ152 DADES
sys-devices-pci0000:00-0000:00:17.0-ata2-host1-target1:0-0-1:0-0-0-block-sda-sda4.device loaded active plugged ST500LM021-1KJ152 4
```



```

sys-devices-pci0000:00-0000:00:17.0-ata2-host1-target1:0:0-1:0:0:0-block-sda-sda5.device loaded active plugged ST500LM021-1KJ152 5
sys-devices-pci0000:00-0000:00:17.0-ata2-host1-target1:0:0-1:0:0:0-block-sda-sda6.device loaded active plugged ST500LM021-1KJ152 6
sys-devices-pci0000:00-0000:00:17.0-ata2-host1-target1:0:0-1:0:0:0-block-sda-sda7.device loaded active plugged ST500LM021-1KJ152 f27
sys-devices-pci0000:00-0000:00:17.0-ata2-host1-target1:0:0-1:0:0:0-block-sda.device loaded active plugged ST500LM021-1KJ152
sys-devices-pci0000:00-0000:00:1c.4-0000:02:00.0-net-wlp2s0.device loaded active plugged Wireless 8265 / 8275
sys-devices-pci0000:00-0000:00:1c.5-0000:03:00.0-net-enp3s0.device loaded active plugged RTL8111/8168/8411 PCI Express Gigab
sys-devices-pci0000:00-0000:00:1f.3-sound-card0.device loaded active plugged /sys/devices/pci0000:00/0000:00:1f.
sys-devices-platform-serial8250-tty-ttyS0.device loaded active plugged /sys/devices/platform/serial8250/tty
.....

```

LOAD = Reflects whether the unit definition was properly loaded.

ACTIVE = The high-level unit activation state, i.e. generalization of SUB.

SUB = The low-level unit activation state, values depend on unit type.

194 loaded units listed. Pass --all to see loaded but inactive units, too.

To show all installed unit files use 'systemctl list-unit-files'.

## Llistat de totes les unitats de tipus **service**, serveis del sistema:

### # systemctl list-units --type=service

UNIT	LOAD	ACTIVE	SUB	DESCRIPTION
● abrt.service	loaded failed failed			ABRT Automated Bug Reporting Tool
accounts-daemon.service	loaded active running			Accounts Service
alsa-state.service	loaded active running			Manage Sound Card State (restore and store)
auditd.service	loaded active running			Security Auditing Service
avahi-daemon.service	loaded active running			Avahi mDNS/DNS-SD Stack
bluetooth.service	loaded active running			Bluetooth service
chronyd.service	loaded active running			NTP client/server
colord.service	loaded active running			Manage, Install and Generate Color Profiles
crond.service	loaded active running			Command Scheduler
cups.service	loaded active running			CUPS Scheduler
dbus.service	loaded active running			D-Bus System Message Bus
dracut-shutdown.service	loaded active exited			Restore /run/initramfs on shutdown
fedora-import-state.service	loaded active exited			Import network configuration from initramfs
fedora-readonly.service	loaded active exited			Configure read-only root support
firewalld.service	loaded active running			firewalld - dynamic firewall daemon
fwupd.service	loaded active running			Firmware update daemon
gdm.service	loaded active running			GNOME Display Manager
geoclue.service	loaded active running			Location Lookup Service
gssproxy.service	loaded active running			GSSAPI Proxy Daemon
iscsi-shutdown.service	loaded active exited			Logout off all iSCSI sessions on shutdown
kmod-static-nodes.service	loaded active exited			Create list of required static device nodes for the current kernel
libvirt-guests.service	loaded active exited			Suspend/Resume Running libvirt Guests
libvirt.service	loaded active running			Virtualization daemon

LOAD = Reflects whether the unit definition was properly loaded.

ACTIVE = The high-level unit activation state, i.e. generalization of SUB.

SUB = The low-level unit activation state, values depend on unit type.

63 loaded units listed. Pass --all to see loaded but inactive units, too.

To show all installed unit files use 'systemctl list-unit-files'.

### # systemctl list-units --type=target

UNIT	LOAD	ACTIVE	SUB	DESCRIPTION
basic.target	loaded active active			Basic System
bluetooth.target	loaded active active			Bluetooth
cryptsetup.target	loaded active active			Local Encrypted Volumes
getty.target	loaded active active			Login Prompts
graphical.target	loaded active active			Graphical Interface
local-fs-pre.target	loaded active active			Local File Systems (Pre)
local-fs.target	loaded active active			Local File Systems
multi-user.target	loaded active active			Multi-User System
network-online.target	loaded active active			Network is Online
network-pre.target	loaded active active			Network (Pre)
network.target	loaded active active			Network
nfs-client.target	loaded active active			NFS client services
nss-user-lookup.target	loaded active active			User and Group Name Lookups
paths.target	loaded active active			Paths
remote-fs-pre.target	loaded active active			Remote File Systems (Pre)
remote-fs.target	loaded active active			Remote File Systems
rpc_pipefs.target	loaded active active			rpc_pipefs.target
slices.target	loaded active active			Slices
sockets.target	loaded active active			Sockets
sound.target	loaded active active			Sound Card
swap.target	loaded active active			Swap
sysinit.target	loaded active active			System Initialization
timers.target	loaded active active			Timers
virt-guest-shutdown.target	loaded active active			Libvirt guests shutdown

LOAD = Reflects whether the unit definition was properly loaded.

ACTIVE = The high-level unit activation state, i.e. generalization of SUB.

SUB = The low-level unit activation state, values depend on unit type.

24 loaded units listed. Pass --all to see loaded but inactive units, too.

To show all installed unit files use 'systemctl list-unit-files'.

Llistat de les dependències. Podem observar cada target quines dependències té:

## # systemctl list-dependencies

### default.target

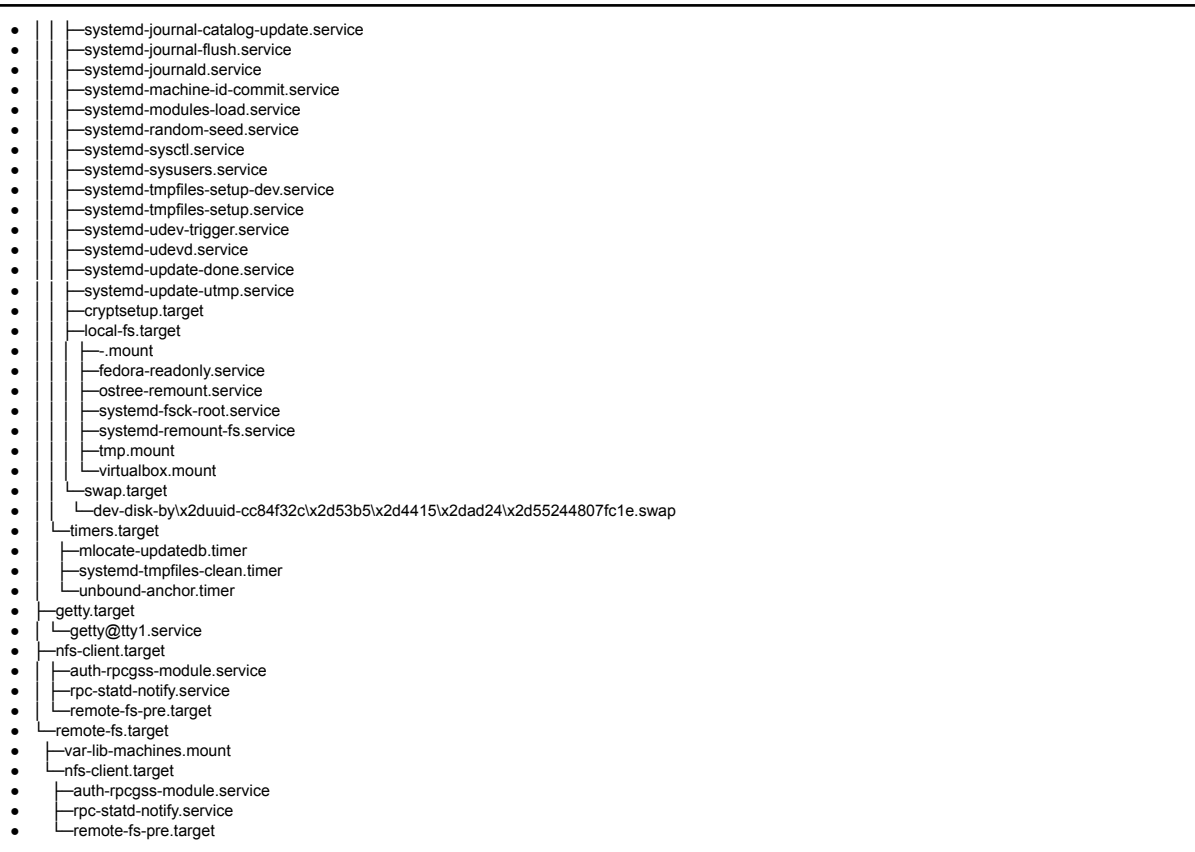
- accounts-daemon.service
- gdm.service
- rtkit-daemon.service
- switcheroo-control.service
- systemd-update-utmp-runlevel.service
- udisks2.service

### multi-user.target

- abrt-journal-core.service
- abrt-oops.service
- abrt-vmcore.service
- abrt-xorg.service
- abrt.service
- auditd.service
- avahi-daemon.service
- chronyd.service
- crond.service
- cups.path
- dbus.service
- dbxtool.service
- firewall.service
- mclog.service
- mdmonitor.service
- ModemManager.service
- NetworkManager.service
- plymouth-quit-wait.service
- plymouth-quit.service
- rngd.service
- sssd.service
- systemd-ask-password-wall.path
- systemd-logind.service
- systemd-update-utmp-runlevel.service
- systemd-user-sessions.service
- vboxautostart.service.service
- vboxballoonctrl.service.service
- vboxdrv.service
- vboxweb.service.service
- vmttoolsd.service

### basic.target

- .mount
- dnf-makecache.timer
- selinux-autorelabel-mark.service
- tmp.mount
- paths.target
- slices.target
- .slice
- system.slice
- sockets.target
- avahi-daemon.socket
- cups.socket
- dbus.socket
- dm-event.socket
- iscsid.socket
- iscsiui.socket
- sssd-kcm.socket
- sssd-secrets.socket
- systemd-coredump.socket
- systemd-initctl.socket
- systemd-journal-audit.socket
- systemd-journal-dev-log.socket
- systemd-journal.socket
- systemd-udev-control.socket
- systemd-udev-kernel.socket
- sysinit.target
- dev-hugepages.mount
- dev-mqueue.mount
- dracut-shutdown.service
- fedora-import-state.service
- iscsi.service
- kmod-static-nodes.service
- ldconfig.service
- lvm2-lvmetad.socket
- lvm2-lvmpolld.socket
- lvm2-monitor.service
- multipathd.service
- plymouth-read-write.service
- plymouth-start.service
- proc-sys-fs-binfmt\_misc.automount
- sys-fs-fuse-connections.mount
- sys-kernel-config.mount
- sys-kernel-debug.mount
- systemd-ask-password-console.path
- systemd-binfmt.service
- systemd-firstboot.service
- systemd-hwdb-update.service



Llistat de les dependències filtrant només targets. Ens permet veure quins targets depenen de quins altres:



Observem que default.target (que en aquest cas és graphical.target) requereix de multi-user.target. Aquest ehem vist abans que necessita de molts serveis (els típics que engeguem...) però també dels targets prèvis basic.target, getty.target, etc.

Imaginem que pugem una muntanya i tenim l'objectiu de everest.target, per aconseguir-ho hem de passar per campbase.target i a més a més engegar tot de serveis individuals.service com oxigen.service o cerveseta.service.

En la requadre següent podem veure un extracte dels dos anteriors on podem visualitzar clarament que per tenir el cromo de default.target es passa per multi-user.target i aquest passa per basic.target i aquest passa per sysinit.target que requereix de tres targets previs:

```
# systemctl list-dependencies | grep .target
default.target
• └─multi-user.target
•   └─basic.target
•     └─sysinit.target
•       └─cryptsetup.target
•       └─local-fs.target
•       └─swap.target
```

En resum, amb els runlevels es definia pas a pas que fer per arribar a un destí, amb sysinit es defineixen els requeriments que calen per arribar el destí, però no els passos.

## Eines per monitorar l'arrancada

Podeu consultar les eines per monitorar l'arrancada a l'apartat:

[Gràfics de l'arrencada i de les relacions de les unitats](#)

---

# Serveis / systemctl

---

## Gestió bàsica dels serveis

Els sistemes operatius GNU/Linux que utilitzen systemd gestionen els serveis amb **systemctl**. Anem a veure en aquest apartat les ordres necessàries per activar / desactivar serveis puntualment o per sempre i altres opcions de gestió.

Primerament hem d'entendre el concepte de servei. Un servei és un programa (o conjunt de programes) que realitzen una tasca generalment en background i que també s'anomenen usualment dimonis daemon. En serveis d'arquitectura client/servidor el servei s'engega i es queda escoltant les peticions que li arriben (quan li arriben) dels clients, les atèn, serveix, i es torna a quedar escoltant.

Són exemples típics de serveis el servidor web httpd, el servidor de SSH sshd, el servei de poder utilitzar el punter del ratolí en mode text gpm, etc.

Les principals característiques a gestionar són:

- ☐ Examinar l'estatus d'un servei
- ☐ Activar / Desactivar serveis
- ☐ Enable / Disable de serveis
- ☐ Restart / Reload
- ☐ Mask / Unmask Enmascarar serveis
- ☐ Identificar el fitxer de servei / examinar el paquet
- ☐ Identificar l'enable del servei

## Status d'un servei

Podem consultar l'estat d'un servei amb les ordres:

- ☐ `systemctl status <nomservei>`
- ☐ `systemctl is-active <nomservei>`
- ☐ `systemctl is-enabled <nomservei>`

Anem a veure alguns exemples. En el primer exemple consultem l'estat del servei https (el servei web apache) i podem observar que no està actiu ni enabled.

```
# systemctl status httpd
● httpd.service - The Apache HTTP Server
  Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
  Active: inactive (dead)
  Docs: man:httpd.service(8)
```

```
# systemctl is-active httpd
inactive

# systemctl is-enabled httpd
disabled
```

En aquest exemple podem observar que el servei sshd (el servidor SSH) si que està actiu, però no enabled. Observem també que el servei té obert el port 22 per on escolta les connexions entrants dels clients.

```
# systemctl status sshd
● sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; disabled; vendor preset: disabled)
   Active: active (running) since Wed 2020-04-22 09:24:35 CEST; 26s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
    Main PID: 3579 (sshd)
      Tasks: 1 (limit: 4915)
     Memory: 2.2M
        CPU: 22ms
    CGroup: /system.slice/sshd.service
            └─3579 /usr/sbin/sshd -D
-oCiphers=aes256-gcm@openssh.com,chacha20-poly1305@openssh.com,aes256-ctr,aes256-cbc,aes128-gcm@openssh.com,aes128

Apr 22 09:24:34 lenovo systemd[1]: Starting OpenSSH server daemon...
Apr 22 09:24:35 lenovo sshd[3579]: Server listening on 0.0.0.0 port 22.
Apr 22 09:24:35 lenovo sshd[3579]: Server listening on :: port 22.
Apr 22 09:24:35 lenovo systemd[1]: Started OpenSSH server daemon.

