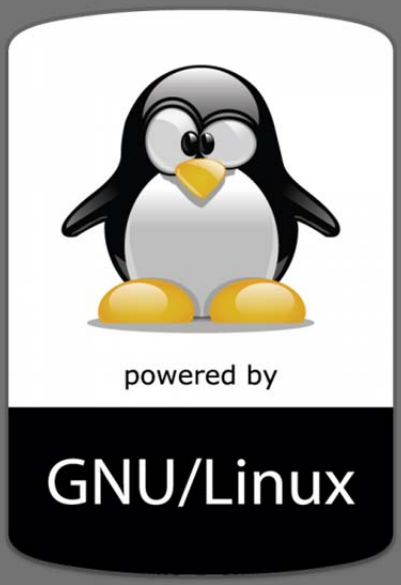
2016

SystemD جایگزین init در لینوکس

(ویرایش اول)

نویسنده حسام الدین توحید



skywan13@chmail.ir


بسم الله الرحمن الرحیم

مقدمه مولف :

آنچه پیش رو دارید ویرایش **اول** مقاله systemD جایگزین init در لینوکس است که به صورت رایگان به علاقه مندان لینوکس هدیه می گردد . در تهیه این مقاله از سر فصل های درسی گفته شده در دوره های LPIc2 و RHCE استفاده شده ولازم می دانم از **جعفر عقربی** به خاطر راهنمایی های مفیدشان و همچنین **استاد پوررنجبر** تشکر کافی را داشته باشم. این مطلب با نگاهی کاربردی و بدون پرداختن به بحث های تئوریک گردآوری و عرضه شده که امید است با توجه به کمبود مطالب فارسی در مورد systemD ، این مقاله بتواند باعث ارتقاء دانش فنی کاربران لینوکس و متخصصین این حوزه از IT شود.

موفق باشید

حسام الدین توحید

تیر 1395

فهرست

Systemd vs SysVinit

Systemd Commands: http://linoxide.com/linux-command/linux-systemd-commands/

Service Related Commands

Comments	SysVinit	Systemd
Start a service	service dummy start	systemctl start dummy.service
Stop a service	service dummy stop	systemctl stop dummy.service
Restart a service	service dummy restart	systemctl restart dummy.service
Reload a service	service dummy reload	systemctl reload dummy.service
Service status	service dummy status	systemctl status dummy.service
Restart a service if already running	service dummy condrestart	systemctl condrestart dummy.service
Enable service at startup	chkconfig dummy on	systemctl enable dummy.service
Disable service at startup	chkconfig dummy off	systemctl disable dummy.service
Check if a service is enabled at startup	chkconfig dummy	systemctl is-enabled dummy.service
Create a new service file or modify configuration	chkconfig dummy --add	systemctl daemon-reload

Note : New version of systemd support "systemctl start dummy" format.

Runlevels

Comments	SysVinit	Systemd
System halt	0	runlevel0.target, poweroff.target
Single user mode	1, s, single	runlevel1.target, rescue.target
Multi user	2	runlevel2.target, multi-user.target
Multi user with Network	3	runlevel3.target, multi-user.target
Experimental	4	runlevel4.target, multi-user.target
Multi user, with network, graphical mode	5	runlevel5.target, graphical.target
Reboot	6	runlevel6.target, reboot.target
Emergency Shell	emergency	emergency.target
Change to multi user runlevel/target	telinit 3	systemctl isolate multi-user.target (OR systemctl isolate runlevel3.target)
Set multi-user target on next boot	sed s/^id:.*:initdefault:/ id:3:initdefault:/	ln -sf /lib/systemd/system/multi-user.target /etc/systemd/system/default.target
Check current runlevel	runlevel	systemctl get-default
Change default runlevel	sed s/^id:.*:initdefault:/ id:3:initdefault:/	systemctl set-default multi-user.target

Miscellaneous Commands

Comments	SysVinit	Systemd
System halt	halt	systemctl halt
Power off the system	poweroff	systemctl poweroff
Restart the system	reboot	systemctl reboot
Suspend the system	pm-suspend	systemctl suspend
Hibernate	pm-hibernate	systemctl hibernate
Follow the system log file	tail -f /var/log/messages or tail -f /var/log/syslog	journalctl -f

Systemd New Commands

Comments	Systemd
Execute a systemd command on remote host	systemctl dummy.service start -H user@host
Check boot time	systemd-analyze or systemd-analyze time
Kill all processes related to a service	systemctl kill dummy
Get logs for events for today	journalctl --since=today
Hostname and other host related information	hostnamectl
Date and time of system with timezone and other information	timedatectl

Brought to you by LinOxide Team

LINOXIDE Linux Freedom Thoughts

systemD جایگزین init در لینوکس

گذری بر init

بطور سنتی و تا به امروز در بسیاری از توزیع‌های لینوکس، نخستین پروسسی که در طی فرایند بوت شدن سیستم، توسط Kernel ایجاد می‌شود Init می‌باشد. init یک فرایند پس زمینه یا یک Background Process (Daemon) است که با PID=1 در پس زمینه همواره در حال اجرا بوده و دیگر پروسس‌ها را کنترل می‌کند. Init در واقع Parent تمامی فرایندهای درون سیستم است و مخفف کلمه initialization بوده که در ترجمه فارسی به معنی مقدار دهی اولیه می‌باشد. init پردازشی است که همه پردازش های مورد نیاز سیستم عامل را فراخوانی و اجرا می‌کند.

Init نخستین فرایندی است که ایجاد شده و تا زمانی که سیستمShutdown نشود در ram سیستم باقی می‌ماند و آخرین فرایندی است که از بین می رود اگر به هر دلیلی init نتواند کار خود را آغاز کند هیچ فرایند دیگری هم نمی‌تواند اجرا شود و سیستم به مرحله ای می‌رسد که به آن "**kernel panic**" می‌گویند. Init رایج ترین مرجع برای **System V init** است. SystemV اولین سیستم عامل یونیکس تجاری بود که طراحی شد. کاربردهای init در بسیاری از توزیع‌های لینوکس امروزی مشابه سیستم عامل SystemV است با موارد استثنایی همچون Slackware که از BSD-style و Gentoo که از custom init استفاده می‌کند.

تا قبل از ارائه systemD مخزن لینوکس دایرکتوری init.d بود و تمامی فایلهایی که در دایرکتوریهای rc قرار داشتند به دایرکتوری init.d لینک می‌شدند. در سیستم‌های مبتنی بر init به محض اینکه یک سرویس در لینوکس نصب می‌شود یک

اثر از خودش در init.d به جا می‌گذارد. در نسخه‌های قدیمی لینوکس init فقط یک فایل پیکربندی داشت که گاها به چند صد خط می‌رسید. یعنی تمام فایل‌هایی که در دایرکتوریهای rc قرار دارند در یک فایل کانفیگ بزرگ جمع شده بودند که این خود یک ضعف و آسیب پذیری به شمار می‌آمد، چون اگر برای این فایل مشکلی پیش می‌آمد سیستم up نمی‌شد. لذا در نسخه‌های بعدی آن فایل بزرگ چند صد خطی به تعداد زیادی فایل تبدیل شد تا مدیریت و پایداری سیستم تضمین بشود.

init اولین نرم‌افزاری است که در user mode کار کرده و به صورت event base عمل می‌کند یعنی یک فرایند فقط زمانی شروع می شود که فرایند قبلی‌اش شروع و با موفقیت به اتمام رسیده باشد. این سریالی عمل کردن init در اجرای فرایندها یکی از ضعف‌های است که باعث کند شدن روند بوت سیستم‌های مبتنی بر init می‌شود.

اولین کاری که init انجام می‌دهد این است که محتویات فایل تنظیمات خودش یا initialization file ای که در مسیر/etc/inittab وجود دارد را بخواند. محتویات این فایل به init می‌گوید که یک اسکریپت تنظیمات اولیه محیطی یا environment configuration script را اجرا کند که در این اسکریپت تعیین مسیرها یا pathها ، فرآیند Swapping ، بررسی فایل سیستم و ... انجام می‌شود. تقریبا می‌توان گفت اجرای این اسکریپت هر چیزی که سیستم شما نیاز دارد تا فرآیند مقداردهی اولیه یا همان initialization را انجام دهد ، شامل می‌شود ، حتی تعیین کردن ساعت سیستم، پورتهای سریال و ... همه در این مرحله انجام می‌شود.

در ادمه این فایل به init اعلام می‌کند که سیستم قرار است چگونه و در چه run level پیکربندی شود . یکی از مهم ترین وظایف پروسه init تعیین نوع اجرای لینوکس یا تعیین سطح اجرایی کاربری لینوکس(RunLevel) است که شما آن را روی صفحه نمایش می‌بینید. همانطور که گفته شد در فرآیند init بر اساس تنظیمات فایل inittab یک run level برای سیستم انتخاب می‌شود که بر حسب run level انتخابی یک سری اسکریپت از پوشه های مورد نظر اجرا می‌شوند.

run level در واقع پیکربندی فرآیندها یا پردازش‌های موجود در سیستم است. بعد از اینکه run level پیشفرض برای سیستم در نظر گرفته شد ، init با توجه به محتویات دایرکتوری rc ای که به run level مربوطه اختصاص یافته تمامی پردازش‌ها یاProcess های پس زمینه‌ای که برای اجرا شدن سیستم مورد نیاز هستند را اجرا می‌کند. Init هر کدام از اسکریپت های kill را که با حرف K شروع می‌شوند را با یک پارامتر stop و هر کدام از اسکریپت های Start را که با S شروع می‌شوند را اجرا و سرویس‌ها و برنامه‌های کاربردی که درrun level مربوطه وجود خواهند داشت را اجرایی خواهد کرد. اسکریپت های زیر دایرکتوری/etc/init.d/ توسط این فرایند کنترل می‌شوند و می‌توان از دستور service برای کنترل آنها استفاده نمود.

همچنین در زیر دایرکتوری‌های /etc/rc<x>.d/ فایل‌هایی وجود دارند که لینک‌هایی از فایل‌های درون دایرکتوری /etc/init.d/ می‌باشند، این فایل‌ها یا همان لینک‌ها، سرویس‌های موجود در زمان ورود به یک سطح اجرایی را مشخص می‌کنند. هیچکدام از اسکریپت‌هایی که در پوشه /etc/rc<x>.d/ قرار گرفته‌اند سرویس‌ها را stop یا start نمی‌کنند

بلکه تمامی فایل‌هایی که در پوشه /etc/rc<x>.d/ قرار گرفته‌اند به عنوان یک لینک به اسکریپت‌هایی که در پوشه /etc/init.d/ قرار دارند اشاره می‌کنند، به اینگونه لینک ها symbolic link گفته می‌شود ، یکsymbolic link چیزی بیشتر از یک فایل که به یک فایل دیگر اشاره می‌کند نیست و زمانی استفاده می‌شود که شما می‌خواهید بدون ایجاد کردن و یا حذف کردن یک فایل اسکریپت آن را اجرا و سرویس‌ها را start و یا stop کنید.

startup scriptهای مربوط به init در پوشه/etc/rc.d/ قرار گرفته‌اند اما اسکریپت‌هایی که برای run level ها استفاده می‌شوند درsubdirectory هایی به شکل /etc/rc.d/rc0.d/ **تا** /etc/rc.d/rc6.d/ بر اساس انتخاب init از بین run level های 0 تا 6 اجرا می‌شوند. در آخر نیز init هر چیزی که در پوشه /etc/rc.d/rc.local/ پیدا می‌کند را فارق از اینکه در چهrun level ای قرار دارد اجرا خواهد کرد.

توجه کنید که در زمان startup سیستم معمولا دو اسکریپت rc2.d و rc3.d اجرا می‌شوند. در این حالت هیچ سرویسی در حالت stopped قرار نمی‌گیرد یا حداقل می توان گفت بصورت دائمی در حالت stopped قرار نمی‌گیرد.

در نهایت اگر همه چیز درست باشد اسکریپت‌های مربوط به run level مورد نظر انتخاب و اجرا خواهند شد و سیستم به سراغ مرحله بعدی می‌رود.

لزوم جایگزینی init با چیزی کامل‌تر، از مدت‌ها احساس می‌شد و جایگزین‌های مختلفی مرحله به مرحله توسعه داده شد، که برخی جایگزین native init توزیع شدند، بعضی از آنها عبارتند از:

Upstart: یک سرویس جایگزین init که در **اوبونتو** اجرا شد و برای شروع پروسه غیر همزمان طراحی شده است.

Epoch: یک سرویس جایگزین init که از همه طرف سادگی و مدیریت سرویس ایجاد کرده است، این برای شروع پروسه single thread طراحی شده است.

Mudar: یک سرویس جایگزین init که به زبان پایتون نوشته شده است، در پاردوس گنو/لینوکس اجرا شده و برای شروع پروسه های غیر همزمان طراحی شده است.

و در نهایت **systemd** ، یک سرویس جایگزین init که برای شروع پروسه های موازی طراحی شده و در تعدادی از توزیع های استاندارد Fedora, OpenSuSE, Arch, RHEL, CentOS وغیره اجرا شده است، پروسه init به راحتی آغاز می‌شود، یک task درست بلافاصله بعد از آخرین task ای که با موفقیت در startup است، شروع می‌شود و در حافظه Load می‌گردد. این اغلب به تاخیر و طولانی شدن مدت زمان بوت، منجر می‌شود. با این حال، systemd برای سرعت بخشیدن به انجام کارها از تمام تاخیرهای غیر ضروری اجتناب می‌کند.

البته بسیاری از توزیع های لینوکسی هنوز هم از سبک Unix System V پیروی می‌کنند ولی برخی مانندSlackware از سبک BSD طبعیت کرده‌اند و برخی هم مانند Gentoo سبک سفارشی خودشان را دارند. در ادامه با توجه به فراگیر شدن systemD بر بعضی از مفاهیم و دستورات آن مروری خواهیم داشت.

مقدمه ای بر SystemD

systemD یک Service Manager و System Manager برای سیستم عامل لینوکس است که کار توسعه و جایگزینی آن به جای init از مارس سال ۲۰۱۰ در شرکت Red Hat توسط دو برنامه نویس آلمانی به نام های Lennart Poettering و Kay Sievers شروع شد و با ارائه سری هفت لینوکس Red Hat وارد سیستم عامل های شبکه ای گردید. اوج فراگیر شدن آن هم از سری هفت RHEL است.

systemD یک مجموعه پیچیده ودرهم تنیده از هزاران باینری است که به شدت به هم پیوند خورده‌اند و مطابق قرار داد یونیکس با اضافه کردن حرف d به آخر آن نامگذاری شده است.