# systemctl is-active sshd
active

# systemctl is-enabled sshd
disabled

# nmap localhost
Starting Nmap 7.60 ( https://nmap.org ) at 2020-04-22 09:31 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000029s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 998 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
```

## Activar / Desactivar un servei

Les ordres amb les que gestionarem l'activació i aturada de serveis són:

- ☐ systemctl start <nomservei>
- ☐ systemctl stop <nomservei>
- ☐ systemctl is-active <nomservei>
- ☐ systemctl is-enabled <nomservei>

## Start

Per poder activar un servei primerament cal tenir-lo instal·lat al sistema. Segurament el servei ssh ja el teniu instal·lat però el més usual és que serveis com per exemple httpd, xinetd o gpm no ho estiguin.

Per instal·lar un servei usem l'eina dnf. Anem a instal·lar els serveis httpd, gpm i xinetd.

```
# dnf -y install httpd xinetd gpm
```

Els serveis quan s'instal·len no s'activen automàticament, cal fer-ho manualment amb l'ordre systemctl. Amb dnf simplement s'ha baixat els paquets del repositori de Fedora i els ha instal·lat, és feina de l'administrador decidir si vol activar puntualment el servei (systemctl start) o configurar-lo per activar-se en iniciar el sistema (systemctl enable).

Anem a engegar el servei httpd i verificar que funciona. Observem que el servei està actiu però no enabled i que ara el host té obert el port 80 per on escolta connexions clients que li demanen accedir al servei de pàgines web:

```
# systemctl start httpd

# systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
   Active: active (running) since Wed 2020-04-22 09:41:58 CEST; 39s ago
     Docs: man:httpd.service(8)
  Main PID: 4340 (httpd)
    Status: "Running, listening on: port 80"
     Tasks: 213 (limit: 4915)
    Memory: 25.5M
       CPU: 292ms
    CGroup: /system.slice/httpd.service
           └─4340 /usr/sbin/httpd -DFOREGROUND
             └─4344 /usr/sbin/httpd -DFOREGROUND
               └─4346 /usr/sbin/httpd -DFOREGROUND
                 └─4347 /usr/sbin/httpd -DFOREGROUND
                   └─4348 /usr/sbin/httpd -DFOREGROUND

Apr 22 09:41:37 lenovo systemd[1]: Starting The Apache HTTP Server...
Apr 22 09:41:53 lenovo httpd[4340]: AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using fe80::37dc:7fba:6694:
Apr 22 09:41:58 lenovo httpd[4340]: Server configured, listening on: port 80
Apr 22 09:41:58 lenovo systemd[1]: Started The Apache HTTP Server.

# systemctl is-active httpd
active

# systemctl is-enabled httpd
disabled

# nmap localhost
Starting Nmap 7.60 ( https://nmap.org ) at 2020-04-22 09:43 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000028s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 997 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
```

Si ens fa il·lusió veure el servidor web en funcionament podem crear una simple pàgina web i connectar amb telnet o amb el navegador per veure-la.

Crear la pàgina web:

```
# echo "Hola sóc la teva pàgina web!" > /var/www/html/index.html
```

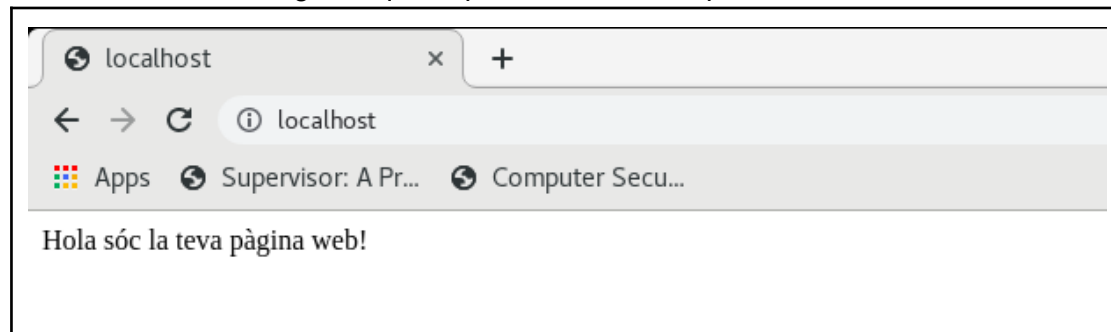
Verificar en consola amb telnet que hi podem accedir:

```
# telnet localhost 80
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 22 Apr 2020 07:49:48 GMT
Server: Apache/2.4.34 (Fedora)
Last-Modified: Wed, 22 Apr 2020 07:49:31 GMT
ETag: "1f-5a3dc5f5e6fe"
Accept-Ranges: bytes
Content-Length: 31
Connection: close
Content-Type: text/html; charset=UTF-8

Hola sóc la teva pàgina web!
Connection closed by foreign host.
```

Verificar amb un navegador que hi podem accedir: <http://localhost>



**Activar un servei:** activar un servei amb `systemctl start` activa puntualment el servei, el servei estarà engegat a partir d'ara però no per sempre. Quan la màquina es reiniciï el servei tornarà a estar apagat. Amb `systemctl start` li diem que s'engegi el servei ara però no li diem que estigui engegat sempre que s'engegi la màquina. Podeu verificar-ho reiniciant l'ordinador i veureu com el servei `httpd` torna a estar apagat.

## Stop

Per aturar un servei el més lògic és que estigui prèviament engegat tot i que el sistema no es queixa. Anem a veure exemples per aturar serveis amb l'ordre `systemctl stop`.



Observem que li manem aturar el servei xinetd que de fet no estava engegat. Després aturem el servei httpd i ho verifiquem amb status i is-active. També podem observar que el port 80 ja no està obert.

```
# systemctl stop xinetd

# systemctl stop httpd

# systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
   Active: inactive (dead)
     Docs: man:httpd.service(8)

Apr 22 09:41:37 lenovo systemd[1]: Starting The Apache HTTP Server...
Apr 22 09:41:53 lenovo httpd[4340]: AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using fe80::37dc:7fba:6694:
Apr 22 09:41:58 lenovo httpd[4340]: Server configured, listening on: port 80
Apr 22 09:41:58 lenovo systemd[1]: Started The Apache HTTP Server.
Apr 22 09:48:03 lenovo systemd[1]: Reloading The Apache HTTP Server.
Apr 22 09:48:09 lenovo httpd[4672]: AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using fe80::37dc:7fba:6694:
Apr 22 09:48:09 lenovo systemd[1]: Reloaded The Apache HTTP Server.
Apr 22 09:48:09 lenovo httpd[4340]: Server configured, listening on: port 80
Apr 22 09:59:22 lenovo systemd[1]: Stopping The Apache HTTP Server...
Apr 22 09:59:23 lenovo systemd[1]: Stopped The Apache HTTP Server.

# systemctl is-active httpd
inactive

# systemctl is-enabled httpd
disabled

# nmap localhost
Starting Nmap 7.60 ( https://nmap.org ) at 2020-04-22 10:01 CEST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000028s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 998 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
```

**Aturar un servei:** aturar un servei l'atura puntualment. No l'atura per sempre. Si un servei estava configurat per activar-se sempre que s'inicia l'ordinador amb la ordre systemctl stop aturem el servei puntualment, però la propera vegada que es reiniciï el host el servei tornarà a estar activat.

Podem comprovar-ho aturant un servei que estigui enabled (per exemple sshd) i reiniciant l'ordinador. Veureu que en reiniciar torna a estar engegat.

## Enable / Disable d'un servei

Hem vist com instal·lar serveis i com activar-los i desactivar-los puntualment, però com o podem fer per establir que el servei estigui sempre actiu en iniciar el sistema? O a l'inrevés, que estigui sempre aturat en iniciar el sistema?

Amb les ordres:

```
❏ systemctl enable <nomservei>
```

❑ `systemctl disable <nomservei>`

Amb `systemctl enable` s'indica al sistema que volem que el servei estigui sempre engegat en arrencar el sistema operatiu. Atenció, l'ordre `systemctl enable` no engega el servei ara, defineix que a partir d'ara, quan el sistema s'engegui el servei ha d'estar engegat (però si ara estava aturat continua ara aturat fins al proper reinici).

Amb `systemctl disable` s'indica al sistema que el servei estigui sempre aturat en engegar de nou el sistema operatiu. Atenció, no l'atura ara sinó que defineix que a partir d'ara quan el sistema s'engegui el servei ha d'estar aturat (però si ara està engegat continuarà engegat fins al nou reinici).

En el següent exemple anem a configurar a fer el següent:

- Engegar el servei `httpd` ara però configurar-lo com a `disabled`, de manera que en reiniciar el sistema no s'engegarà.
- Aturar el servei `sshd` ara, però configurar-lo com a `enabled`, de manera que en reiniciar el sistema s'engegarà.

```
# systemctl start httpd

# systemctl disable httpd

# systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
   Active: active (running) since Wed 2020-04-22 10:18:10 CEST; 57s ago
     Docs: man:httpd.service(8)
  Main PID: 5544 (httpd)
    Status: "Running, listening on: port 80"
     Tasks: 213 (limit: 4915)
    Memory: 23.6M
       CPU: 291ms
    CGroup: /system.slice/httpd.service
           └─5544 /usr/sbin/httpd -DFOREGROUND
             └─5545 /usr/sbin/httpd -DFOREGROUND
               └─5547 /usr/sbin/httpd -DFOREGROUND
                 └─5548 /usr/sbin/httpd -DFOREGROUND
                   └─5553 /usr/sbin/httpd -DFOREGROUND

Apr 22 10:17:59 lenovo systemd[1]: Starting The Apache HTTP Server...
Apr 22 10:18:10 lenovo httpd[5544]: AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using fe80::37dc:7fba:6694:
Apr 22 10:18:10 lenovo httpd[5544]: Server configured, listening on: port 80
Apr 22 10:18:10 lenovo systemd[1]: Started The Apache HTTP Server.

# systemctl is-active httpd
active

# systemctl is-enabled httpd
disabled
```

```
# systemctl stop sshd

# systemctl enable sshd
Created symlink /etc/systemd/system/multi-user.target.wants/ssh.service →
/usr/lib/systemd/system/ssh.service.

# systemctl status sshd
```

```
● sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: disabled)
   Active: inactive (dead)
     Docs: man:sshd(8)
           man:sshd_config(5)

Apr 22 09:24:34 lenovo systemd[1]: Starting OpenSSH server daemon...
Apr 22 09:24:35 lenovo sshd[3579]: Server listening on 0.0.0.0 port 22.
Apr 22 09:24:35 lenovo sshd[3579]: Server listening on :: port 22.
Apr 22 09:24:35 lenovo systemd[1]: Started OpenSSH server daemon.
Apr 22 10:21:01 lenovo systemd[1]: Stopping OpenSSH server daemon...
Apr 22 10:21:01 lenovo sshd[3579]: Received signal 15; terminating.
Apr 22 10:21:01 lenovo systemd[1]: Stopped OpenSSH server daemon.

# systemctl is-active sshd
inactive

# systemctl is-enabled sshd
enabled
```

En aquest moment el servei sshd està apagat i el servei httpd està engegat, però si es reinicia l'ordinador veurem que per defecte el servei httpd està apagat i el servei sshd està engegat.

## Restart / Reload d'un servei

Sovint hem de reiniciar un servei perquè funciona malament o perquè hem fet canvis en la configuració i es necessita recarregar. Usualment hi ha dues opcions per fer això que són lleugerament diferents:

**Restart** molt sovint equival a fer un stop i tot seguit un start. Literalment és apagar i tornar a engegar el servei.

**Reload** en canvi significa recarregar el servei però sense aturar-lo. Sovint això es fa enviant el senyal SIGHUP (senyal 1) al dimoni del servei.

## Mask / Unmask Enmascarar un servei

Hem vist com activar / desactivar / enable / disable / restart / reload de serveis. Però com a administradors del sistema podem bloquejar un servei de manera que no es pugui activar?

Si, per fer-ho usem les ordres:

- ❑ `systemctl mask <nomservei>`
- ❑ `systemctl unmask <nomservei>`

En aquest exemple tenim el servei gpm que està stop i disabled. L'enmascarem amb la ordre **systemctl mask** i podem observar que ara quan demanem si està enabled ens contesta que està **masked**. Si mirem d'engegar-lo ens diu que no es pot.

```
# systemctl mask gpm
Created symlink /etc/systemd/system/gpm.service → /dev/null.

# systemctl status gpm
• gpm.service
  Loaded: masked (/dev/null; masked)
  Active: inactive (dead) since Wed 2020-04-22 10:30:58 CEST; 2min 13s ago
  Main PID: 6030 (code=exited, status=0/SUCCESS)
  CPU: 2ms

Apr 22 10:30:52 lenovo systemd[1]: Starting Console Mouse manager...
Apr 22 10:30:52 lenovo systemd[1]: Started Console Mouse manager.
Apr 22 10:30:52 lenovo /usr/sbin/gpm[6030]: *** info [daemon/startup.c(136)]:
Apr 22 10:30:52 lenovo /usr/sbin/gpm[6030]: Started gpm successfully. Entered daemon mode.
Apr 22 10:30:58 lenovo systemd[1]: Stopping Console Mouse manager...
Apr 22 10:30:58 lenovo systemd[1]: Stopped Console Mouse manager.

# systemctl is-active gpm
inactive

# systemctl is-enabled gpm
masked

# systemctl start gpm
Failed to start gpm.service: Unit gpm.service is masked.
```

Ara desbloquejem el servei amb **unmask** per activar-lo i el tornem a bloquejar amb **mask**. El servei està actiu però a la propera arrancada no s'engegarà perquè està **masked**.

```
# systemctl unmask gpm
Removed /etc/systemd/system/gpm.service.