این پروسس به صورت مستقیم یا غیر مستقیم Parent همه فرایندهای دیگر سیستم می‌باشد و چون نخستین فرایندی است که در سیستم ایجاد می‌شود از این رو معمولا به آن **"pid=1"**اختصاص داده می‌شود و مانند init نخستین فرایندی است که ایجاد و در Ram بارگزاری شده و آخرین فرایندی است که پایان می‌یابد.

systemD بر خلاف فلسفه یونیکس عمل می‌کند که می‌گوید:" یک کار انجام بده ولی آن را درست انجام بده"

زیرا systemD کاری بیشتر از استارت برنامه‌های اصلی انجام می‌دهد و مسئولیت‌های آن به طور فزاینده‌ایی از حیطه یک init system به بیرون تجاوز کرده به طوری که مدیریت قدرت، مدیریت دیوایس، mount pointها، رمزگذاری دیسک، دادن سوکت به API/inetd ، پیکربندی شبکه، مدیریت logging و session ها، پیدا کردن پارتیشن GPT ، مدیریت زمان، محل و hostname، استارت logging سیستم، Network Stack، برنامه‌ریزی کارها به صورت cron-style، لاگین‌های کاربران و بسیاری از کارهای دیگر را عهده دارد می‌شود و البته systemD بر خلاف init ، کنترل بهتری را بر روی فرایندها دارد.

systemD با اجرای فرایندهای دیگر به صورت موازی باعث کاهش زمان بوت، Over Head محاسباتی کمتر و برتری‌های دیگری نسبت به init شده است. داشتن اسکریپت‌های کمتر و توانایی در اجرای Task های هم زمان بیشتر به صورت Parallel باعث شده نسبت به init سرعت بالاتری داشته باشد. این روند ممکن است برای کاربران و admin های سیستم خوشایند باشد اما بعضی از توسعه دهندگان لینوکس از آن متنفرند.

با وجود اینکه systemD برای غلبه بر کاستی‌های init طراحی شده، اما منتقدین اعتقاد دارند که با سایر اعضاء خانواده یونیکس سازگار نیست. آن ها همین طور بر طراحی «یکپارچه و گرایش شدید به دسکتاپ»، که آن را به یک انتخاب ضعیف برای بسیاری از موارد مورد استفاده لینوکس تبدیل می‌کند، ایراد می‌گیرند. فایل های جورنال systemD، که با journald مدیریت می‌شود، در یک فرمت باینری پیچیده نگهداری می‌شوند و باید با journalctl کوئری شوند. این موضوع باعث می‌شود که لاگ های جورنال به صورت بالقوه تخریب پذیر باشند.

اندازه systemD آن را تبدیل به یک single point of failure کرده است، که این باعث شده گزارش‌های بیشماری از آسیب پذیری‌های آن ارائه شود، گرچه از نظر وسعت کوچک‌تر از خود کرنل لینوکس است.

این ها فقط چند مورد از انتقاداتی است که به این موجود بزرگ و تازه، وارد شده است. منتقدانش می‌گویند درست است که باید به دنبال جایگزینی مناسب برای init بود، اما این جایگزین هر چه باشد systemD نیست. systemD هیچ کاری هم که نکرده باشد باعث گروه‌بندی و طیف‌بندی جامعه لینوکس و یونیکس شده است. جامعه‌ای که به صورت یکپارچه هرروز برای رشد، توسعه و کارآمدی این سیستم عامل دوست داشتنی فعالیت می‌کردند. ولی حالا باید مقداری از انرژی خود را صرف متقاعد کردن دیگری برای استفاده یا کنارگذاشتن systemD کنند.

اما چیزی که راجع به بحث درباره systemD عجیب است، اینست که با وجود انتقادهای زیاد، به طور گسترده پذیرفته شده است. گنوم از ورژن 3.8 به بعد برای اجرا به systemD احتیاج دارد. فدورا15، (جامعه کاربری Red Hat) نخستین توزیع مادری بود که شروع به استفاده از آن به صورت پیشفرض کرد. از آن زمان به بعد دبیان، اوپن سوزه و اوبونتو همگی آن را پذیرفتند.

علت استفاده از systemD

چهار دلیل زیر را برای استفاده از systemD به جای init می توان ذکر کرد:

1- Parallel Startup of Service at boot time

اجرای موازی سرویس‌ها در زمان بوت سیستم که این منجر به کاهش زمان بوت می‌شود.

2- On-demand activation of demands

بر اساس نیاز و درخواست‌ها سرویس را اجرا می‌کند.

3- Support for System State Snapshots

کمک به پروسه بکاپ گیری سیستم.

4- Dependency-based service Control Logic

وابستگی‌های اجزای یک سرویس را برای اجرا کاملا چک می‌کند.

بر خلاف توزیع‌هایی که هم اکنون از init استفاده می‌کنند مانند نسخه های 5 و 6 توزیع‌های RHEL و CentOS ، در توزیع‌هایی که از systemD استفاده می‌کنند مانند فدورا نسخه های 15 به بعد، دیگر نمی‌توان از دستور های service یا chkconfig استفاده نمود بلکه تنها یک دستور برای کنترل سرویس ها و البته کنترل فرایند systemd وجود دارد و آن دستور **systemctl** می‌باشد که واقعا تحول و سادگی بسیاری را برای کنترل سرویس‌ها فراهم کرده است. در توزیع هایی مانند فدورا 20 که از فرایند systemD استفاده می‌شود وظایف دو دستور فوق بر عهده دستور **systemctl** می‌باشد.

در init از دستورات service برای کنترل روی سرویس‌ها مانند ریست کردن و یا عملیات دیگر، و از دستور chkconfig برای مدیریت سرویس‌ها و اینکه مشخص کنیم چه سرویسی در چه سطح اجرایی در زمان بوت سیستم اجرا گردد، استفاده می‌شود. هر تغییری که برای اجرای سرویس‌ها توسط دستور service اعمال می‌شد موقتی بود و باید توسط دستور chkconfig وضعیت آنها را به دائمی تغییر می‌دادیم.

با ارئه systemD در مفهوم سطوح اجرایی هم تغییراتی بوجود آمد. در توزیع هایی که از init استفاده می‌کنند سطوح اجرایی مطابق با آنچه است که در بخش init گفته شد و نوع سطح اجرایی سیستم های مبتنی بر init با مقدار initdefault که در فایل /etc/inittab آمده مشخص می‌گردد، اما در توزیع هایی که از systemD استفاده می‌کنند تعیین و نوع سطح اجرایی پیشفرض تغییر کرده و با مفهومی به نام Target جایگزین شده که ادامه به آن خواهیم پرداخت.

Unit ها در systemD

systemD برای کنترل و مدیریت سیستم از یکسری subsystem هایی به نام unit استفاده می‌کند. تعداد این یونیت‌ها 12 عدد است و داخل هر یونیتی می‌تواند صدها unit از همان جنس قرار گرفته باشد. لیست این یونیتها در زیر آمده است:

List Units SystemD

1. **.snapshot** ➔ System Saved Status
2. **.scope** ➔ External Process that Not Started by systemd
3. **.slice** ➔ Management Unit of Process
4. **.patch** ➔ File or Directory Units
5. **.mount** ➔ File Systems Mount Points Units
6. **.automount** ➔ File Systems Auto Mounts Units
7. **.service** ➔ System Service Units
8. **.target** ➔ Group of Units

9. **.device** ➔ Kernel Device Names
10. **.timer** ➔ Systemd Timer
11. **.swap** ➔ Swap File Unit
12. **.socket** ➔ IPC Socket Unit

در ادامه هر کدام از این یونیتها به طور مختصر توضیح داده می‌شود:

.snapshot ➔ System Saved Status

این unit ها وضعیت بخشهای دیگر systemD را درون خود نگهداری می کند که به نوعی مانند بکاپ است.

.scope ➔ External Process that Not Started by systemD

پراسسی که توسط systemD اجرا نشده باشد در این یونیت قرار می گیرد.

.slice ➔ Management Unit of Process

پروسس‌هایی که توسط systemD اجرا شوند در این unit قرار می گیرند. پروسس ها با .slice مدیریت می‌شوند.

.patch ➔ File or Directory Units

این unit مسیر فایل یا دایرکتوری را در سیستم مشخص می کند. از این یونیت برای معرفی فایل و پوشه استفاده می‌شود.

.mount ➔ File Systems Mount Points Units

این یونیت برای کار و اشاره به mountهای فایل سیستم استفاده می‌شود.

.automount ➔ File Systems Auto Mounts Units

این یونیت برای شناختن دیوایس‌هایی به کار می‌رود که قابلیت auto mount دارند مثل حافظه‌های جانبی.

.service ➔ System Service Units

تمامی سرویس‌های سیستم از جنس این یونیت هستند. این یونیت اشاره به سرویس‌های سیستم دارد.

.target ➔ Group of Units

سطوح اجرایی سیستم در این یونیت تعریف می‌شوند. تشخیص runlevel های سیستم مهم است و محل این تشخیص هم بستگی به System Manager و Service Manager سیستم دارد.

.device ➔ Kernel Device Names

هر دیوایسی که در سیستم اضافه شود کار شناسایی آن توسط این یونیت انجام می‌شود. تمامی دیوایس‌های شناسایی شده در سیستم در این یونیت قرار می‌گیرند.

.timer ➔ Systemd Timer

هر چیزی که در سیستم قابلیت زمان بندی داشته باشد در این یونیت قرار می‌گیرد.

.swap ➔ Swap File Unit

مفاهیم swap کلا در این یونیت قرار می‌گیرد.

.socket ➔ IPC Socket Unit

این یونیت برای مدیریت سوکت‌های ارتباطی شبکه‌ای سیستم استفاده می‌شود.

با مجموعه آپشن‌های دستور systemctl می‌توان بر روی Unit ها کنترل بیشتری داشت، Unit هایی مانند: سرویس‌ها، سوکت‌ها، دیوایس‌ها و یا نقاط اتصال. برای فهرست کردن تمامی Unitها می‌توانید از دستور زیر استفاده کنید.

#systemctl list-units

خروجی دستور بالا لیستی از تمام Unit های سیستم را نشان می‌دهد، اما اگر بخواهیم Unit خاصی را مشاهده کنیم از دستور زیر بهره می‌بریم:

#systemctl list-unit-files --type=service or #systemctl list-unit-files --type=target

البته اگر دستور systemctl را بدون هیچ گونه آپشنی به کار ببریم لیست تمام یونیت‌های systemD را نشان می‌دهد:

#systemctl

بهتر است از این دستور ، بدون آپشن استفاده نکنید چون لیست بسیار طولانی به ما نشان می دهد که جستجو در آن وقت گیر و مشکل است. بهتر است از سوئیچ list-units در ادامه دستور استفاده شود. با این کار یونیت هایی که یکبار از انها استفاده شده در لیست خروجی نمایش داده می شود:

#systemctl list-units

می توانیم با فیلتر گذاری خروجی این دستور را اختصاصی‌تر کنیم. مثلا با دستور زیر کاری می‌کنیم که فقط یونیتهایی نشان داده شوند که وضعیت آنها inactive است. یعنی نوع فیلتر گذاری ما بر اساس وضعیت است. یا می‌توانیم فیلتر را بر اساس نوع یونیت مشخص کنیم که برای این دو حالت در ادامه مثال هایی آمده است:

#systemctl list-units --all --state=inactive
#systemctl list-units -type=service
or
#systemctl list-units -type=service --state=inactive

اگر بخواهیم تمامی یونیتهایی که اسکریپت دارند را مشاهده کنیم با دستور زیر می‌توانیم این کار را انجام دهیم:

#systemctl list-unit-files

و اگر بخواهیم بفهمیم یک unit برای اجرا چه وابستگی‌هایی دارد از این دستور استفاده می کنیم. اگر در آخر این خط از سوئیچ R- استفاده شود در خروجی نشان میدهد این unit خودش وابستگی چه unit های دیگری است:

#systemctl list-dependencies xxx.service
#systemctl list-dependencies xxx.service -R

مسیرهای مهم در systemD

درون هر unit هزاران پارامتر از جنس همان unit می‌تواند قرار بگیرد که به آنها subunit گفته می‌شود. هر کدام از این subunit ها فایل اسکریپتی دارند، این فایل‌ها در یکی از سه مسیر زیر قرار دارند که در ادامه توضیح داده می‌شود. systemD از روی این اسکریپتهاست که می‌داند از کجا باید یک subunit ها را کنترل کند. اگر در دستورات systemD نام یونیت را کاملا معرفی نکنیم سیستم فرض را بر این می گذارد که در یونیت service مشغول به کار هستیم.
فایل‌های مربوط به یونیتها را در مسیرهای زیر می‌توان پیدا کرد:

/usr/lib/systemd/system/
/run/systemd/system/
/etc/systemd/system/

در ادامه هر یک از این مسیرها را بررسی خواهیم کرد:

مسیر اول : /usr/lib/systemd/system/

اولین جائیکه که درون آن اسکریپت‌های زیادی برای subunit های **.service** وجود دارد این مسیر است. این اسکریپتها با نصب سرویس‌ها در این مسیر اضافه شده‌اند. هر پکیجی که با rpm نصب می‌شود برای اینکه خود را به عنوان یک subunit به سیستم اضافه کند باید در این مسیر یک اسکریپت بسازد.

مسیر دوم : /run/systemd/system/

دومین مسیر، اسکریپت‌هایی از جنس **.scop** در آن قرار دارد. Subunit ها موقع اجرا به عنوان سرویس‌های زیر مجموعه systemD ، باعث ایجاد شدن یکسری subunit های دیگر مثل Process Subunit ها می‌شوند. subunit هایی که موقع runtime کرنل اجرا می‌شوند در این مسیر قرار می گیرند.