# systemctl start gpm

# systemctl enable gpm

# systemctl is-active gpm
active

# systemctl is-enabled gpm
enabled

# systemctl mask gpm
Created symlink /etc/systemd/system/gpm.service → /dev/null.

# systemctl is-active gpm
active

# systemctl is-enabled gpm
masked

# systemctl status gpm
• gpm.service
  Loaded: masked (/dev/null; masked)
  Active: active (running) since Wed 2020-04-22 10:35:45 CEST; 37s ago
  Main PID: 6132 (gpm)
  Tasks: 1 (limit: 4915)
  Memory: 272.0K
  CPU: 2ms
  CGroup: /system.slice/gpm.service
          └─6132 /usr/sbin/gpm -m /dev/input/mice -t exps2

Apr 22 10:35:45 lenovo systemd[1]: Starting Console Mouse manager...
```

```
Apr 22 10:35:45 lenovo systemd[1]: gpm.service: Failed to read PID from file /var/run/gpm.pid: Invalid argument
Apr 22 10:35:45 lenovo systemd[1]: Started Console Mouse manager.
Apr 22 10:35:45 lenovo /usr/sbin/gpm[6132]: *** info [daemon/startup.c(136)]:
Apr 22 10:35:45 lenovo /usr/sbin/gpm[6132]: Started gpm successfully. Entered daemon mode.
```

#### Pràctica proposada:

11. Instal·leu els serveis httpd, gpm, xinetd i ssh.
12. Activeu els serveis httpd i ssh i configureu-los disabled. Verifiqueu que estan actius. Reiniciar el sistema i observar que no estan Iniciar el sistema.
13. Amb els serveis inactius condfigurar-los (httpd i sshd) com a enabled. Verificar que estan enables però inactius. Reiniciar el sistema i observar que estan actius i enabled.
14. Fer mask del servei httpd i observar que no es pot fer enable. Fer unmask.

## Tipus de unit .service

### Identificar el fitxer de servei / Contingut d'un paquet

En els apartats anteriors hem vist que amb `systemctl` podem fer tota la gestió dels serveis del sistema i hem practicat amb els serveis `httpd`, `sshd` i `gpm`. El nom dels serveis usats són molt coneguts però quan instal·lem software com podem saber com s'anomena el servei?

Anem a pams, per instal·lar per exemple el servidor web instal·lem un paquet anomenat `httpd` i el servei té el mateix nom. Però per instal·lar el servei `ssh` cal instal·lar el paquet `openssh-server` i en canvi el nom del servei és `sshd` (no és igual que el nom del paquet).

Quan instal·lem els paquets d'un servei convé llistar el contingut del paquet i examinar quin és el fitxer que governa el servei (no l'executable) amb `systemctl`. Usualment serà un fitxer amb extensió `.service` i més concretament serà un fitxer amb la següent ubicació:

`/usr/lib/systemd/system/<nomservei>.service`

En aquest exemple examinem el paquet `gpm` i observem el seu fitxer de servei:

```
# rpm -ql gpm
/etc/gpm-root.conf
/etc/gpm-syn.conf
/etc/gpm-twiddler.conf
/usr/bin/disable-paste
/usr/bin/display-buttons
/usr/bin/display-coords
```

```
/usr/bin/get-versions
/usr/bin/gpm-root
/usr/bin/hltest
/usr/bin/mev
/usr/bin/mouse-test
/usr/lib/systemd/system/gpm.service
/usr/sbin/gpm
/usr/share/doc/gpm
...
```

En aquest exemple podem observar el fitxer de servei de OpenSSH-server

```
# rpm -ql openssh-server
/etc/pam.d/ssh
/etc/ssh/ssh_config
/etc/sysconfig/ssh
/usr/lib/.build-id
/usr/lib/.build-id/a8
/usr/lib/.build-id/a8/5de750b7404d62ee97164ac18606dd04b5e1b2
/usr/lib/.build-id/da
/usr/lib/.build-id/da/0187adabc6f72c457042a7c3ce1a577afc28fd
/usr/lib/systemd/system/ssh-keygen.target
/usr/lib/systemd/system/ssh-keygen@.service
/usr/lib/systemd/system/ssh.service
/usr/lib/systemd/system/ssh.socket
/usr/lib/systemd/system/ssh@.service
/usr/lib/tmpfiles.d/openssh.conf
/usr/lib64/fipscheck/ssh.hmac
/usr/libexec/openssh/sftp-server
/usr/libexec/openssh/ssh-keygen
/usr/sbin/ssh
/usr/share/man/man5/moduli.5.gz
/usr/share/man/man5/ssh_config.5.gz
/usr/share/man/man8/sftp-server.8.gz
/usr/share/man/man8/ssh.8.gz
/var/empty/ssh
```

Així doncs, si volem saber com es diu el servei hem d'identificar el fitxer `.service` en el llistat del contingut del paquet que hem instal·lat.

Un truc usual per identificar on està el fitxer de servei és usar l'ordre `locate`:

```
# locate httpd.service
/usr/lib/systemd/system/httpd.service
/usr/lib/systemd/system/httpd.service.d
/usr/share/man/man8/httpd.service.8.gz
/var/lib/docker/overlay2/ad8f34123659db7990e16c2bb95b5ad968d777b1e79253d9a91bb2e58d146796/diff/usr/lib/systemd/system/httpd.service
/var/lib/docker/overlay2/ad8f34123659db7990e16c2bb95b5ad968d777b1e79253d9a91bb2e58d146796/diff/usr/lib/systemd/system/httpd.service.d
```

Els fitxers `.service` són unitats de `systemctl` que descriuen el funcionament de serveis del sistema (no són l'executable del servei). Aquests fitxers indiquen quin és l'executable i en quines condicions s'ha d'engegar (què necessita per engegar-se i amb què és incompatible per engegar).

## Unit `.service`

Anem a observar el contingut d'un fitxer de servei, per exemple el de `gpm`:

```
# cat /usr/lib/systemd/system/gpm.service
```

```

[Unit]
Description=Console Mouse manager

# This could probably benefit from socket activation, but honestly I think it
# is time for gpm to go away, and hence I am not planning to spend the time
# to add socket activation here.

[Service]
ExecStart=/usr/sbin/gpm -m /dev/input/mice -t exps2
Type=forking
PIDFile=/var/run/gpm.pid

[Install]
WantedBy=multi-user.target

```

Podem observar que es tracta d'un fitxer de configuració en text pla que descriu tres seccions:

**[Unit]** és la secció on es descriu aquest fitxer, en concret aquesta és una unit de tipus `.service` corresponent a un servei, al servei `gpm`.

**[Service]** és la secció on es descriu com engegar/aturar/recarregar el servei. En aquesta secció hem de saber identificar quin és i com es diu l'executable, el binari, del dimoni del servei. En aquest exemple observem:

- **ExecStart** indica la ruta, el nom de l'executable i les opcions del binari executable del servei.
- **Type**: `forking` indica que el servei genera un fork, un subprocés.
- **PIDFile**: és costum dels serveis desar un fitxer de pid on s'hi desa el PID del servei. Aquest fitxer per convenció està a `/var/run` i té per nom el nom del servei.pid.

**[install]** en aquesta secció s'indica a quin target es vol que el servei sigui actiu quan es fa l'enable del servei. Observem que en aquest exemple `WantedBy` indica `multi-user.target`. Això significa que en fer `systemctl enable gpm` el sistema configurarà que sempre que s'iniciï l'ordinador el servei estigui actiu en el `multi-user.target`.

Examinem ara per exemple el servei `httpd`:

```

# cat /usr/lib/systemd/system/httpd.service
# See httpd.service(8) for more information on using the httpd service.

# Modifying this file in-place is not recommended, because changes
# will be overwritten during package upgrades. To customize the
# behaviour, run "systemctl edit httpd" to create an override unit.

# For example, to pass additional options (such as -D definitions) to
# the httpd binary at startup, create an override unit (as is done by
# systemctl edit) and enter the following:

#           [Service]
#           Environment=OPTIONS=-DMY_DEFINE

[Unit]
Description=The Apache HTTP Server
Wants=httpd-init.service
After=network.target remote-fs.target nss-lookup.target httpd-init.service

```

```
Documentation=man:httpd.service(8)

[Service]
Type=notify
Environment=LANG=C

ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND
ExecReload=/usr/sbin/httpd $OPTIONS -k graceful
# Send SIGWINCH for graceful stop
KillSignal=SIGWINCH
KillMode=mixed
PrivateTmp=true

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

Observant aquest fitxer `httpd.service` podem identificar clarament l'ordre que s'executa per iniciar el servei descrita a [ExecStart](#). L'ordre que s'executa per reiniciar el servei descrita a [ExecReload](#). Veiem que per finalitzar el servei s'utilitza el senyal indicat per [KillSignal](#) `SIGWINCH`. I finalment identifiquem que el servei per defecte s'instal·la en el mode `multi-user.target` tal i com indica [WantedBy](#).

Un últim exemple amb el fitxer `sshd.service`:

```
# cat /usr/lib/systemd/system/sshd.service
[Unit]

Description=OpenSSH server daemon
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target sshd-keygen.target
Wants=sshd-keygen.target

[Service]
Type=notify
EnvironmentFile=/etc/crypto-policies/back-ends/openssh-server.config
EnvironmentFile=/etc/sysconfig/ssh
ExecStart=/usr/sbin/sshd -D $OPTIONS $CRYPTO_POLICY
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s

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

Observem que el dimoni executable del servei `sshd` s'anomena `/usr/sbin/sshd` i que el senyal per fer un reload del sistema és el clàssic `SIGHUP` (Signal 1).

## list-unit-files

En el disseny de l'arrencada del sistema usant `systemd` els fitxers dels serveis s'instal·len a `/usr/lib/systemd/system`. En aquest directori hi podem trobar els fitxers `.service` de tots els serveis que hem instal·lat al sistema, tant si estan actius o `enabled` com si no.

Podem saber quins serveis hi ha instal·lats amb:

- Fer un llistat de tots els fitxers `.service` del directori `/usr/lib/systemd/system`.
- Usant `systemctl list-unit-files [--type=service]`



- Usant systemctl list-units [--type=service] --all

```
# cd /usr/lib/systemd/system
# ls *.service
abrt-ccpp.service          iscsi.service          sssd-nss.service
abrt-d.service            iscsi-shutdown.service sssd-pac.service
abrt-journal-core.service iscsiui.service        sssd-pam.service
abrt-oops.service         kdump.service         sssd-secrets.service
abrt-pstoreoops.service   kmod-static-nodes.service sssd.service
abrt-vmcore.service       kube-apiserver.service sssd-ssh.service
abrt-xorg.service         kube-controller-manager.service sssd-sudo.service
accounts-daemon.service   kubelet.service        suricata.service
alsa-restore.service      kube-proxy.service      switcheroo-control.service
alsa-state.service        kube-scheduler.service  systemd-ask-password-console.service
arp-ethers.service        ldconfig.service        systemd-ask-password-plymouth.service
atd.service               libvirt.service         systemd-ask-password-wall.service
...
```

```
# systemctl list-unit-files
UNIT FILE                                STATE
proc-sys-fs-binfmt_misc.automount      static
~.mount                                generated
dev-hugepages.mount                    static
dev-mqueue.mount                       static
proc-fs-nfsd.mount                     static
proc-sys-fs-binfmt_misc.mount          static
sys-fs-fuse-connections.mount          static
sys-kernel-config.mount                static
sys-kernel-debug.mount                 static
tmp.mount                              static
var-lib-machines.mount                 static
var-lib-nfs-rpc_pipefs.mount           static
virtualbox.mount                       generated
cups.path                              enabled
systemd-ask-password-console.path      static
systemd-ask-password-plymouth.path     static
systemd-ask-password-wall.path         static
session-2.scope                        transient
session-c1.scope                       transient
abrt-ccpp.service                      disabled
abrt-journal-core.service              enabled
abrt-oops.service                      enabled
...
420 unit files listed.
```

```
# systemctl list-unit-files --type=service
UNIT FILE                                STATE
abrt-ccpp.service                      disabled
abrt-journal-core.service              enabled
abrt-oops.service                      enabled
abrt-pstoreoops.service                disabled
abrt-vmcore.service                   enabled
abrt-xorg.service                      enabled
abrt-d.service                        enabled
accounts-daemon.service                enabled
alsa-restore.service                  static
alsa-state.service                    static
arp-ethers.service                    disabled
atd.service                           disabled
auditd.service                        enabled
auth-rpcgss-module.service             static
autovt@.service                       enabled
avahi-daemon.service                  enabled
blk-availability.service               disabled
bluetooth.service                     enabled
```

```
brltty.service          disabled
btattach-bcm@.service   static
cannberra-system-bootup.service    disabled
cannberra-system-shutdown-reboot.service disabled
cannberra-system-shutdown.service  disabled
chrony-dnssrv@.service    static
chrony-wait.service       disabled
chronyd.service           enabled
clean-mount-point@.service static
...
287 unit files listed.
```

#### # systemctl list-units --type=service

UNIT	LOAD	ACTIVE	SUB	DESCRIPTION
● abrt.service	loaded	failed	failed	ABRT Automated Bug Reporting Tool
accounts-daemon.service	loaded	active	running	Accounts Service
alsa-state.service	loaded	active	running	Manage Sound Card State (restore and store)
auditd.service	loaded	active	running	Security Auditing Service
avahi-daemon.service	loaded	active	running	Avahi mDNS/DNS-SD Stack
bluetooth.service	loaded	active	running	Bluetooth service
chronyd.service	loaded	active	running	NTP client/server
colord.service	loaded	active	running	Manage, Install and Generate Color Profiles
crond.service	loaded	active	running	Command Scheduler
cups.service	loaded	active	running	CUPS Scheduler
dbus.service	loaded	active	running	D-Bus System Message Bus
dracut-shutdown.service	loaded	active	exited	Restore /run/initramfs on shutdown
fedora-import-state.service	loaded	active	exited	Import network configuration from initramfs
fedora-readonly.service	loaded	active	exited	Configure read-only root support
firewalld.service	loaded	active	running	firewalld - dynamic firewall daemon
fwupd.service	loaded	active	running	Firmware update daemon
gdm.service	loaded	active	running	GNOME Display Manager
geoclue.service	loaded	active	running	Location Lookup Service
● gpm.service	masked	active	running	gpm.service
gssproxy.service	loaded	active	running	GSSAPI Proxy Daemon
iscsi-shutdown.service	loaded	active	exited	Logout off all iSCSI sessions on shutdown

...  
LOAD = Reflects whether the unit definition was properly loaded.  
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.  
SUB = The low-level unit activation state, values depend on unit type.

60 loaded units listed. Pass --all to see loaded but inactive units, too.  
To show all installed unit files use 'systemctl list-unit-files'.

#### # systemctl list-units --type=service --all

UNIT	LOAD	ACTIVE	SUB	DESCRIPTION
abrt-ccpp.service	loaded	inactive	dead	Install ABRT coredump hook
abrt-journal-core.service	loaded	inactive	dead	Creates ABRT problems from coredumpctl messages
abrt-oops.service	loaded	inactive	dead	ABRT kernel log watcher
abrt-vmcore.service	loaded	inactive	dead	Harvest vmcores for ABRT
abrt-xorg.service	loaded	inactive	dead	ABRT Xorg log watcher
● abrt.service	loaded	failed	failed	ABRT Automated Bug Reporting Tool
accounts-daemon.service	loaded	active	running	Accounts Service
alsa-restore.service	loaded	inactive	dead	Save/Restore Sound Card State
alsa-state.service	loaded	active	running	Manage Sound Card State (restore and store)
auditd.service	loaded	active	running	Security Auditing Service
auth-rpccss-module.service	loaded	inactive	dead	Kernel Module supporting RPCSEC_GSS
avahi-daemon.service	loaded	active	running	Avahi mDNS/DNS-SD Stack
blk-availability.service	loaded	inactive	dead	Availability of block devices
bluetooth.service	loaded	active	running	Bluetooth service
chronyd.service	loaded	active	running	NTP client/server
colord.service	loaded	active	running	Manage, Install and Generate Color Profiles
crond.service	loaded	active	running	Command Scheduler
cups.service	loaded	active	running	CUPS Scheduler
dbus.service	loaded	active	running	D-Bus System Message Bus
...				
wpa_supplicant.service	loaded	active	running	WPA supplicant
● ypbind.service	not-found	inactive	dead	ypbind.service

...  
LOAD = Reflects whether the unit definition was properly loaded.  
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.  
SUB = The low-level unit activation state, values depend on unit type.

153 loaded units listed.  
To show all installed unit files use 'systemctl list-unit-files'.

## Identificar l'enable / symlink manual

Sabem que en instal·lar els paquets de software d'un servei el paquet conté un fitxer `.service` que es copia al directori `/usr/lib/systemd/system`. Aquest fitxer `<nomservei>.service` és el que descriu com es governa el servei.

Amb les ordres `systemctl enable` i `systemctl disable` es configura el sistema operatiu per engegar automàticament (o no) el servei en iniciar el sistema. Però com ho fa?

Tornem a repassar amb el servei `httpd` què passa en fer `enable` i `disable`. Quan s'activa l'inici per defecte del servei (recordeu que no l'activa ara, per activar-lo ara cal fer un `start`) es crea un symbolic link que és realment el que serveix per indicar a l'arrancada que cal engegar el servei.

```
# systemctl enable httpd
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service →
/usr/lib/systemd/system/httpd.service.