به این نکته مهم توجه کنید که فایل های کنترلی subunit ایجاد شده در موقع runtime به فایل‌های کنترلی نصب شده در زمان نصب اولیه ارجحیت اجرایی دارند. در بحث اجرا ، مسیر **/run/sydtemd/system/** به مسیر **/usr/lib/systemd/system/** ارجحیت اجرایی دارد.

به عنوان مثال systemD برای شناختن سرویس FirewallD نیاز به یک اسکریپت دارد، این اسکریپت باید در مسیر **/usr/lib/systemd/system/** قرار داشته باشد، اما برای اجرای FirewallD در قالب یک Process به یک اسکریپت نیاز دارد که این فایل باید در مسیر **/run/sydtemd/system/** وجود داشته باشد.

با reset کردن سیستم ، اسکریپت‌هایی که در این مسیر قرار دارد از بین می‌رود. اسکریپت‌های موجود در این مسیر تا زمانی اجرا می‌شوند که سیستم UP باشد.

مسیر سوم : /etc/systemd/system/

سومین محلی که می توان فایل های کنترلی subunit ها را درون آن پیدا کرد این مسیر است که مهمترین آنها **.target** ها می‌باشند. اسکریپتهای موجود در این مسیر نسبت به مسیرهای گفته شده ارجحیت اجرایی بالاتری دارند به همین دلیل وقتی برای یک subunit فایل کنترلی می‌نویسیم آن را باید در این مسیر قرار دهیم.

شیوه نوشتن اسکریپت در systemD

همانطور که گفته شد فایل‌های کنترلی subunit ها نوعی از اسکریپت هستند که فورمت نوشتاری خاص خود را دارند. در این بخش نمونه‌ای از محتویات یک فایل کنترلی ساده را مورد بررسی قرار می‌دهیم. در زیر نمونه ای از یک اسکریپت آمده است.

```
[Unit]
Description=Daemon to detect crashing apps
After=syslog.target

[Service]
ExecStart=/usr/sbin/abrtd
Type=forking
```

[Install]
WantedBy=multi-user.target

[Unit]: در این قسمت یکسری از اطلاعات در مورد subunit مورد نظر داده می‌شود. که مهترین قسمت آن بخش After می‌باشد. After پیشنهادی است که نویسنده unit می‌دهد و مشخص می‌کند که ترجیحا این subunit بعد از استارت یک subunit دیگر، اجرا شود.

[Service]: این قسمت اطلاعات سرویس را مشخص می‌کند. آدرس مقابل ExecStart مشخص کننده این است که این اسکریپت از کجا باید اجرا شود. در اصل محل فایل اجرایی آن را نشان می‌دهد.
حتی روش اتمام کار این subunit را می‌توان با گزینه مقابل Type مشخص کرد که 99 درصد مواقع fork می‌باشد. کلا در جلوی گزینه Type یکی از دو گزینه worker ویا forking می‌تواند قرار بگیرد. forking یعنی زمانی که process اصلی این subunit از بین می‌رود و systemD آن را به عنوان پروسه اتمام شده قبول می‌کند.

[Install]: این قسمت مشخص کننده این است که سرویس مورد نظر در چه Target ای کار می‌کند. این بخش در زمان اجرای دستور **systemctl enable SERVICENAME** خوانده می‌شود. گزینه enable باعث می‌گردد سرویس مورد نظر در هنگام بوت سیستم فعال باشد.
وقتی برای یک سرویس، اسکریپتی نوشته می‌شود systemD از وجود آن مطلع نیست لذا باید کاری کنیم علاوه بر مطلع شدن از اسکریپت جدید آن را در هنگام بوت فعال کند. برای این کار، ابتدا باید فایل ایجاد شده را در مسیر **/etc/systemd/system/** کپی کنیم و سپس با دستور زیر این تغییرات را به اطلاع systemD بی‌رسانیم:

#systemctl daemon--reload

برای دیدن ویژگی های بیشتر یک unit از این دستور استفاده می‌کنیم. این ویژگی‌ها در داخل فایل اسکریپت دیده می‌شود:

#systemctl show UNITNAME

دستور زیر هم فقط بخش After یک اسکریپت unit را نشان می‌دهد. این دستور معادل grep کردن یک بخش از فایل است:

#systemctl show xxx.service -p After

و اگر بخواهیم بفهمیم یک unit برای اجرا چه وابستگی‌هایی دارد از این دستور استفاده می‌کنیم. اگر در آخر این خط از سوئیچ R- استفاده کنیم در خروجی نشان میدهد این unit خودش وابستگی چه unit های دیگری است.

#systemctl list-dependencies xxx.service
#systemctl list-dependencies xxx.service -R

این آپشن برای مواقعی خوب است که می‌خواهیم یک سرویس را پاک کنیم، لذا قبل از پاک کردن آن چک می‌کنیم سرویس مورد نظر وابستگی سرویس دیگری نباشد.

Target جانشین Run Level

یکی دیگر از کارهای systemD مدیریت runlevel های سیستم در قالب Target unite ها می‌باشد .در توزیع‌هایی مانند RHLE/CentOS 5,6 که از init استفاده می‌کنند مفهوم سطح اجرایی با توزیع‌هایی مانند فدورا 20 و یا RHEL7 که از systemD استفاده می‌کنند، متفاوت است.

systemD از مفهوم target به جای Run level استفاده می‌کند که شبیه به سطوح اجرایی init است اما تفاوتهایی با هم دارند. هر target با یک نام مشخص می‌شود در صورتی که در init هر سطح اجرایی با یک عدد مشخص می‌شد و برای هدفی خاص در نظر گرفته شده است. همانند سیستم init ، در systemD نیز شروع به کار هر target با سرویس‌هایی همراه است که می‌توانند سرویس‌هایی را از target دیگری به ارث ببرد.

بطور مثال یک target به نام **graphical.target** باعث start شدن X11 و ایجاد یک محیط گرافیکی می‌شود، و سرویسی به نام gdm.service را اجرا می‌کند و یا target ای به نام **multi-user.target** سرویس‌های دیگری مانند سرویس مدیریت شبکه یا همان NetworkManager.service را اجرا می‌کند. پس همانطور که ملاحضه می‌کنید بر خلاف Run level های init ، در systemD چندین target می‌تواند همزمان اجرا شوند و هر کدام سرویس‌هایی را اجرا کنند، برای نشان دادن target پیشفرض یا default target از دستور زیر استفاده کنید.

#systemd get-default
OUTPUT: runlevel5.target

runlevel5.target یک لینک به target ای به نام **graphical.target** است که معادل سطح اجرایی 5 در سیستم init است، یا target ای به نام **multi-user.target** معادل سطح اجرایی 3 در سیستم init است که در systemD معادل **runlevel3.target** و به **multi-user.target** لینک می شود.

#ll /etc/systemd/system/default.target
#ll /usr/lib/systemd/runlevel5.target

همانطور که در خروجی دستورات بالا می‌بینید فایل **default.target** یک لینک به **runlevel5.target** و خود این فایل یک لینک به فایل **graphical.target** می‌باشد. برای سوئیچ از **graphical.target** به **multi-user.target** از دستور زیر استفاده کنید:

#systemctl isolate multi-user.target

و برای سوئیچ از **multi-user.target** به **graphical.target** دستور زیر را به کار ببرید که البته اعمال این دستورات کمی کند و با تاخیر انجام می شود:

#systemctl isolate graphical.target

از فرمت کلی زیر برای تغییر target پیشفرض استفاده کنید. توجه شود که تمامی target ها به پسوند target. ختم می‌شوند. با دستور زیر متوجه می شویم سیستم با چه target پیش فرضی بوت می‌شود:

#systemctl set-default

برای تغییر پیش فرض سیستم به جای کلمه get از کلمه set استفاده می کنیم:

#systemctl get-default graphical.target

لیست تمامی یونیت‌های که target. دارند توسط دستور زیر نشان داده می‌شود:

#systemctl list-units --type=target

و برای دیدن Target پیش فرض سیستم هم می‌توانید فایل زیر را ببینید:

#cat /etc/systemD/system/default.target

اگر بخواهیم بدانیم چه تارگت‌های در سیستم وجود دارند و حداقل یکبار استفاده شده‌اند از دستور زیر استفاده می کنیم .

#systemctl list-unit-files --type=target

or

#systemctl list-units --type=target

در systemD مد rescue معادل همان runlevel 1 در init می‌باشد. برای رفتن به این مد، هم می‌توان در زمان بوت سیستم اقدام کرد و هم می‌توان در هنگام کار سیستم توسط دستور زیر به این مد رفت:

#systemctl rescue

نکته: با استفاده از دستور زیر مشاهده کنید که فرایند systemD به عنوان والد تمامی فرایندهای موجود در سیستم و با شناسه 1 در صدر آنها قرار می گیرد.

#pstree -Ap

- جدول زیر Run Levelهای معادل init در systemD را نشان می دهد.

SysV Runlevel	systemd Target	Notes
0	runlevel0.target, poweroff.target	Halt the system.
1, s, single	runlevel1.target, rescue.target	Single user mode.
2, 4	runlevel2.target, runlevel4.target, multi-user.target	User-defined/Site-specific runlevels. By default, identical to 3.
3	runlevel3.target, multi-user.target	Multi-user, non-graphical. Users can usually login via multiple consoles or via the network.
5	runlevel5.target, graphical.target	Multi-user, graphical. Usually has all the services of runlevel 3 plus a graphical login.
6	runlevel6.target, reboot.target	Reboot
emergency	emergency.target	Emergency shell

مدیریت لینوکس با systemD

در این قسمت به معرفی دستور systemctl و بعضی از دستورات systemD و همچنین به چگونگی کنترل سرویسها که پیش از این توسط دستور service انجام می‌شد نیز خواهیم پرداخت.

همانطور که گفته شد systemD بر خلاف init وظایف زیادی را به عهده گرفته است. یکی از وظایفی که systemD انجام می‌دهد چک و handle کردن وقایع رخ داده شده بر روی سیستم است. برای query گرفتن از سیستم می‌توان از دستور journalctl و سوئیچ‌های آن استفاده کرد. دستور journalctl تمام اتفاقات روی سیستم، از زمانی که سیستم عامل نصب شده است را نشان می‌دهد و به ما کمک می‌کند log های سیستم بر اساس تاریخ و یا فیلتر خاصی مشاهده کنیم. ابزار journalctl جایگزین ابزار demsg شده که مشابهت‌هایی با هم دارند. اگر می‌خواهیم فقط موارد مربوط به یک سرویس خاص را ببینیم بهتر است مسیر اجرایی برنامه مورد نظر را به آن معرفی کنیم :

#journalctl /sbin/crond

با سوئیچ b– هم مشخص می‌کنیم اتفاقات آخرین بوت سیستم را به ما نشان دهد:

#journalctl -b

و یا با آپشن **--since=today** می‌توان زمان رخ دادن اتفاقات را هم مشخص کرد. کلمه today قابل تغییر است و یا حتی می‌شود به جای today تاریخ یک روز را نوشت:

#journalctl -b --since=today

در لینوکس برای اشاره به بعضی از کلمات یکسری حروف اختصار داریم مثل:

err = error
info = information
warn = warning
emer = emergency

از این موارد خلاصه شده می‌توان در دستورjournalctlاستفاده کرد. با سوئیچ p– می‌توان مشخص کرد جنس وقایعی که مایلیم نشان داده شود چه چیزی باشد. سوئیچ p– به نوعی مانند فیلتر برای ما عمل می‌کند. اگر از سوئیچ pb– استفاده کنیم خروجی که به ما نشان داده می‌شود اتفاقاتی می باشد که از زمان آخرین بوت سیستم رخ داده است:

#journalctl -p err
#journalctl -pb err

ما قبلا برای دیدن لاگهای سیستم به صورت live از ترفند دستور tail استفاده می‌کردیم. در حال حاضر هم این دستور کارکرد دارد اما journalctl راه کار راحت‌تری به ما نشان می‌دهد. برای این کار از سوئیچ f– آن استفاده می‌کنیم:

#tail -f /var/log/messages
#journalctl -f

با سوئیچ K– هم می‌توان از وقایع کرنل مطلع شد:

#journalctl -K

یکی دیگر از کارهای اصلی systemD مدیریت پروسه بوت می‌باشد بنابراین براحتی می‌توان از تمام وقایعی که در هنگام پروسه بوت اتفاق افتاده query گرفت. مثلا اگر بخواهیم بدانیم سیستم در چند ثانیه بوت شده از این دستور:

#systemd-analyze

و اگر بخواهیم بدانیم در این چند ثانیه بوت دقیقا چه اتفاقی افتاده است دستور بالا را به همراه کلمه blame می‌آوریم. اگر سیستم کند بالا بیاید یعنی یک سرویس، تایم زیادی از سیستم برای بارگذاری گرفته است. با دستور زیر می‌توان تایم سرویس‌هایی که در زمان بوت سیستم load شده‌اند را مشاهده کرد:

#systemd-analyze blame

systemD پروسس ها را در قالب Control Groups دسته بندی و کنترل می‌کند. مثلا تمامی پروسس‌هایی که تحت سرویس sshd استارت شده‌اند در یک CG قرار می‌گیرند. برای دیدن Control Group های سیستم از این دستور بهره می‌بریم:

#systemd-cgls

خروجی این دستور تمامی دسته بندی‌های یک سرویس را نشان می‌دهد، این دستور معادل pstree می‌باشد.
ممکن است در سرورهای عملیاتی پروسس یا گروهی از آنها منابع زیادی از سیستم را اشغال کرده و در عملکرد آن اختلال بوجود بیاورند، در چنین شرایطی باید در محیط خشک cli بتوانیم این پروسس‌ها را پیدا و مهار کنیم. دستور زیر گروه سرویس‌ها را بر اساس میزان استفاده از cpu و ram و ورودی/خروجی هارد دیسک به صورت sort شده نشان می‌دهد. از این دستور برای گزارش گیری از سیستم هم استفاده می‌شود.