# ll /etc/systemd/system/multi-user.target.wants/httpd.service
lrwxrwxrwx 1 root root 37 Apr 22 11:53 /etc/systemd/system/multi-user.target.wants/httpd.service ->
/usr/lib/systemd/system/httpd.service
```

Fer un `enable` és tan senzill com crear un enllaç simbòlic dins d'un target que apunti al servei que volem engegar en aquell target. Tornem-hi que no es deu haber entès!.

Si fem `enable` de `httpd.service`:

- Volem que el servei `httpd` s'engegi automàticament en iniciar el sistema operatiu. S'engega automàticament en el mode `multi-user.target` (i superiors).
- El fitxer del servei es troba a `/usr/lib/systemd/system/httpd.service`. Aquest és el fitxer de govern del servei que s'ha instal·lat en instal·lar el paquet.
- Quan hem examinat el fitxer `http.service` hem vist que està configurat a la secció `[install]` amb la directiva `WantedBy` indicant `multi-user.target`.
- Significa que el servei `httpd.service` s'instal·la amb `enable` al target `multi-user.target`. Però com?

Com:

- Simplement creant un enllaç simbòlic dins de `multi-user.target.wants` amb el nom del servei que apunti on hi ha realment el servei.
- A `/etc/systemd/system/multi-user.target.wants` hi ha tot allò que vol, que requereix, que vol engegar el target `multi-user.target`.
- Observeu (en el requadre anterior) que després de fer l'`enable` s'ha creat un enllaç simbòlic de nom `httpd.service` dins de `wants` (a `/etc/...`) que apunta al fitxer del servei (dins de `/usr/lib/...`).

Anàlogament què fa `systemctl` realment quan li diem que volem desactivar arrencada automàtica d'un servei en iniciar el sistema amb `systemctl disable`? Doncs simplement **esborra el symlink** que ha creat en fer l'enable.

Observem ara que en fer el `systemctl disable` contesta que està esborrant l'enllaç simbòlic. Si mirem de llistar-lo de `/etc...` ara ja no hi és.

```
# systemctl disable httpd
Removed /etc/systemd/system/multi-user.target.wants/httpd.service.

# ll /etc/systemd/system/multi-user.target.wants/httpd.service
ls: cannot access '/etc/systemd/system/multi-user.target.wants/httpd.service': No such file or directory
```

## Enable / Disable manual

Segur? Es cert tot això o és com el que explica el govern?

Anem a fer l'enable i el disable manualment amb l'ordre de crear simbòlic **links ln -s**:

```
❑ ln -s <origen> <link-destí>
```

```
❑ ln -s /usr/lib/systemd/system/<nomservei>.service /etc/systemd/system/multi-user.target.wants/<nomservei>.service
```

Crear el enable

```
# systemctl is-enabled httpd
disabled

# ln -s /usr/lib/systemd/system/httpd.service /etc/systemd/system/multi-user.target.wants/httpd.service

# systemctl is-enabled httpd
enabled

# ls -l /etc/systemd/system/multi-user.target.wants/httpd.service
lrwxrwxrwx 1 root root 37 Apr 22 12:13 /etc/systemd/system/multi-user.target.wants/httpd.service -> /usr/lib/systemd/system/httpd.service
```

Eliminar el enable

```
# systemctl is-enabled httpd
enabled

# rm /etc/systemd/system/multi-user.target.wants/httpd.service
rm: remove symbolic link '/etc/systemd/system/multi-user.target.wants/httpd.service'? y

# systemctl is-enabled httpd
disabled

# ls -l /etc/systemd/system/multi-user.target.wants/httpd.service
ls: cannot access '/etc/systemd/system/multi-user.target.wants/httpd.service': No such file or directory
```

Pràctica proposada:

15. Identificar el fitxer `.service` del servei `xinetd`. Identificar l'executable.

16. Configurar manualment amb symlinks el enable del servei xinetd i verificar-ho.
17. Eliminar el enable del servei xinetd manualment amb el symlink. Verificar-ho.

---

# El model de funcionament de systemd

---

## Arquitectura de systemd

Systemd es va dissenyar per poder realitzar una arrencada més ràpida de la màquina, en lloc d'anar carregant els serveis un després de l'altre com feia el sistema antic amb init systemd ho engega tot de cop, en paral·lel. Els serveis aniran progressant mentre puguin i no depenguin d'altres. Quan tinguin una dependència esperaran a que aquesta sigui satisfeta i reprendran el funcionament.

Els conceptes claus de systemd són:

- ❑ Tot està estructurat en units. Hi ha units de diversos tipus en especial identifiquem les de tipus `.service`, `.target` i `.socket`.
- ❑ Quan instal·lem serveis com per exemple `sshd`, `httpd` o `gpm` els fitxers `.service` de govern del servei es desen sempre a `/usr/lib/systemd/system`.
- ❑ A `/usr/lib/systemd/system` hi ha els fitxers de les units instal·lades, amb independència de si estan actives o enabled.
- ❑ Els fitxers de unit de serveis `.service` tenen una secció anomenada `[install]` on s'indica amb `WantedBy` a quin target o targets (pot ser més d'un) s'instal·la el servei amb `enable`.
- ❑ L'arrencada del sistema està dissenyada a través de targets. Un target és un objectiu a complir (no una llista de passos a fer). Per assolir un target cal complir una sèrie de requeriments per exemple que estiguin actius altres targets (per arribar al cim de l'everest primer he de passar pel camp base, pel camp 4, pel camp 5, etc). Veurem que per assolir `graphical.target` és requisit prèvi estar actiu `multi-user.target`. A la vegada aquest requereix de `basic.target`, etc.
- ❑ Un target també defineix quines coses vol per poder-se considerar assolit, amb el directori `Wants` s'estructura quins són els serveis que aquest target vol que estiguin engegats. Un exemple és mirar el directori `/etc/systemd/system/multi-user.target.wants` que conté els enllaços simbòlics a tots els serveis que amb `enable` s'han configurat per estar actius en aquest target.

Repassem, en el sistema anterior init es feia una llista ordenada de tot allò que s'havia d'anar fent, pas a pas per engegar la màquina. El pas x havia d'esperar els passos anteriors a estar completats per poder-se executar.

Amb systemd es defineix un objectiu, volem engegar la màquina per exemple en mode gràfic. El target graphical.target t'una sèrie de requisits que evidentment no es compleixen, de manera que el que fa és anar engegant tot allò que necessita per a complir-los. D'alguna manera podem dir que systemd comença al revés, en lloc de dir què cal engegar diu que per tenir engegat tal target què li fa falta.

Tot comença al grub, en el grub s'indica quin és el target que systemd ha d'engegar. Recordem que el grub carrega el kernel i que el kernel carrega systemd com a primer procés amb PID 1 del sistema. En aquest punt systemd intenta fer actiu el target indicat, usualment el default.target corresponent a graphical.target. De manera que es mira el fitxer de unit de graphical.target i va estirant allò que necessita.

## Systemd Units

Però frenem i anem a mirar-ho tot una mica més a poc a poc. Systemd treballa amb units, n'hi ha de varis tipus tot i que les que treballem principalment són .service, .target i .socket.

### # systemctl -t help

Available unit types:

service  
socket  
busname  
target  
device  
mount  
automount  
swap  
timer  
path  
slice  
scope

El propi nom del tipus de units ja descriu de què són: .service correspon a units de serveis, .socket de sockets de xarxa, .mount de muntatge de sistemes de fitxers, .target de definició de nivells d'arrencada, etc.

Podem llistar le sunits amb les ordres:

- ☐ systemctl list-units
- ☐ systemctl list-units --type=<tipus>
- ☐ systemctl list-units --type=<tipus> --all
- ☐ systemctl list-unit-files
- ☐ systemctl list-unit-files --type=<tipus>

Per saber quina és la configuració d'un fitxer de units podem consultar l'ajuda de [man systemd.unit](#). Per saber la configuració concreta d'un tipus d'unit podem consultar el man específic d'aquest tipus, per exemple [man systemd.target](#) i [man systemd.service](#).

## Systemd .target / Analitzem l'arrencada

L'arrencada del sistema és que systemd aconsegueixi fer actiu el target (usualment default.target). Un target és un tipus de unit on es descriuen els requeriments que calen per assolir aquell target. És com anar estirant un fil, per tenir graphical.target cal tenir engegat multi-user.target, per tenir multi-user cal tenir engegat tal altre i tots els wants que hi ha definits. Per engegar tal servei (dins del directori wants requereix que tal altre estigui actiu)... and so on.

Anem a veure els fitxers de units d'alguns dels targets:

```
# cat /usr/lib/systemd/system/graphical.target
# This file is part of systemd.
#
# systemd is free software; you can redistribute it and/or modify it
# under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation; either version 2.1 of the License, or
# (at your option) any later version.

[Unit]
Description=Graphical Interface
Documentation=man:systemd.special(7)
Requires=multi-user.target
Wants=display-manager.service
Conflicts=rescue.service rescue.target
After=multi-user.target rescue.service rescue.target display-manager.service
AllowIsolate=yes
```

Observem que té una única secció [Unit] on es descriuen els requeriments per poder assolir el target. Si consultem el man systemd.target veurem que hi ha infinitat d'opcions però ens conformem en mirar d'entendre les que apareixen llistades:

### Requires

Indica que és necessari que estigui prèviament actiu multi-user.target. De manera que en aquest cas caldrà anar a examinar multi-user.target i fer tot allò necessari per engegar-lo abans.

### Wants

Tot i ser semblants Wants i Requires indiquen coses diferents (potser per a un nivell més avançat?). En aquest exemple s'indica que engegi el display-manager. De fet el graphical.target és un multi-user.target més l'entorn gràfic.

### Conflicts

Indica amb quins altres targets és incompatible. És lògic que el servei gràfic és incompatible amb els targets de rescue que són en mode text i amb menys serveis.



## After

Indica que aquest target (graphical.target) s'ha d'activar **després** (si després i no abans!) dels units llistats. Ha de quedar clar que aquí diu que només es pot activar graphical.target si prèviament s'han activat (després de) els targets multi-user.target o bé rescue.target.

No diu que sigui obligatori activar el que hi ha llistat al After (és incompatible activar multi-user.target i rescue.target al mateix temps). Imaginem que fem un camí muntanya amunt, el que diu és que per ser al cim graphical.target o bé venim del camí rescue.target o bé del camí multi-user.target.

Generalment els alumnes entenen aquesta directiva al revés i diuen que després de graphical.target s'engega tot allò descrit a la directiva After. Això va fantàstic per saber a quins alumnes cal suspendre!.

## AllowIsolate

Aquesta directiva indica que es tracta d'un target al que podem iniciar la màquina, al que podem accedir per treballar-hi. Tenen configurat amb **Yes** els targets típics de emergency, rescue, multi-user i graphical.

Quan la directiva indica **No** significa que es tracta d'un target intermedi, necessari per a l'arrencada del sistema com a pas previ d'altres targets, però que no podem fer-ne un target d'arrencada. En són exemples sysinit.target o basic.target.

Anem ara a veure el target multi-user.target:

```
# cat /usr/lib/systemd/system/multi-user.target
# This file is part of systemd.
#
# systemd is free software; you can redistribute it and/or modify it
# under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation; either version 2.1 of the License, or
# (at your option) any later version.

[Unit]
Description=Multi-User System
Documentation=man:systemd.special(7)
Requires=basic.target
Conflicts=rescue.service rescue.target
After=basic.target rescue.service rescue.target
AllowIsolate=yes
```

Estudiem què és el que fa falta per engegar la màquina en mode consola multi-user.target. Simplement diu que **Requires** basic.target i indica que prèviament cal haver engegat o basic.target o rescue.target (**After**). Per tant systemd per poder fer actiu multi-user.target sap que abans ha d'engegar basic.target, que al seu temps requerirà d'altres coses...

Observem que aquí no hi ha la directiva **Wants**. Aquesta directiva es pot implementar en el fitxer de unit .target o es pot implementar en forma de directori Wants. Hem vist que en general els serveis que instal·lem per defecte s'instal·len al multi-user.target quan fem l'enable. Per tant per poder estar actiu multi-user.target abans cal haver engegat tots els serveis que hi ha al seu directori **.wants**.

## Directori .wants

Com a mecanisme alternatiu al a directiva Wants dind un fitxer de unit .target systemd permet la utilització d'un directori .wants. Aquest directori conté dins seu els symlinks que apunten als serveis que cal engegar.

Anem a veure el llistat dels serveis que engega una màquina amb multi-user.target. Aquesta és una màquina amb pocs serveis.:

```
# ls -l /etc/systemd/system/multi-user.target.wants/
total 0
lrwxrwxrwx. 1 root root 37 Sep 17 2019 abrt.service -> /usr/lib/systemd/system/abrt.service
lrwxrwxrwx. 1 root root 49 Sep 17 2019 abrt-journal-core.service -> /usr/lib/systemd/system/abrt-journal-core.service
lrwxrwxrwx. 1 root root 41 Sep 17 2019 abrt-oops.service -> /usr/lib/systemd/system/abrt-oops.service
lrwxrwxrwx. 1 root root 43 Sep 17 2019 abrt-vmcore.service -> /usr/lib/systemd/system/abrt-vmcore.service
lrwxrwxrwx. 1 root root 41 Sep 17 2019 abrt-xorg.service -> /usr/lib/systemd/system/abrt-xorg.service
lrwxrwxrwx. 1 root root 38 Sep 17 2019 auditd.service -> /usr/lib/systemd/system/auditd.service
lrwxrwxrwx. 1 root root 44 Sep 17 2019 avahi-daemon.service -> /usr/lib/systemd/system/avahi-daemon.service
lrwxrwxrwx. 1 root root 39 Sep 25 2019 chronyd.service -> /usr/lib/systemd/system/chronyd.service
lrwxrwxrwx. 1 root root 37 Sep 17 2019 crond.service -> /usr/lib/systemd/system/crond.service
lrwxrwxrwx. 1 root root 33 Sep 17 2019 cups.path -> /usr/lib/systemd/system/cups.path
lrwxrwxrwx. 1 root root 39 Sep 17 2019 dbxtool.service -> /usr/lib/systemd/system/dbxtool.service
lrwxrwxrwx. 1 root root 41 Sep 17 2019 firewalld.service -> /usr/lib/systemd/system/firewalld.service
lrwxrwxrwx. 1 root root 35 Apr 22 09:36 gpm.service -> /usr/lib/systemd/system/gpm.service
lrwxrwxrwx. 1 root root 38 Sep 17 2019 mcelog.service -> /usr/lib/systemd/system/mcelog.service
lrwxrwxrwx. 1 root root 41 Sep 17 2019 mdmonitor.service -> /usr/lib/systemd/system/mdmonitor.service
lrwxrwxrwx. 1 root root 44 Sep 17 2019 ModemManager.service -> /usr/lib/systemd/system/ModemManager.service
lrwxrwxrwx. 1 root root 46 Sep 17 2019 NetworkManager.service -> /usr/lib/systemd/system/NetworkManager.service
lrwxrwxrwx. 1 root root 41 Sep 17 2019 nfs-client.target -> /usr/lib/systemd/system/nfs-client.target
lrwxrwxrwx. 1 root root 40 Sep 17 2019 remote-fs.target -> /usr/lib/systemd/system/remote-fs.target
lrwxrwxrwx. 1 root root 36 Sep 17 2019 rngd.service -> /usr/lib/systemd/system/rngd.service
lrwxrwxrwx. 1 root root 36 Apr 22 10:21 sshd.service -> /usr/lib/systemd/system/sshd.service
lrwxrwxrwx. 1 root root 36 Sep 17 2019 sssd.service -> /usr/lib/systemd/system/sss.service
lrwxrwxrwx. 1 root root 53 Sep 27 2019 vboxautostart.service -> /usr/lib/systemd/system/vboxautostart.service
lrwxrwxrwx. 1 root root 55 Sep 27 2019 vboxballoonctrl.service -> /usr/lib/systemd/system/vboxballoonctrl.service
lrwxrwxrwx. 1 root root 39 Sep 27 2019 vboxdrv.service -> /usr/lib/systemd/system/vboxdrv.service
lrwxrwxrwx. 1 root root 47 Sep 27 2019 vboxweb.service -> /usr/lib/systemd/system/vboxweb.service
lrwxrwxrwx. 1 root root 40 Sep 17 2019 vmttoolsd.service -> /usr/lib/systemd/system/vmttoolsd.service
```

Tot allò que configurem amb `systemctl enable` com a serveis que volem activar en l'arrencada del sistema ens apareixerà en aquest llistat.

Per tant fins ara hem vist que per engegar en mode gràfic (graphical.target) cal engegar el servidor gràfic un cop estigui actiu (després de) el mode text (multi-user.target). Per engegar el mode text cal que estigui actiu el target basic.target i un cop ho estigui systemd ha d'engegar tots els serveis indicats al directori .wants:

[/etc/systemd/system/multi-user.target.wants.](#)

Continuem analitzant l'arrencada, ara és el torn de basic.target:

```
# cat /usr/lib/systemd/system/basic.target
# This file is part of systemd.
#
# systemd is free software; you can redistribute it and/or modify it
# under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation; either version 2.1 of the License, or
# (at your option) any later version.