#systemd-cgtop

برای تغییر hostname سیستم از سه طریق می‌توان این کار را انجام داد. یا باید از طریق ویرایش فایل hostnamt در زیر دایرکتوری etc/ نام سیستم را تغییر داد و یا اینکه با دستور hostname این کا را انجام دهیم که تغییر حاصل شده دائمی نیست و با reset سیستم از بین می‌رود. سیاست systemD این است که شما کمترین میزان مراجعه به فایل ها را داشته باشید، لذا امکاناتی را در اختیار کاربر می گذارد که مستقیم و بدون مراجعه به فایل بتواند تغییرات دائمی را در سیستم انجام دهد. یکی از این دستورات hostnamectl است. دستور زیر علاوه بر نام سیستم شما اطلاعات تکمیلی دیگری هم ارائه می‌کند:

#hostnamectl

اما اگر بخواهیم نام سیستم را تغییر دهیم این دستور را وارد می‌کنیم. این نام به صورت دائمی تنظیم می‌شود:

#hostnamectl set-hostname newName

با systemD می‌توانیم به راحتی زبان‌های سیستم را مدیریت کنیم. اگر بخواهیم بدانیم زبان local ماشین، زبان کیبورد و زبان نمایش محیط گرافیکی ما بر روی چه چیزی تنظیم شده است از دستور زیر استفاده می‌کنیم:

#localectl

با دستورات زیر به ترتیب زبان سیستم، زبان کیبورد و زبان قابل نمایش در دسکتاپ را تنظیم می‌کنیم:

#localectl set-locale LANG=en_us
#localectl set-keymap en_us
#localectl set-x11-keymap en_us

systemd توانایی نشان دادن تنظیم زمان‌های سیستم را دارد. با دستورات زیر می‌توان تنظیمات مورد نظر را اعمال کرد. این دستور تاریخ جاری سیستم نشان داده می‌شود:

```
#timedatectl
```

و دستورات زیر به ترتیب تاریخ سیستم، زمان سیستم و تایم zone را تنظیم می‌کند:

```
#timedatectl  set-time  YYY-MM-DD
#timedatectl  set-time   HH:MM:SS
#timedatectl  set-time   America/new_York
```

این دستور هم لیست timezone های سیستم را نشان می‌دهد:

```
#timedatectl  list-timezones
```

یکی از امکانات جالب systemD ، مدیریت یوزرها می‌باشد. دستور زیر لیست یوزرهای سیستم را نشان می‌دهد:

```
#loginctl  list-users
```

و این دستور نشان می دهد چه یوزرهایی به سیستم لاگین کرده‌اند. این دستور معادل who می‌باشد اما شیوه نشان دادن آنها با هم فرق دارد:

```
#loginctl  list-sessions
```

این دستور هم مشخصات کامل یک یوزر را نشان می‌دهد:

```
#loginctl  show-user  root
```

کار با دستور systemctl

در init به ازای هر کار، یک دستوری داشتیم اما در systemD تمام کارهای init در مجموعه جدیدی از دستورات به نام **systemctl** گردآوری شده که بسیاری از نیازهای یک admin را پوشش می‌دهد، بهترین دستوری که برای ارتباط با این system manager و کنترل یونیتها می‌توان استفاده کرد systemctl است. برای شروع کار، به جهت نمایش متغیر های محلی $PATH **و**$LANG از دو دستور زیر استفاده می‌کنیم:

#systemctl show-environment

و یا برای تغییر متغیر محیطی $PATH و افزودن یک مسیر جدید از این دستور هم می‌توان استفاده کرد:

#systemctl set-environment PATH=$PATH:/tohid

یکی از موارد مدیریت سیستم، kill کردن یک پروسه است. با systemctl به راحتی می‌توان پروسسی را kill کرد:

#systemctl kill sshd

و یا به پروسه‌ای سیگنال خاصی فرستاد :

#systemctl kill -s SIGKILL sshd.service

و اگر هم بخواهیم یک سرویسی را از ریشه، با تمام وابستگی های آن kill کنیم از دستور زیر بهره می‌بریم:

#systemctl kill -s HUP --kill-who=main sshd.service

در سیستم عامل، cpu سیستم با عدد مشترک 1024 در اختیار پروسس‌ها گذاشته می‌شود اگر این عدد برای یک سرویس به 2048 تغییر کند یعنی دو برابر بقیه پروسس‌ها از cpu استفاده می‌کند و اگر این عدد 512 تنظیم شود نصف بقیه پروسس‌ها (سرویس ها) می تواند از cpu را در اختیار داشته باشد. این عدد یک معیار بوده و به صورت اعداد زوج اختصاص داده می‌شود. به این معیار cpu share گفته می‌شود. در systemD این کار توسط دستور زیر قابل انجام است:

#systemctl set-property sshd.service CPUShares=2048

این دستور به صورت پیش فرض تغییرات را به صورت دائمی اعمال می‌کند و حتی با ریبوت سیستم از بین نمی‌رود مگر اینکه از سوئیچ **runtime--** استفاده کنیم. این سوئیچ باعث می‌شود دستور وارد شده در زمان جاری سیستم اجرایی باشد و به محض ریبوت سیستم از بین می‌رود:

#systemctl set-property sshd.service CPUShares=2048 --runtime

بعد از تغییر cpu share یک سرویس باید مطمئن شویم تنظیمات وارد شده به صورت صحیح اعمال شده باشد. دستور زیر این کار را برای ما انجام می‌دهد ، علاوه بر این، با دستور زیر می‌توان فهمید به بقیه سرویس‌ها چه مقدار Cpu share اختصاص داده شده است:

#systemctl show -p CPUSares sshd.service
#systemctl show CPUShares

یکی دیگر از کارهای اصلی systemD مدیریت سرویس‌ها می باشد. این اعمال مدیریت بر سرویس‌ها توسط دستور systemctl انجام می‌شود. یک Service Unit مشخص کننده یک سرویس است که به صورت یک فرایند با یک شناسه

در سیستم وجود دارد مانند سرویس sshd. با یکسری از سوئیچ های systemctl می‌توانیم از اعمال تغییرات بر روی سرویس اطمینان حاصل کرد و یا در نحوه اجرای آن سرویس تغییراتی بوجود آورد. به طور عمومی بر روی هر سرویس شش عمل enable, disable, reload, restart, stop, start انجام می گیرد. این اعمال در زیر توضیح داده شده است. شیوه نوشتاری این دستور برعکس دستورات init می باشد.

#systemctl COMMAND SERVICE_NAME.service

#systemctl start|stop|restart|reload|status|is-active|enable|disable **servicename.servic**

دقت کنید اگر در این اینجا نام یونیت را کامل ننویسیم به طور پیش فرض آن را در یونیت **.service** قرار می‌دهد. در ادامه بعضی از این سوئیچ ها توضیح داده می‌شود.

- **enable:** این کلمه سرویس را در session جاری استارت نمی کند بلکه باعث می‌شود در بوت بعدی سیستم سرویس مورد نظر به صورت خودکار فعال شده و شروع به کار کند. این دستور همان کاری را انجام می‌دهد که دستور chkconfig در init انجام می‌دهد.