[Unit]
Description=Basic System
Documentation=man:systemd.special(7)
Requires=sysinit.target
Wants=sockets.target timers.target paths.target slices.target
After=sysinit.target sockets.target paths.target slices.target tmp.mount

# We support /var, /tmp, /var/tmp, being on NFS, but we don't pull in
# remote-fs.target by default, hence pull them in explicitly here. Note that we
```

```
# require /var and /var/tmp, but only add a Wants= type dependency on /tmp, as
# we support that unit being masked, and this should not be considered an error.
RequiresMountsFor=/var /var/tmp
Wants=tmp.mount
```

```
# ls -l /etc/systemd/system/basic.target.wants/
total 0
lrwxrwxrwx. 1 root root 43 Sep 17 2019 dnf-makecache.timer -> /usr/lib/systemd/system/dnf-makecache.timer
```

Podem observar que systemd continua estirant la cadena de què cal engegar abans per poder fer actiu basic.target. Es requereix sysinit.target i veiem fins que no estiguin actius sysinit.target, sockets.target, paths.target, slices.target i tmp.mount no es pot activar basic.target.

Observem que no hi ha la directiva [AllowIsolation](#), en no ser-hi per defecte el valor és [No](#). Per tant aquest és un target al que no es pot arrencar la màquina.

Continuem ara analitzant sysinit.target:

```
# cat /usr/lib/systemd/system/sysinit.target
# This file is part of systemd.
#
# systemd is free software; you can redistribute it and/or modify it
# under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation; either version 2.1 of the License, or
# (at your option) any later version.

[Unit]
Description=System Initialization
Documentation=man:systemd.special(7)
Conflicts=emergency.service emergency.target
Wants=local-fs.target swap.target
After=local-fs.target swap.target emergency.service emergency.target
```

Veiem que requereix de coses de més baix nivell (cada vegada som més a prop del hardware...) i que és incompatible amb emergency.target.

Així podriem seguir anant desgranant tota l'arrancada...

En el requadre següent podem observar com s'ha iniciat el sistema:

```
# systemctl list-dependencies | grep .target
default.target
├─multi-user.target
├─basic.target
│   ├── paths.target
│   ├── slices.target
│   ├── sockets.target
│   └─sysinit.target
│       ├── cryptsetup.target
│       ├── local-fs.target
│       └─swap.target
│   └─timers.target
├─getty.target
├─nfs-client.target
├─remote-fs-pre.target
├─remote-fs.target
├─nfs-client.target
└─remote-fs-pre.target
```

Descripció global: kernel engega al default.target (o al indicat amb systemd.unit=nom.target com a parametre del kernel). Aquest target estira de totes les dependències prèvies (les podem llistar amb list-dependencies o amb systemd-analyze). A part de les descrites en els fitxers target cal carregar els elements dels directoris .wants de cada target per poder assolir completament el target com a is-active.

## Target dependencies

Així doncs arrencada amb systemd consisteix en definir les dependències que tenen les units. I el que és més sorprenen és que systemd comença per dalt, dient què vol tenir engegat, i a partir d'aquí va estirant tot allò que necessita per engegar-ho (cap avall).

Podem veure les dependències dels targets (i de les units en general) amb les ordres:

- ❑ `systemctl list-dependencies`
- ❑ `systemctl list-dependencies | grep .target`
- ❑ `systemctl list-dependencies | grep .service`
- ❑ `systemctl list-dependencies <target>`
- ❑ `systemctl list-dependencies <target> | grep .target`
- ❑ `systemctl list-dependencies <target> | grep .service`

Veure totes les dependències:

```
# systemctl list-dependencies
default.target
● —accounts-daemon.service
● —gdm.service
● —rtkit-daemon.service
● —switcheroo-control.service
● —systemd-update-utmp-runlevel.service
● —udisks2.service
● —multi-user.target
● —abrt-journal-core.service
● —abrt-oops.service
...
```

Veure totes les dependències de multi-user.target:

```
multi-user.target
● —abrt-journal-core.service
● —abrt-oops.service
● —abrt-vmcore.service
● —abrt-xorg.service
● —abrt.service
● —auditd.service
● —avahi-daemon.service
● —chronyd.service
● —crond.service
● —cups.path
● —dbus.service
● —dbxtool.service
● —firewalld.service
● —gpm.service
...
```

Veure totes les dependències de tipus .service de multi-user.target:

```
# systemctl list-dependencies multi-user.target | grep .service
● └─abrt-journal-core.service
● └─abrt-oops.service
● └─abrt-vmcore.service
● └─abrt-xorg.service
● └─abrt.service
● └─auditd.service
● └─avahi-daemon.service
● └─chronyd.service
● └─crond.service
● └─dbus.service
● └─dbxtool.service
...
```

Veure tots els targets dels que depèn multi-user.target:

```
# systemctl list-dependencies multi-user.target | grep .target
multi-user.target
● └─basic.target
● └─paths.target
● └─slices.target
● └─sockets.target
● └─sysinit.target
● └─cryptsetup.target
● └─local-fs.target
● └─swap.target
● └─timers.target
● └─getty.target
● └─nfs-client.target
● └─remote-fs-pre.target
● └─remote-fs.target
● └─nfs-client.target
● └─remote-fs-pre.target
```

Podem comprovar que els elements destacats en blau són els targets requerits per basic.target:

```
# systemctl list-dependencies basic.target | grep .target
basic.target
● └─paths.target
● └─slices.target
● └─sockets.target
● └─sysinit.target
● └─cryptsetup.target
● └─local-fs.target
● └─swap.target
● └─timers.target
```

Què és tot el que engega sysinit.target?

```
# systemctl list-dependencies sysinit.target
sysinit.target
● └─dev-hugepages.mount
● └─dev-mqueue.mount
● └─dracut-shutdown.service
● └─fedora-import-state.service
● └─iscsi.service
● └─kmod-static-nodes.service
● └─ldconfig.service
● └─lvm2-lvmetad.socket
● └─lvm2-lvmpolld.socket
● └─lvm2-monitor.service
● └─multipathd.service
...
```

Podem veure informació més detallada de les unitats (de tot tipus) i observar quins són els seus requeriments. Ho podem fer amb l'ordre següent, que mostra tanta informació que fins i tot costa de pair:

❏ `systemctl show <unit>`

En el següent exemple podem veure que `multi-user.target` requereix de `basic.target` i al mateix temps és requerit per `graphical.target`. Quins elements `Wants` i la llista de `Before` i de `After`:

```
# systemctl show multi-user.target
Id=multi-user.target
Names=runlevel3.target runlevel2.target runlevel4.target multi-user.target
Requires=basic.target
Wants=systemd-user-sessions.service ModemManager.service vboxautostart-service.service getty.target vmtoolsd.service remote-fs.target dbxtool.service
RequiredBy=graphical.target
Conflicts=rescue.target shutdown.target rescue.service
Before=systemd-update-utmp-runlevel.service shutdown.target graphical.target
After=systemd-user-sessions.service dbxtool.service NetworkManager.service vmtoolsd.service abrt.service plymouth-quit.service crond.service basic.ta
Documentation=man:systemd.special(7)
Description=Multi-User System
LoadState=loaded
ActiveState=active
SubState=active
FragmentPath=/usr/lib/systemd/system/multi-user.target
UnitFileState=static
UnitFilePreset=disabled
StateChangeTimestamp=Wed 2020-04-22 08:52:59 CEST
StateChangeTimestampMonotonic=40596724
InactiveExitTimestamp=Wed 2020-04-22 08:52:59 CEST
InactiveExitTimestampMonotonic=40596724
ActiveEnterTimestamp=Wed 2020-04-22 08:52:59 CEST
ActiveEnterTimestampMonotonic=40596724
ActiveExitTimestampMonotonic=0
InactiveEnterTimestampMonotonic=0
CanStart=yes
CanStop=yes
CanReload=no
CanIsolate=yes
StopWhenUnneeded=no
RefuseManualStart=no
RefuseManualStop=no
AllowIsolate=yes
DefaultDependencies=yes
OnFailureJobMode=replace
```

Llistat de `basic.target`. Podem observar el valor de `Allowisolate` a No.

```
# systemctl show basic.target
Id=basic.target
Names=basic.target
Requires=-.mount sysinit.target
Wants=slices.target sockets.target tmp.mount selinux-autorelabel-mark.service paths.target timers.target dnf-makecache.timer
RequiredBy=multi-user.target initrd.target
Conflicts=shutdown.target
Before=sshd-keygen@ed25519.service abrt.service dbxtool.service rtkit-daemon.service unbound-anchor.service wpa_supplicant.service geoclue.service
chronyd.service sssd-kcm.service sssd-secrets.service abrt-vmcore.service sshd.service udisks2.service packagekit.service
After=dnf-makecache.timer slices.target sockets.target tmp.mount paths.target systemd-ask-password-plymouth.path sysinit.target -.mount
RequiresMountsFor=/var /var/tmp
Documentation=man:systemd.special(7)
Description=Basic System
LoadState=loaded
ActiveState=active
SubState=active
FragmentPath=/usr/lib/systemd/system/basic.target
UnitFileState=static
UnitFilePreset=disabled
StateChangeTimestamp=Wed 2020-04-22 08:52:34 CEST
StateChangeTimestampMonotonic=16036886
InactiveExitTimestamp=Wed 2020-04-22 08:52:34 CEST
InactiveExitTimestampMonotonic=16036886
ActiveEnterTimestamp=Wed 2020-04-22 08:52:34 CEST
ActiveEnterTimestampMonotonic=16036886
ActiveExitTimestamp=Wed 2020-04-22 10:52:22 CEST
ActiveExitTimestampMonotonic=3886979
InactiveEnterTimestamp=Wed 2020-04-22 10:52:22 CEST
InactiveEnterTimestampMonotonic=3886979
CanStart=yes
CanStop=yes
CanReload=no
```

```
CanIsolate=no
StopWhenUnneeded=no
RefuseManualStart=no
RefuseManualStop=no
AllowIsolate=no
DefaultDependencies=yes
OnFailureJobMode=replace
IgnoreOnIsolate=no
NeedDaemonReload=no
JobTimeoutUSec=infinity
JobRunningTimeoutUSec=infinity
JobTimeoutAction=none
ConditionResult=yes
AssertResult=yes
ConditionTimestamp=Wed 2020-04-22 08:52:34 CEST
ConditionTimestampMonotonic=16036877
AssertTimestamp=Wed 2020-04-22 08:52:34 CEST
AssertTimestampMonotonic=16036877
Transient=no
Perpetual=no
StartLimitIntervalSec=10000000
StartLimitBurst=5
StartLimitAction=none
InvocationID=b6b037a04c63486b87b519ad8d0dfa83
```

## Analitzem l'arrencada d'un servei

Hem analitzat l'arrancada del sistema i repassat els targets, anem ara a veure com s'hi integra un servei com per exemple el servei http.

Fitxer del servei:

```
# cat /usr/lib/systemd/system/httpd.service
# See httpd.service(8) for more information on using the httpd service.

# Modifying this file in-place is not recommended, because changes
# will be overwritten during package upgrades. To customize the
# behaviour, run "systemctl edit httpd" to create an override unit.

# For example, to pass additional options (such as -D definitions) to
# the httpd binary at startup, create an override unit (as is done by
# systemctl edit) and enter the following:

#           [Service]
#           Environment=OPTIONS=-DMY_DEFINE

[Unit]
Description=The Apache HTTP Server
Wants=httpd-init.service
After=network.target remote-fs.target nss-lookup.target httpd-init.service
Documentation=man:httpd.service(8)

[Service]
Type=notify
Environment=LANG=C

ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND
ExecReload=/usr/sbin/httpd $OPTIONS -k graceful
# Send SIGWINCH for graceful stop
KillSignal=SIGWINCH
KillMode=mixed
PrivateTmp=true

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

Anteriorment ja hem analitzat la secció [\[Service\]](#) on es descriu com engegar / recarregar / aturar el servei, i la secció [\[install\]](#) que indica que és un servei de multi-user.target. Fixem-nos ara en la secció [\[unit\]](#).

La directiva [After](#) indica clarament que per poder usar aquest servei abans cal que estigui engegada la xarxa amb network.target. Té lògica oi? Si volem usar el servidor web caldrà tenir xarxa!.

Anem ara a veure la descripció del servei sshd:

```
# cat /usr/lib/systemd/system/sshd.service
[Unit]
Description=OpenSSH server daemon
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target sshd-keygen.target
Wants=sshd-keygen.target

[Service]
Type=notify
EnvironmentFile=/etc/crypto-policies/back-ends/openssh-server.config
EnvironmentFile=/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D $OPTIONS $CRYPTO_POLICY
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s

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

Altre cop podem observar que amb molta lògica fa falta disposar del servei de xarxa network.target per poder usar sshd. Però fixem-nos en la directiva [Wants](#) que requereix el target sshd-keygen.target.

El servei ssh necessita que en el host estiguin creades unes claus de host (claus públiques/privades) per poder funcionar. Això s'explica abastament a hisx2. Ara de moment quedem-nos amb la idea de que si el host no té les claus creades no pot funcionar. En un ordinador on no s'ha activat mai el servei sshd si volguessim engegar directament l'executable del dimoni (podem veure que és /usr/sbin/sshd) ens fallaria. Diria que no hi ha les claus de host i no pot funcionar.

On es creen aquestes claus?

Si estirem el fil de ssh.service veiem que requereix de ssh-keygen.target. Examinem-lo:

```
# cat /usr/lib/systemd/system/sshd-keygen.target
[Unit]
Wants=sshd-keygen@rsa.service
Wants=sshd-keygen@ecdsa.service
Wants=sshd-keygen@ed25519.service
PartOf=sshd.service
```

Aquest és un fitxer ben simple que 'estira', requereix la creació de claus de tipus rsa, ecdsa i ed25519 (són tipus de claus criptogràfiques).



Examinem sshd-keygen@.service:

```
# cat /usr/lib/systemd/system/sshd-keygen@.service
[Unit]
Description=OpenSSH %i Server Key Generation
ConditionFileNotEmpty=!/etc/ssh/ssh_host_%i_key

[Service]
Type=oneshot
EnvironmentFile=-/etc/sysconfig/ssh
ExecStart=/usr/libexec/openssh/sshd-keygen %i

[Install]
WantedBy=sshd-keygen.target
```

Finalment podem observar on es creen les claus, en la instrucció descrita en la directiva **ExecStart** que diu `/usr/libexec/openssh/sshd-keygen %i`. Executa l'ordre **ssh-keygen** que crea les claus del tipus rebut com a argument.

L'objectiu d'aquesta explicació de sshd NO és entendre tot això de les claus (ho fem a hisx2) sinó jugar a estirar el fil dels unitats.

Pràctica proposada:

18. Instal·la el paquet xinetd. Llista els seus components.
19. Identifica el fitxer de servei de xinetd i llista'l. Identifica l'executable del servei i a quin target s'instal·la.
20. Llista el directori on hi ha els fitxers de les unitats de servei de multi-user.target. Identifica el fitxer corresponent al servei xinetd.
21. Fes enable del servei xinetd. S'ha creat un link. Llista el directori origen i el directori destí del link. On s'ha creat el link? Perquè?
22. Llista les dependències de multi-user.target: totes; només els targets; només els serveis.
23. Llista les dependències de xinetd.service.
24. Fes l'ordre `systemctl show` del servei xinetd i identifica els elements: After i WantedBy.
25. Es pot iniciar sessió en el sysinit.target? Explica el perquè documentant-ho.

## Arrencades de targets diferents

### targets de recuperació i emergència

En els apartats anteriors ens hem centrat en treballar els targets generals d'arrancada del sistema `graphical.target` i `multi-user.target`, teot veient les seves dependències de `basic.target` i `sysinit.target`. Però hi ha més targets, de fet molts més!.

Altres targets de systemd:

- ❑ Emergency.target
- ❑ Rescue.target

## emergency.target

Target per a solucions d'emergència, engega el kernel i systemd però res més. Continua essent necessari escriure el password de root per poder accedir al shell.

### Fitxer de emergency.target

```
# cat /usr/lib/systemd/system/emergency.target
# This file is part of systemd.
#
# systemd is free software; you can redistribute it and/or modify it
# under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation; either version 2.1 of the License, or
# (at your option) any later version.

[Unit]
Description=Emergency Mode
Documentation=man:systemd.special(7)
Requires=emergency.service
After=emergency.service
AllowIsolate=yes
```

Com podem veure és un ‘joan palom jo me'l guis jo me'l com’ que requereix d'ell mateix i es necessita només a ell mateix.

```
# systemctl list-dependencies emergency.target
emergency.target
● └─emergency.service
```

```
# systemctl show emergency.target
Id=emergency.target
Names=emergency.target
Requires=emergency.service
Conflicts=shutdown.target
ConflictedBy=dracut-pre-trigger.service dracut-initqueue.service dracut-pre-udev.service dracut-pre-pivot.service dracut-mount.service dracut-cmdline.
Before=initrd-switch-root.target shutdown.target sysinit.target
After=fedora-import-state.service fedora-readonly.service emergency.service
Documentation=man:systemd.special(7)
Description=Emergency Mode
LoadState=loaded
ActiveState=inactive
SubState=dead
FragmentPath=/usr/lib/systemd/system/emergency.target
UnitFileState=static
UnitFilePreset=disabled
StateChangeTimestamp=Wed 2020-04-22 10:52:22 CEST
StateChangeTimestampMonotonic=3723257
InactiveExitTimestampMonotonic=0
ActiveEnterTimestampMonotonic=0
ActiveExitTimestampMonotonic=0
InactiveEnterTimestampMonotonic=0
CanStart=yes
CanStop=yes
CanReload=no
CanIsolate=yes
StopWhenUnneeded=no
RefuseManualStart=no
RefuseManualStop=no
AllowIsolate=yes
DefaultDependencies=yes
OnFailureJobMode=replace
IgnoreOnIsolate=no
NeedDaemonReload=no
JobTimeoutUSec=infinity
```

## rescue.target

Target de rescat però en un mode menys bàsic que emergency.target. Aquí es carregen més coses hi té dependència de sysinit.target. Observem-ho:

```
# cat /usr/lib/systemd/system/rescue.target
# This file is part of systemd.
#
# systemd is free software; you can redistribute it and/or modify it
# under the terms of the GNU Lesser General Public License as published by
# the Free Software Foundation; either version 2.1 of the License, or
# (at your option) any later version.

[Unit]
Description=Rescue Mode
Documentation=man:systemd.special(7)
Requires=sysinit.target rescue.service
After=sysinit.target rescue.service
AllowIsolate=yes

[Install]
Alias=kbrequest.target
```

### # systemctl list-dependencies rescue.target

```
rescue.target
├─rescue.service
├─systemd-update-utmp-runlevel.service
└─sysinit.target
   ├─dev-hugepages.mount
   ├─dev-mqueue.mount
   ├─dracut-shutdown.service
   ├─fedora-import-state.service
   ├─iscsi.service
   ├─kmod-static-nodes.service
   ├─ldconfig.service
   ├─lvm2-lvmetad.socket
   ├─lvm2-lvmpolld.socket
   ├─lvm2-monitor.service
   ├─multipathd.service
   ├─plymouth-read-write.service
   ├─plymouth-start.service
   ├─proc-sys-fs-binfmt_misc.automount
   ├─sys-fs-fuse-connections.mount
   ├─sys-kernel-config.mount
   ├─sys-kernel-debug.mount
   ├─systemd-ask-password-console.path
   ├─systemd-binfmt.service
   ├─systemd-firstboot.service
   ├─systemd-hwdb-update.service
   ├─systemd-journal-catalog-update.service
   ├─systemd-journal-flush.service
   ├─systemd-journald.service
   ├─systemd-machine-id-commit.service
   ├─systemd-modules-load.service
   ├─systemd-random-seed.service
   ├─systemd-sysctl.service
   ├─systemd-sysusers.service
   └─systemd-tmpfiles-setup-dev.service
```

### # systemctl show rescue.target

```
Id=rescue.target
Names=rescue.target runlevel1.target
Requires=sysinit.target rescue.service
Wants=systemd-update-utmp-runlevel.service
Conflicts=shutdown.target
ConflictedBy=multi-user.target graphical.target
Before=graphical.target multi-user.target initrd.target shutdown.target systemd-update-utmp-runlevel.service
After=sysinit.target rescue.service
Documentation=man:systemd.special(7)
Description=Rescue Mode
```

```

LoadState=loaded
ActiveState=inactive
SubState=dead
FragmentPath=/usr/lib/systemd/system/rescue.target
UnitFileState=disabled
UnitFilePreset=disabled
StateChangeTimestamp=Wed 2020-04-22 10:52:22 CEST
StateChangeTimestampMonotonic=3723171
InactiveExitTimestampMonotonic=0
ActiveEnterTimestampMonotonic=0
ActiveExitTimestampMonotonic=0
InactiveEnterTimestampMonotonic=0
CanStart=yes
CanStop=yes
CanReload=no
CanIsolate=yes
StopWhenUnneeded=no
RefuseManualStart=no
RefuseManualStop=no
AllowIsolate=yes
DefaultDependencies=yes
OnFailureJobMode=replace
IgnoreOnIsolate=no
NeedDaemonReload=no

```

## Altres targets de systemd

A part dels targets principals n'hi molts més, de intermitjos i de targets 'virtuals' fets per ser compatibles amb l'antiga arrencada amb init / runlevels.

Podem consultar el man de [systemd.special](#) per obtenir-ne un llistat:

SYSTEMD.SPECIAL(7)	systemd.special	SYSTEMD.SPECIAL(7)
<p><b>NAME</b></p> <p>systemd.special - Special systemd units</p>		
<p><b>SYNOPSIS</b></p> <p>basic.target, bluetooth.target, cryptsetup-pre.target, cryptsetup.target, ctrl-alt-del.target, default.target, emergency.target, exit.target, final.target, getty.target, graphical.target, halt.target, hibernate.target, hybrid-sleep.target, initrd-fs.target, initrd-root-device.target, initrd-root-fs.target, kbrequest.target, kexec.target, local-fs-pre.target, local-fs.target, machines.target, multi-user.target, network-online.target, network-pre.target, network.target, nss-lookup.target, nss-user-lookup.target, paths.target, poweroff.target, printer.target, reboot.target, remote-cryptsetup.target, remote-fs-pre.target, remote-fs.target, rescue.target, rpcbind.target, runlevel2.target, runlevel3.target, runlevel4.target, runlevel5.target, shutdown.target, sigpwr.target, sleep.target, slices.target, smartcard.target, sockets.target, sound.target, suspend.target, swap.target, sysinit.target, syslog.socket, system-update.target, time-sync.target, timers.target, umount.target,</p> <p>-.slice, system.slice, user.slice, machine.slice,</p> <p>dbus.service, dbus.socket, display-manager.service, system-update-cleanup.service</p>		
<p><b>DESCRIPTION</b></p> <p>A few units are treated specially by systemd. Many of them have special internal semantics and cannot be renamed, while others simply have a standard meaning and should be present on all systems.</p>		

Aquestes són algunes de les descripcions que el man en fa:

### basic.target

A special target unit covering basic boot-up. systemd automatically adds dependency of the type After= for this target unit to all services (except for those with DefaultDependencies=no).

Usually, this should pull-in all local mount points plus /var, /tmp and /var/tmp, swap devices, sockets, timers, path units and other basic initialization necessary for general purpose daemons. The mentioned mount points are special cased to allow them to be remote.

This target usually does not pull in any non-target units directly, but rather does so indirectly via other early boot targets. It is instead meant as a synchronization point for late boot services. Refer to bootup(7) for details on the targets involved.

**ctrl-alt-del.target**

systemd starts this target whenever Control+Alt+Del is pressed on the console. Usually, this should be aliased (symlinked) to reboot.target.

**default.target**

The default unit systemd starts at bootup. Usually, this should be aliased (symlinked) to multi-user.target or graphical.target.

The default unit systemd starts at bootup can be overridden with the systemd.unit= kernel command line option.

**emergency.target**

A special target unit that starts an emergency shell on the main console. This target does not pull in any services or mounts. It is the most minimal version of starting the system in order to acquire an interactive shell; the only processes running are usually just the system manager (PID 1) and the shell process. This unit is supposed to be used with the kernel command line option systemd.unit=; it is also used when a file system check on a required file system fails, and boot-up cannot continue. Compare with rescue.target, which serves a similar purpose, but also starts the most basic services and mounts all file systems.

Use the "systemd.unit=emergency.target" kernel command line option to boot into this mode. A short alias for this kernel command line option is "emergency", for compatibility with SysV.

In many ways booting into emergency.target is similar to the effect of booting with "init=/bin/sh" on the kernel command line, except that emergency mode provides you with the full system and service manager, and allows starting individual units in order to continue the boot process in steps.

**exit.target**

A special service unit for shutting down the system or user service manager. It is equivalent to poweroff.target on non-container systems, and also works in containers.

systemd will start this unit when it receives a request to shut down over D-Bus or a SIGTERM or SIGINT signal when running as user service daemon.

Normally, this (indirectly) pulls in shutdown.target, which in turn should be conflicted by all units that want to be scheduled for shutdown when the service manager starts to exit.

**graphical.target**

A special target unit for setting up a graphical login screen. This pulls in multi-user.target.

Units that are needed for graphical logins shall add Wants= dependencies for their unit to this unit (or multi-user.target) during installation. This is best configured via WantedBy=graphical.target in the unit's "[Install]" section.

**hibernate.target**

A special target unit for hibernating the system. This pulls in sleep.target.

**halt.target**

A special target unit for shutting down and halting the system. Note that this target is distinct from poweroff.target in that it generally really just halts the system rather than powering it down.

Applications wanting to halt the system should start this unit.

**multi-user.target**

A special target unit for setting up a multi-user system (non-graphical). This is pulled in by graphical.target.

Units that are needed for a multi-user system shall add Wants= dependencies for their unit to this unit during installation. This is best configured via WantedBy=multi-user.target in the unit's "[Install]" section.

**poweroff.target**

A special target unit for shutting down and powering off the system.

Applications wanting to power off the system should start this unit.

runlevel0.target is an alias for this target unit, for compatibility with SysV.

**reboot.target**

A special target unit for shutting down and rebooting the system.

Applications wanting to reboot the system should start this unit.

runlevel6.target is an alias for this target unit, for compatibility with SysV.

**rescue.target**

A special target unit that pulls in the base system (including system mounts) and spawns a rescue shell. Isolate to this target in order to administer the system in single-user mode with all file systems mounted but with no services running, except for the most basic. Compare with emergency.target, which is much more reduced and does not provide the file systems or most basic services.

runlevel1.target is an alias for this target unit, for compatibility with SysV.

Use the "systemd.unit=rescue.target" kernel command line option to boot into this mode. A short alias for this kernel command line option is "1", for compatibility with SysV.

**runlevel2.target, runlevel3.target, runlevel4.target, runlevel5.target**

These are targets that are called whenever the SysV compatibility code asks for runlevel 2, 3, 4, 5, respectively. It is a good idea to make this an alias for (i.e. symlink to) graphical.target (for runlevel 5) or multi-user.target (the others).

**shutdown.target**

A special target unit that terminates the services on system shutdown.

Services that shall be terminated on system shutdown shall add Conflicts= and Before= dependencies to this unit for their service unit, which is implicitly done when DefaultDependencies=yes is set (the default).

**sysinit.target**

systemd automatically adds dependencies of the types Requires= and After= for this target unit to all services (except for those with DefaultDependencies=no).

This target pulls in the services required for system initialization. System services pulled in by this target should declare DefaultDependencies=no and specify all their dependencies manually, including access to anything more than a read only root filesystem.

For details on the dependencies of this target, refer to `bootup(7)`

**Pràctica proposada:**

26. Mostra les dependències de `recue.target`: totes; targets; serveis.
27. Mostra les dependències de `emergency.target`: totes; targets; serveis.
28. Estableix com a target per defecte `reboot.target` i reinicia la màquina.
29. Punyeta! arregla-ho!

## Gràfics de l'arrencada i de les relacions de les unitats

En aquest apartat anem a repassar els conceptes vistos fins ara de l'arrencada del sistema amb systemd però introduint noves eines de systemd-`<ordre>` per veure gràfics i esquemes de l'arrencada.

**Atenció:**

- Per poder veure els gràfics de les seccions següents cal instal·lar [eog](#) i [graphviz](#).

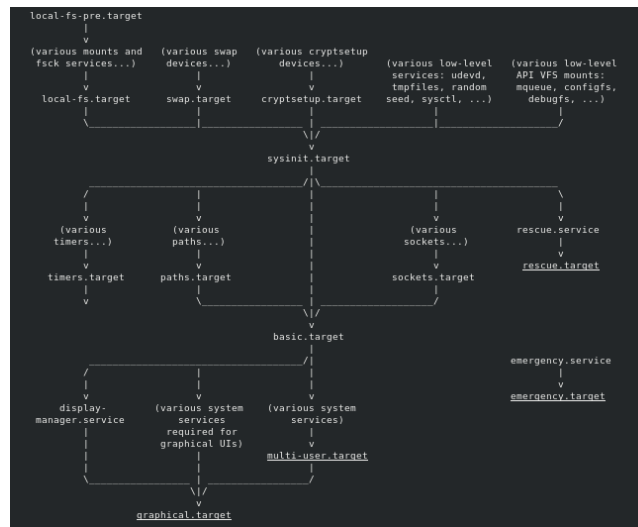
### Esquemes d'arrencada en mode text

Consultant l'ordre `man bootup` podem veure amb ascii art esquemes de l'arrencada del sistema amb systemd:

### Esquema d'arrencada simple sense ramdisk initramfs:

The following chart is a structural overview of these well-known units and their position in the boot-up logic. The arrows

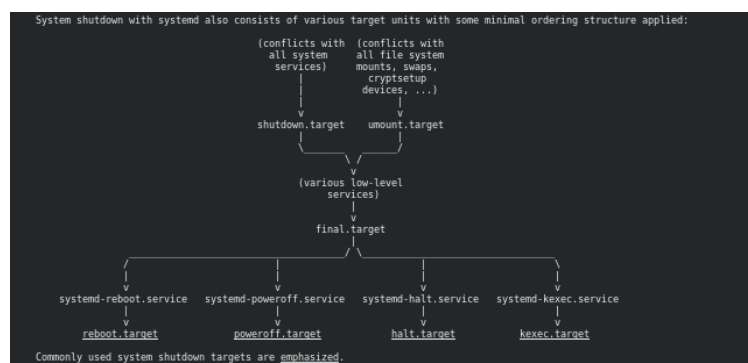
describe which units are pulled in and ordered before which other units. Units near the top are started before units nearer to the bottom of the chart.



Esquema d'arrancada amb initramfs:



Finalment podem veure l'esquema de com és l'apagada de la màquina:



## Systemctl list...

### List unit files

#### # systemctl list-unit-files | head

UNIT FILE	STATE
proc-sys-fs-binfmt_misc.automount	static
-.mount	generated
dev-hugepages.mount	static
dev-mqueue.mount	static
proc-fs-nfsd.mount	static
proc-sys-fs-binfmt_misc.mount	static
sys-fs-fuse-connections.mount	static
sys-kernel-config.mount	static
sys-kernel-debug.mount	static

### List units

#### # systemctl list-units | head

UNIT	LOAD	ACTIVE	SUB	DESCRIPTION
proc-sys-fs-binfmt_misc.automount		loaded	active	waiting Arbitrary Executable File Formats File System Automount Point
sys-devices-pci0000:00-0000:00:02.0-drm-card0-card0\x2deDP\x2d1-intel_backlight.device		loaded	active	plugged
/sys/devices/pci0000:00/0000:00:02.0/drm/card0/card0-eDP-1/intel_backlight				
sys-devices-pci0000:00-0000:00:14.0-usb1-1\x2d8-1\x2d8:1.0-bluetooth-hci0.device		loaded	active	plugged
/sys/devices/pci0000:00/0000:00:14.0/usb1/1-8/1-8:1.0/bluetooth/hci0				
sys-devices-pci0000:00-0000:00:17.0-ata2-host1-target1:0-0-1:0-0-0-block-sda-sda1.device		loaded	active	plugged ST500LM021-1KJ152
Reservado_para_el_sistema				
sys-devices-pci0000:00-0000:00:17.0-ata2-host1-target1:0-0-1:0-0-0-block-sda-sda2.device		loaded	active	plugged ST500LM021-1KJ152 SISTEMA
sys-devices-pci0000:00-0000:00:17.0-ata2-host1-target1:0-0-1:0-0-0-block-sda-sda3.device		loaded	active	plugged ST500LM021-1KJ152 DADES
sys-devices-pci0000:00-0000:00:17.0-ata2-host1-target1:0-0-1:0-0-0-block-sda-sda4.device		loaded	active	plugged ST500LM021-1KJ152 4
sys-devices-pci0000:00-0000:00:17.0-ata2-host1-target1:0-0-1:0-0-0-block-sda-sda5.device		loaded	active	plugged ST500LM021-1KJ152 5
sys-devices-pci0000:00-0000:00:17.0-ata2-host1-target1:0-0-1:0-0-0-block-sda-sda6.device		loaded	active	plugged ST500LM021-1KJ152 6

### List timers

#### \$ systemctl list-timers

NEXT	LEFT	LAST	PASSED	UNIT	ACTIVATES
Wed 2020-04-22 18:10:17 CEST	45min left	Wed 2020-04-22 17:10:12 CEST	14min ago	dnf-makecache.timer	dnf-makecache.service
Thu 2020-04-23 00:00:00 CEST	6h left	Wed 2020-04-22 00:00:38 CEST	17h ago	mlocate-updatedb.timer	mlocate-updatedb.service
Thu 2020-04-23 00:00:00 CEST	6h left	Wed 2020-04-22 00:00:38 CEST	17h ago	unbound-anchor.timer	unbound-anchor.service
Thu 2020-04-23 09:07:36 CEST	15h left	Wed 2020-04-22 09:07:36 CEST	8h ago	systemd-tmpfiles-clean.timer	systemd-tmpfiles-clean.service

4 timers listed.

Pass --all to see loaded but inactive timers, too.

### List sockets

#### \$ systemctl list-sockets

LISTEN	UNIT	ACTIVATES
/dev/rfkill	systemd-rfkill.socket	systemd-rfkill.service
/run/avahi-daemon/socket	avahi-daemon.socket	avahi-daemon.service
/run/dbus/system_bus_socket	dbus.socket	dbus.service
/run/dmeventd-client	dm-event.socket	dm-event.service
/run/dmeventd-server	dm-event.socket	dm-event.service
/run/lvm/lvmetad.socket	lvm2-lvmetad.socket	lvm2-lvmetad.service
/run/lvm/lvmpolld.socket	lvm2-lvmpolld.socket	lvm2-lvmpolld.service
/run/systemd/coredump	systemd-coredump.socket	
/run/systemd/initctl/fifo	systemd-initctl.socket	systemd-initctl.service
/run/systemd/journal/dev-log	systemd-journal-dev-log.socket	systemd-journald.service
/run/systemd/journal/socket	systemd-journald.socket	systemd-journald.service
/run/systemd/journal/stdout	systemd-journald.socket	systemd-journald.service
/run/udev/control	systemd-udev-control.socket	systemd-udev.service
/var/run/.heim_org.h5l.kcm-socket	sssd-kcm.socket	sssd-kcm.service
/var/run/cups/cups.sock	cups.socket	cups.service
/var/run/secrets.socket	sssd-secrets.socket	sssd-secrets.service
@ISCSIADM_ABSTRACT_NAMESPACES	iscsid.socket	iscsid.service
@ISCSID_UIP_ABSTRACT_NAMESPACES	iscsiuio.socket	iscsiuio.service
audit 1	systemd-journald-audit.socket	systemd-journald.service
kobject-uevent 1	systemd-udev-kernel.socket	systemd-udev.service

20 sockets listed.

Pass --all to see loaded but inactive sockets, too.



## System-analyze

Quan el sistema triga en arrencar ens posem nervisos.... Com ho podem fer per buscar qui és el culpable d'una arrencada lenta del sistema? Tenim diverses eines:

- ☐ `systemd -analyze blame`
- ☐ `systemd-analyze time`
- ☐ `systemd -analyze critical-chain`
- ☐ `systemd -analyze plot`
- ☐ `Systemd-analyze dot`

**systemd-analyze** may be used to determine system boot-up performance statistics and retrieve other state and tracing information from the system and service manager, and to verify the correctness of unit files.

## Systemd-analyze blame

**systemd-analyze blame** prints a list of all running units, ordered by the time they took to initialize. This information may be used to optimize boot-up times. Note that the output might be misleading as the initialization of one service might be slow simply because it waits for the initialization of another service to complete.

Aquesta eina proporciona un llistat dels serveis i els temps d'arrancada. El plymouth és l'aplet gràfic de Fedora en l'arrancada (actualment la rodoneta que es va omplint). Hi és des del principi fins al final, en aquest exemple hi és durant 17.3 segons. Podem veure que un dels elements que tarda força a iniciar-se és la xarxa. Això té repercussió en tots aquells serveis que en depenen.