#systemctl enable sshd.service

- **disable:** این کلمه باعث غیرفعال شدن سرویس در session جاری نمی‌شود بلکه در بوت بعدی سیستم از فعال شدن آن جلوگیری می کند.
- **start, stop:** این دو کلمه باعث می‌شوند یک سرویس در session جاری شروع به کار کند یا جلوی کار آن گرفته شود.
- **reload, restart:** اگر بخواهیم تغییرات به یک سرویس اعمال شود آن را reset نمی کنیم بلکه آن را reload می کنیم، چون با restart کردن یک سرویس کانکشن کسانی که به سرویس مورد نظر متصل هستند قطع شده و ممکن است باعث از دست رفتن دیتا و یا ضرر و زیان جبران ناپذیری بشود اما اگر سرویس را reload کنیم دو حالت پیش می آید. در حالت اول کسانی که به آن سرویس متصل هستند با همان تنظیمات قبلی به کار خودشان ادامه می دهند، اما در حالت دوم کسانی که بعد از reload به سیستم وصل می شوند با تنظیمات جدید با آن کار می کنند. اما restart یک سرویس باعث می شود تمام کانکشن ها به آن قطع شده و اتصال های جدید با تنظیمات جدید انجام گیرد. دستور زیر ابتدا چک می کند که آیا می شود سرویس را reload کرد یا خیر. اگر توانست عمل reload را انجام دهد که مشکلی نیست ولی اگر نتوانست آن را reset می کند.

#systemctl reload-or-restart sshd.service

- **is-enabled:** از سوئیچ is-enabled برای این استفاده می‌کنیم که بفهمیم آیا سرویسی در زمان بوت شدن به طور خودکار فعال می‌شود یا خیر. اگر فعال باشد عبارت enabled و اگر غیر فعال باشد عبارت disabled در خروجی نشان داده می‌شود.
- **is-active:** برای مشخص کردن این است که آیا سرویس مورد نظر فعال است یا خیر. در صورتی که سرویسی فعال و در حال اجرا باشد در خروجی عبارت active و اگر غیر فعال باشد عبارت inactive را نشان می‌دهد.

#systemctl is-active httpd.service
active
#systemctl is-active named.service
inactive

- **is-failed:** اگر هم سرویس فعال نشد با این دستور می توانیم علت آن را پیدا کنیم.

از میان دستورهای init، تنها دستور chkconfig –list در توزیع‌هایی که از سیستم systemd استفاده می‌کنند قابل استفاده نمی‌باشد و برای مابقی دستورها می‌توان از معادل init آن استفاده کنید که در این صورت دستور service **و یا** chkconfig به معادل خودش در systemd ترجمه یا Redirect شده و خروجی مشابه زیر نشان داده می شود.

#service sshdd start
Redirecting to /bin/systemctl start sshd.service

و مجموعه دستور chkconfig جای خود را به دستورات زیر داده است:

- #chkconfig nameservice on ===➔ systemctl enable nameservice
- #chkconfig nameservice off ===➔ systemctl disable nameservice
- #chkconfig nameservice ===➔ systemctl is-enabled nameservice
- #chkconfig --list ===➔ systemctl list-unit-file --type=service or target

تعیین وضعیت یک سرویس

در لینوکس فایلی وجود دارد به نام /etc/rc.local که اگر بخواهیم یک پروسه‌ای بعد از بوت سیستم اجرا شود اسکریپت آن را درون این فایل قرار می‌دهیم. اما برای دیدن وضعیت سرویس‌هایی که اجرا شده و در حال کار هستند می‌توانیم از دستور **systemctl status** استفاده کنیم.

#systemctl status sshd.service

در خروجی سوئیچ **status** پارامترهای زیادی می‌تواند نشان داده شود که هر کدام از آنها معنی خاصی دارد. این پارامترها عبارتند از:

- **Loaded:** یعنی سرویس بارگزاری و اجرا شده است.
- **Active (running):** نشان می‌دهد سرویس فعال و در حال اجرا می‌باشد.
- **Active (exited):** این معنی را می‌دهد که سرویس یک بار اجرا و خارج می‌شود.
- **Active (waiting):** یعنی سرویس کار نمی‌کند و منتظر وقوع یک اتفاقی برای اجرا است.
- **Inactive:** یعنی سرویس غیر فعال است.
- **Enabled:** نشان دهنده این است که وقتی سیستم بوت می‌شود این سرویس در کنار بقیه سرویس‌های Target جاری فعال باشد.
- **Disabled:** این گزینه یعنی سرویس مورد نظر بعد از بوت سیستم فعال نشود.

نکته مهم: در خروجی دستور status با عبارتی برخورد می‌کنیم به نام CGroup که مخفف Control Group است. CG ویژگی از کرنل لینوکس می‌باشد که محدودیت‌هایی را بر روی منابعی سیستمی مانند پردازنده، حافظه اصلی و I/O به ازای گروهی از فرایندها اعمال می‌کند. systemD پروسس ها را در قالب Control Groups دسته بندی و کنترل می‌کند. مثلا تمامی پروسس‌هایی که تحت یک سرویس استارت شده‌اند در یک CG قرار می‌گیرند.

Mask کردن یک سرویس

با systemD می‌توان کاری کرد اجرای یک سرویس در انحصار یک یوزر خاص مثل root باشد بطوری که هیچ یوزر و سرویسی به غیر از یوزر root نتواند آن سرویس را فعال کند، به این کار mask کردن سرویس گفته می‌شود. برای این کار باید سرویس مربوطه را ابتدا Stop و Disable کرده سپس با دستور زیر آن را mask کنیم:

#systemctl masq NAMESERVICE
#systemctl mask sshd.service

اگر سرویس mask شده باشد و بخواهیم در وضعیت آن تغییری ایجاد کنیم با پیغام زیر مواجه می‌شویم :

Failed to start sshd.service: unit sshd.service is masked

برای اینکه سرویس را از حالت mask خارج کنیم دستور زیر را وارد می کنیم:

#systenctl unmask sshd.service

مطمئنا در یک مقاله نمی توان تمام موضوعات مربوط به systemd را پوشش داد. در ویرایش های بعدی، دستورات و امکانات بیشتری از این system manager بیان خواهد شد.

- جدول زیر معادل دستورهای init در systemd را نشان می دهد.

Sysvinit Command	Systemd Command	Notes
service frobozz start	systemctl start frobozz.service	Used to start a service (not reboot persistent)
service frobozz stop	systemctl stop frobozz.service	Used to stop a service (not reboot persistent)
service frobozz restart	systemctl restart frobozz.service	Used to stop and then start a service
service frobozz reload	systemctl reload frobozz.service	When supported, reloads the config file without interrupting pending operations.
service frobozz condrestart	systemctl condrestart frobozz.service	Restarts if the service is already running.
service frobozz status	systemctl status frobozz.service	Tells whether a service is currently running.
ls /etc/rc.d/init.d/	systemctl list-unit-files --type=service (preferred) ls /lib/systemd/system/*.service /etc/systemd/system/*.service	Used to list the services that can be started or stopped Used to list all the services and other units
chkconfig frobozz on	systemctl enable frobozz.service	Turn the service on, for start at next boot, or other trigger.
chkconfig frobozz off	systemctl disable frobozz.service	Turn the service off for the next reboot, or any other trigger.
chkconfig frobozz	systemctl is-enabled frobozz.service	Used to check whether a service is configured to start or not in the current environment.
chkconfig --list	systemctl list-unit-files --type=service(preferred) ls /etc/systemd/system/*.wants/	Print a table of services that lists which runlevels each is configured on or off
chkconfig frobozz --list	ls /etc/systemd/system/*.wants/frobozz.service	Used to list what levels this service is configured on or off
chkconfig frobozz --add	systemctl daemon-reload	Used when you create a new service file or modify any configuration