```
# systemd-analyze blame
17.330s plymouth-quit-wait.service
7.753s NetworkManager-wait-online.service
6.856s lvm2-monitor.service
6.064s systemd-udev-settle.service
5.368s firewalld.service
5.139s dev-sda7.device
4.758s udisks2.service
4.263s initrd-switch-root.service
4.069s vboxdrv.service
3.203s sssd.service
3.070s systemd-journal-flush.service
2.817s systemd-tmpfiles-clean.service
2.331s dnf-makecache.service
2.088s ModemManager.service
2.025s abrt.service
1.814s polkit.service
1.804s chronyd.service
1.682s systemd-udev.service
1.579s systemd-tmpfiles-setup-dev.service
1.553s dracut-initqueue.service
1.547s packagekit.service
1.422s fwupd.service
1.396s switcheroo-control.service
1.383s gssproxy.service
1.376s bluetooth.service
1.368s cups.service
1.367s rtkit-daemon.service
1.009s avahi-daemon.service
939ms accounts-daemon.service
719ms systemd-vconsole-setup.service
426ms fedora-readonly.service
```

```
385ms user@42.service
343ms NetworkManager.service
334ms auditd.service
```

## Systemd-analyze time

**systemd-analyze time** prints the time spent in the kernel before userspace has been reached, the time spent in the initial RAM disk (initrd) before normal system userspace has been reached, and the time normal system userspace took to initialize. Note that these measurements simply measure the time passed up to the point where all system services have been spawned, but not necessarily until they fully finished initialization or the disk is idle.

### # systemd-analyze time

startup finished in 1.634s (kernel) + 3.421s (initrd) + 35.551s (userspace) = 40.607s

## Systemd-analyze plot

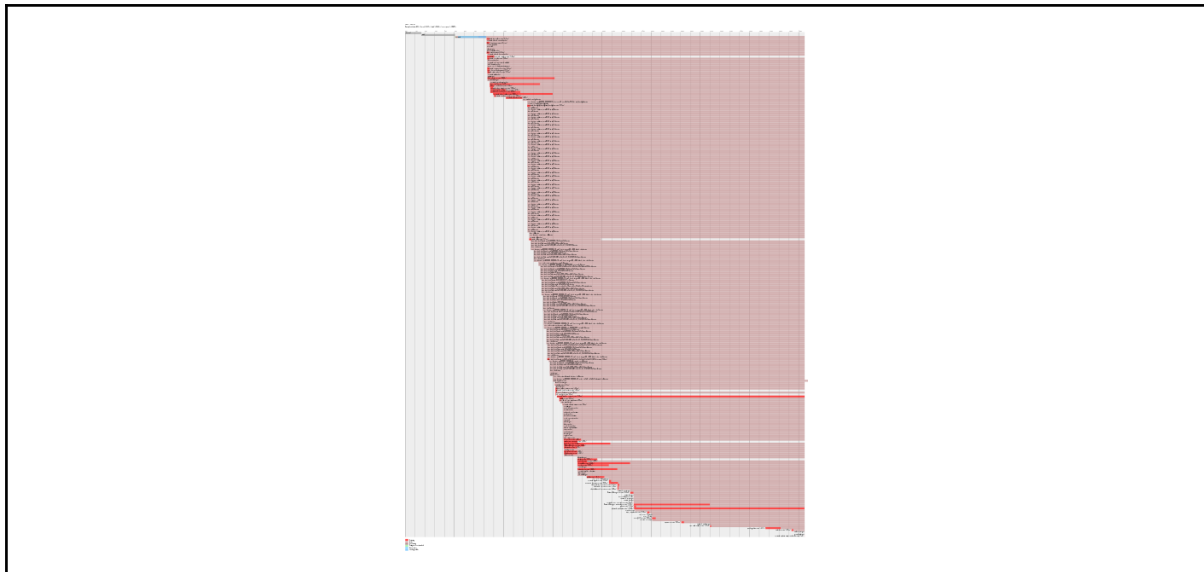
**systemd-analyze plot** prints an SVG graphic detailing which system services have been started at what time, highlighting the time they spent on initialization.

```
# systemd-analyze plot > /tmp/file.svg
# eog /tmp/file.svg
```

Podem observar un mapa (de fet un gran llençol) amb els temps d'engegar cada cosa. Observeu la llegenda de colors per saber-ne el significat.



Si teniu molt bona vista podeu veure TOTA l'arrancada aquí:



## Systemd-analyze crital-chain

**systemd-analyze critical-chain [UNIT...]** prints a tree of the time-critical chain of units (for each of the specified UNITS or for the default target otherwise). The time after the unit is active or started is printed after the "@" character. The time the unit takes to start is printed after the "+" character. Note that the output might be misleading as the initialization of one service might depend on socket activation and because of the parallel execution of units.

### # systemd-analyze critical-chain sshd.service

The time after the unit is active or started is printed after the "@" character.  
The time the unit takes to start is printed after the "+" character.

```
sshd.service +118ms
└─network.target @776ms
   └─wpa_supplicant.service @16.384s +10ms
      └─dbus.service @634ms
         └─basic.target @628ms
            └─sockets.target @628ms
               └─iscsiio.socket @628ms
                  └─sysinit.target @626ms
                     └─systemd-update-utmp.service @622ms +4ms
                        └─auditd.service @586ms +35ms
                           └─systemd-tmpfiles-setup.service @542ms +29ms
                              └─fedora-import-state.service @519ms +18ms
                                 └─local-fs.target @506ms
                                    └─run-user-42.mount @13.123s
                                       └─local-fs-pre.target @506ms
                                          └─lvm2-monitor.service @251ms +251ms
                                             └─lvm2-lvm2metad.service @268ms
                                                └─lvm2-lvm2metad.socket @251ms
                                                   └─.slice
```

### # systemd-analyze critical-chain basic.target

The time after the unit is active or started is printed after the "@" character.  
The time the unit takes to start is printed after the "+" character.

```
basic.target @628ms
└─sockets.target @628ms
   └─iscsiio.socket @628ms
      └─sysinit.target @626ms
         └─systemd-update-utmp.service @622ms +4ms
            └─auditd.service @586ms +35ms
               └─systemd-tmpfiles-setup.service @542ms +29ms
                  └─fedora-import-state.service @519ms +18ms
                     └─local-fs.target @506ms
                        └─run-user-42.mount @13.123s
                           └─local-fs-pre.target @506ms
                              └─lvm2-monitor.service @251ms +251ms
                                 └─lvm2-lvm2metad.service @268ms
```

```
└─lvmetad.socket @251ms
  └─.slice
```

## Systemd-analyse dot

**systemd-analyze dot** generates textual dependency graph description in dot format for further processing with the GraphViz dot(1) tool. Use a command line like **systemd-analyze dot | dot -Tsvg > systemd.svg** to generate a graphical dependency tree. Unless **--order** or **--require** is passed, the generated graph will show both ordering and requirement dependencies. Optional pattern globbing style specifications (e.g. **\*.target**) may be given at the end. A unit dependency is included in the graph if any of these patterns match either the origin or destination node.

```
# systemd-analyze dot gpm.service | dot -Tsvg > gpm.service.svg
Color legend: black    = Requires
                dark blue = Requisite
                dark grey = Wants
                red      = Conflicts
                green    = After

$ eog gpm.service.dot
```

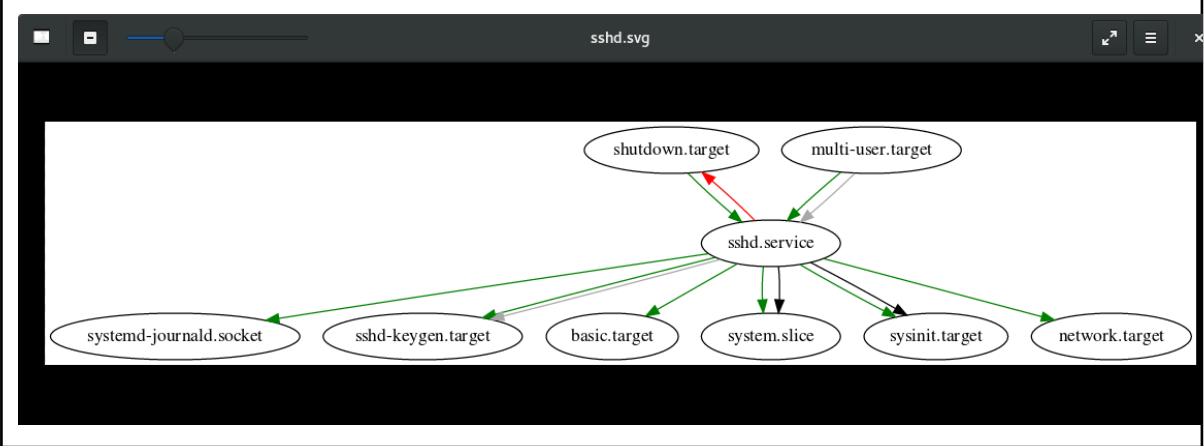
```
# systemd-analyze dot gpm.service | dot -Tsvg > gpm.service.svg
Color legend: black    = Requires
                dark blue = Requisite
                dark grey = Wants
                red      = Conflicts
                green    = After

$ eog gpm.service.dot
```

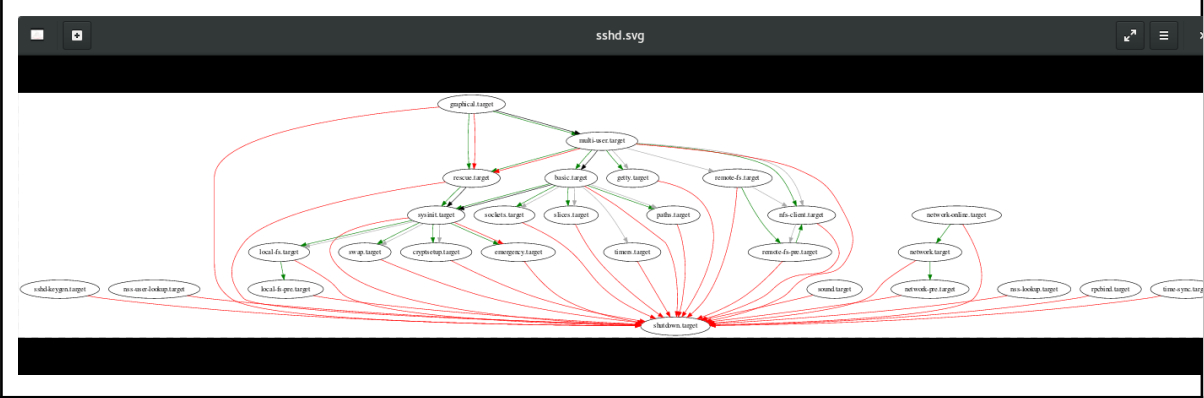
```
# systemd-analyze dot gpm.service | dot -Tsvg > gpm.service.svg
Color legend: black    = Requires
                dark blue = Requisite
                dark grey = Wants
                red      = Conflicts
                green    = After

$ eog gpm.service.dot
```

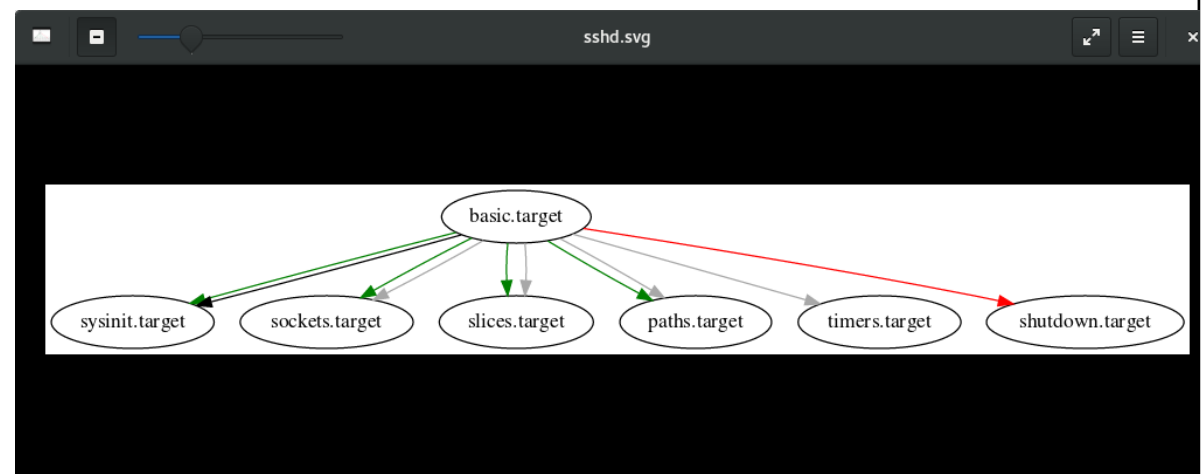
```
# systemd-analyze dot sshd.service | dot -Tsvg > /tmp/sshd.svg ; eog /tmp/sshd.svg
```



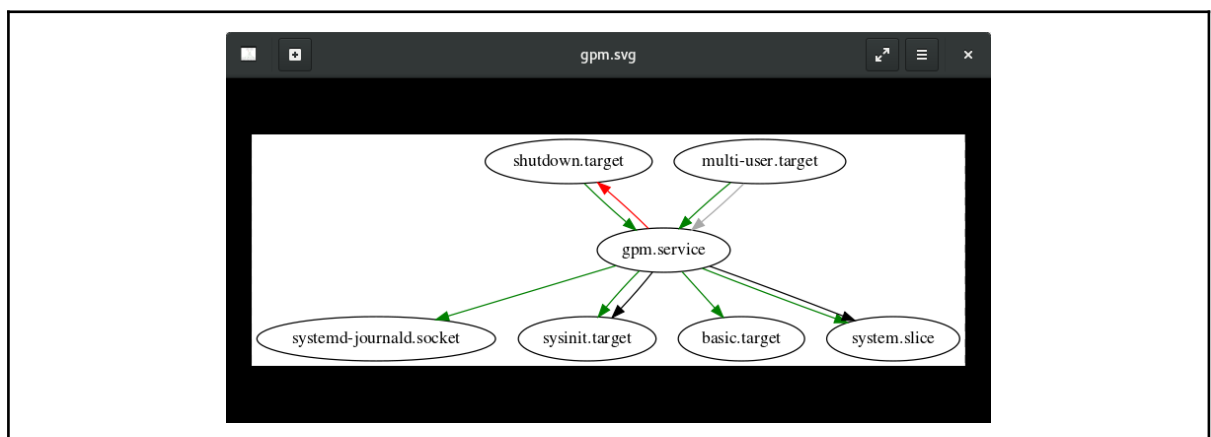
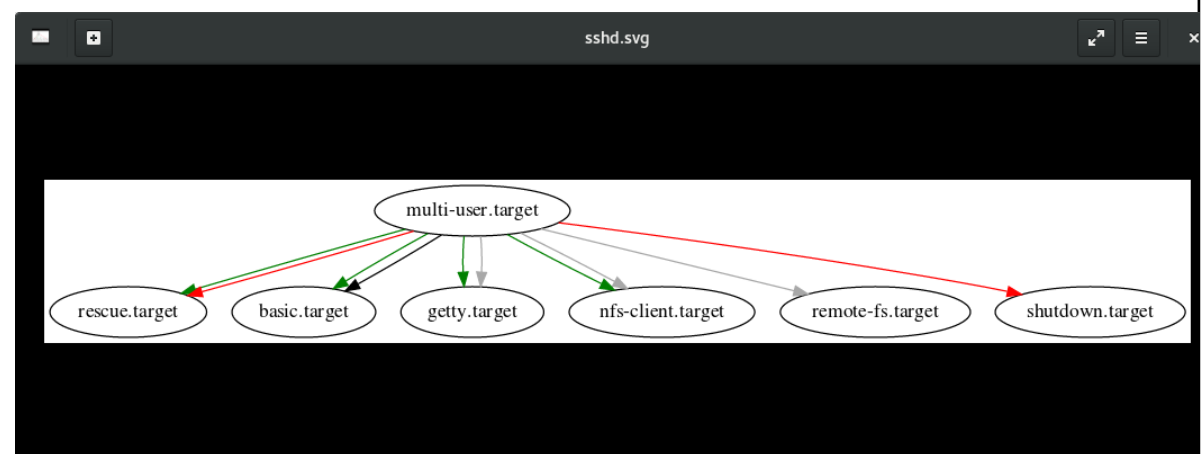
```
# systemd-analyze dot --to-pattern='*.target' --from-pattern='*.target' | dot -Tsvg > /tmp/sshd.svg ; eog /tmp/sshd.svg
```



```
# systemd-analyze dot --to-pattern='*.target' --from-pattern='basic.target' | dot -Tsvg > /tmp/sshd.svg ; eog /tmp/sshd.svg
```



```
# systemd-analyze dot --to-pattern='*.target' --from-pattern='multi-user.target' | dot -Tsvg > /tmp/sshd.svg ; eog /tmp/sshd.svg
```



---

## Crear units propies: targets i services

---

Exercici pràctic:

- Fer un servei [myhello.service](#) on generem un missatge de benvinguda. Crear el `.service` de tipus [oneshot](#) que significa que s'executa un cop i finalitza. Observeu que s'ha comentat la línia [StandardOutput](#) perquè el `echo` no vagi al jorural sinó a la pantalla. El servei s'instal·larà en un target personalitzat anomenat [mytarget.target](#).

```
[Unit]
Description=El meu primer servei

[Service]
Type=oneshot
ExecStart=/usr/bin/echo "bye bye champions"
RemainAfterExit=true
ExecStop=/usr/bin/echo "liverpool"
#StandardOutput=journal

[Install]
WantedBy=mytarget.target
```

- Fer un servei [myweb.service](#) on creen un servei web basat en el servei de `httpd`. Cal instal·lar prèviament el paquet `httpd`. Crear-lo de tipus [fork](#) engegant l'executable de `httpd`. El servei s'instal·larà en un target personalitzat anomenat [mytarget.target](#).

```
[Unit]
Description=My web server

[Service]
Type=forking
ExecStart=/usr/sbin/httpd
ExecStop=/usr/bin/echo "aturant web server myweb"
ExecStop=/usr/bin/pkill httpd
StandardOutput=journal

[Install]
WantedBy=mytarget.target
```

- Fer un target d'arrencada personalitzat anomenat [mytarget.target](#) basat en [rescue.target](#) que tingui el servei [gpm](#), [myhello](#) i el [myweb](#).

```
[Unit]
Description=El meu mode de rescat mytarget
Documentation=man:systemd.special(7)
Requires=sysinit.target rescue.service
After=sysinit.target rescue.service
AllowIsolate=yes
```

```
[Install]
Alias=rescat.target
```

- Pensar a fer el [systemctl daemon-reload](#). Sempre que es modifiquen els units (el contingut dels fitxers dels units) cal fer `systemctl daemon-reload` perquè systemd recarregi la configuració de les units.
- Fer l'enable dels serveis [myhelo](#) i [myweb](#). Verificar amb `list dependencies` les dependències de [mytarget.target](#). Llistar el seu directori [.wants](#).
- Reboot i iniciar el sistema indicant en el GRUB el target [mytarget.target](#). Verificar amb [is-active](#) i [list-dependencies](#) que està actiu i què no està actiu.
- Repetir-ho però en lloc de usar [myweb.service](#) (apagar i disable de myweb) usar [myssh.service](#) o [myhttp.service](#), tunejant els serveis originals http i sshd. Recordeu de fer el [daemon-reload](#). Observar que si es requereix de xarxa per funcionar systemd 'estira' de ella.

# Under Construction

Practicar treballar amb:

- ☐ Targets
- ☐ Isolate
- ☐ Targets definits per l'usuari
- ☐ Snapshots

Pràctiques proposades:

1. **Snapshot:** generar un snapshot amb tres serveis activats i observar que estan actius o no segons si s'està al target del snapshot o no.
2. **Service WantedBy:** lligar un servei a més d'un target.
3. **Target propi:** Generar un target propi basat en multi-user.target que carregi un parell de serveis propis del target.
4. **Serveis propis:** Crear serveis propis que facin accions trivils per practicar systemd.

## Snapshot

- Al target multi-user.target assegurar-se de tenir disabled i stoped els serveis gpm.service, xinetd.service i vsftpd.service.
- Activar aquests tres serveis amb start (sense fer el enable).
- Definir un sanpshot de nom test-serveis.snapshot.
- Com a mesura extra activar el target rescue.target amb isolate. Observar que els tres serveis indicats estan apagats i disabled.
- Activar el target del sanpshot amb isolate. Observar que ara que els tres serveis estan actius amb l'ordre show. Observar que el target test-serveis.snapshot també està actiu.
- Activar el target multi-user.target amb isolate i observar que els tres serveis tornen a star stop. Verificar la configuració amb ordres is-active, is-enable, status i show.
- Llistar les característiques del snapshot amb l'opció show.

## Service WantedBy

- Activar el servei multi-user.target amb isolate.
- Editar el servei gpm.service i xinetd.service i afegir a la secció installation: WantedBy=rescue.target. Ara els serveis son requerits tant per el target multi-user.target com per rescue.target.
- Si cal recordar de fer el systemctl daemon-reload.
- Fer disable / enable dels serveis gpm.service i xinetd.service. Observar que es creen els symlinks i que s'esborren. Dos en cada cas.
- Deixar els dos serveis enabled i observar el llistat de /etc/systemd/system/rescue.target.wants on ara apareixen els dos serveis enabled.
- Activar el target rescue.target amb isolate. Observar amb status que els dos serveis son actius. Verificar també amb is-active i is-enabled.
- Activar el servei multi-user.target i repetir les comprovacions fetes en el cas de



rescue.target.

- Editar els dos fitxers de servei i restablir la configuració inicial. Si cal recordar de fer el `systemctl daemon-reload`.

### Crear target propi

- Crear un target propi anomenat `test-target.target` que es basa amb `multi-user.target` mes els serveis `test-gpm.service` i `test-xinetd.service`.
- Copiar `graphical.target` de `/usr/lib/systemd/system` al mateix directori amb el nom `test-target.target`. Eliminar el `wants`.
- Copiar els serveis `gpm.service` i `xinetd.service` (atmbé a `/usr/lib/systemd/system`) i anomenar-los `test-gpm.service` i `test-xinetd.service`. Editar-los per indicar que són únicament `WantedBy=test-target.target`.
- Fer enable dels nous serveis `test-gpm.target` i `test-xinetd.target`. Observar el llistat de `/etc/systemd/system/test-target.wants` on ara apareixen els dos serveis que requereix el nou target. Els serveis no estan encara start, però si enabled, verificar-ho amb `is-active` i `is-enabled`.
- Activar el target `test-target.target` amb `isolate` i observar amb `is-active` i `list-dependencies` que s'està a aquest target i que els serveis son active i el llistat de dependències.
- Activar el mode `multi-user.target` i observar que els serveis `test-gpm.service` i `test-xinetd.service` no són actius. Verificar-ho amb `is-active`.

### Crear serveis propis

- Service (on-shot, ....)
- mount
- path

---

<http://fedoraproject.org/wiki/Systemd>

<http://0pointer.de/blog/projects/systemd-docs.html>

[http://fedoraproject.org/wiki/SysVinit\\_to\\_Systemd\\_Cheatsheet](http://fedoraproject.org/wiki/SysVinit_to_Systemd_Cheatsheet)

---