



Systemadministration - systemd

Inhaltsverzeichnis

1	Systemadministration - systemd	5
1.1	systemd der System- und Dienste-Manager	5
1.2	Konzeption des systemd	6
1.2.1	Unit Typen	6
1.2.2	systemd im Dateisystem	7
1.2.3	Weitere Funktionen von systemd	8
1.2.4	Handhabung von Diensten	8
1.2.5	Quellen systemd	10
1.3	systemd unit-Datei	11
1.3.1	Ladepfad der Unit-Dateien	11
1.3.2	Aktivierung der Unit-Datei	12
1.3.3	Sektionen der Unit-Datei	13
1.3.3.1	Sektion [Unit]	13
1.3.3.2	Typ spezifische Sektion	16
1.3.3.3	Sektion [Install]	17
1.3.4	Beispiel cupsd	18
1.3.5	Werkzeuge	22
1.3.6	Quellen systemd-unit-Datei	25
1.4	systemd-service	27
1.4.1	service-Unit anlegen	27
1.4.2	Sektion [Service]	27
1.4.3	Quellen systemd-service	31
1.5	systemd.target - Ziel-Unit (Runlevel)	32
1.5.1	Besonderheiten	33
1.5.2	Quellen systemd-target	34
1.6	systemd-mount	35
1.6.1	Inhalt der mount-Unit	36
1.6.2	Inhalt der automount-Unit	37
1.6.3	Beispiele	37
1.6.4	Quellen sytemd-mount	41
1.7	systemd-timer	43
1.7.1	Benötigte Dateien	43
1.7.2	.timer-Unit anlegen	43
1.7.3	.service-Unit anlegen	44
1.7.4	.timer-Unit eingliedern	45

1.7.5	.timer-Unit manuell auslösen	45
1.7.6	.timer-Unit als cron Ersatz	45
1.7.7	Quellen systemd-timer	46
1.8	systemd-path	47
1.8.1	Benötigte Dateien	47
1.8.2	.path-Unit Optionen	47
1.8.3	.path-Unit anlegen	49
1.8.4	.service-Unit anlegen	50
1.8.4.1	Zusätzliche .service-Unit anlegen	51
1.8.5	.path-Unit eingliedern	52
1.8.6	service-Unit manuell ausführen	53
1.8.7	Quellen systemd-path	54
1.9	Systemjournal	55
1.9.1	journald	55
1.9.2	journald über das Netzwerk	56
1.9.3	journald.conf	56
1.9.4	Rangfolge	57
1.10	journalctl	58
1.10.1	Rechte	58
1.10.2	Ausgabe filtern	58
1.10.3	Ausgabe steuern	60
1.10.4	journalctl steuern	61
1.10.5	journalctl beherrschen	61
1.10.6	Quellen	66

ANFANG INFOBEREICH FÜR DIE AUTOREN

Dieser Bereich ist vor der Veröffentlichung zu entfernen !!!

Status: RC3

Änderungen 2021-02 bis 03:

- Entfernen des Kapitels "Installation und Einrichten von systemd"
- Umorganisation in "systemd-start" mit Link auf alle weiteren, im Handbuch aufgeführten, Unit-Typen.
- Korrektur und Aktualisierung aller Links
- Für die Verwendung mit pandoc optimiert.
- Review
- Link zu systemd.automount auf systemd.mount gesetzt, da beides in systemd.mount behandelt wird.
- Absätze "Unit-Dateien" und "Quellen" eingefügt.

Änderungen 2021-04

- Durch “systemd-unit-datei” erforderliche Anpassungen

ENDE INFOBEREICH FÜR DIE AUTOREN

1 Systemadministration - systemd

1.1 systemd der System- und Dienste-Manager

Anmerkung:

Die folgende, allgemeine Einführung zu systemd wurde überwiegend der ins [deutsche übersetzten Manpage](#) entnommen. Der Dank geht an Helge Kreutzmann.

systemd ist ein System- und Diensteverwalter, der beim Systemstart als erster Prozess (als PID 1) ausgeführt wird und somit als **Init-System** agiert, das System hochfährt und auf Anwendungsebene **Dienste verwaltet**.

Entwickelt wird es federführend von den Red Hat Entwicklern Lennart Poettering und Kay Sievers.

In Debian wurde die Einführung des systemd als Standard-Init-System lange, kontrovers und emotional diskutiert bis im Februar 2014 der Technische Ausschuss für systemd stimmte.

Seit der Veröffentlichung von 2013.2 "December" benutzt siduction bereits systemd als Standard-Init-System.

1.2 Konzeption des systemd

Systemd stellt ein Abhängigkeitssystem zwischen verschiedenen Einheiten namens „Units“ in 11 verschiedenen Typen (siehe unten) bereit. Units kapseln verschiedene Objekte, die für den Systemstart und -betrieb relevant sind.

Units können „aktiv“ oder „inaktiv“, sowie im Prozess der „Aktivierung“ oder „Deaktivierung“, d.h. zwischen den zwei erstgenannten Zuständen sein. Ein besonderer Zustand „*fehlgeschlagen*“ ist auch verfügbar, der sehr ähnlich zu „inaktiv“ ist. Falls dieser Zustand erreicht wird, wird die Ursache für spätere Einsichtnahme protokolliert. Siehe die Handbuchseite [Sytemd-Journal](#).

Mit systemd können viele Prozesse parallel gesteuert werden, da die Unit-Dateien mögliche Abhängigkeiten deklarieren und systemd erforderliche Abhängigkeiten automatisch hinzugefügt.

Die von systemd verwalteten Units werden mittels Unit-Dateien konfiguriert.

Die Unit-Dateien sind in verschiedene Sektionen unterteilte, reine Textdateien im INI-Format. Dadurch ist ihr Inhalt ohne Kenntnis einer Scriptsprache leicht verständlich und editierbar. Alle Unit-Dateien müssen eine Sektion entsprechend des Unit Typ, und können die generischen Sektionen „[Unit]“ und „[Install]“ enthalten. Die Handbuchseite [Systemd Unit-Datei](#) erläutert den grundlegenden Aufbau der Unit-Dateien, sowie viele Optionen der generischen Sektionen „[Unit]“ und „[Install]“.

1.2.1 Unit Typen

Bevor wir uns den Unit-Typen zuwenden, ist es ratsam die Handbuchseite [systemd.unit](#) zu lesen, um die Wirkungsweise der generischen Sektionen und ihrer Optionen zu verstehen.

Die folgenden Unit-Typen sind verfügbar, und sofern verlinkt, führt der Link zu einer ausführlicheren Beschreibung in unserem Handbuch:

1. **Dienste-Units** ([systemd.service](#)), die Daemons und die Prozesse, aus denen sie bestehen, starten und steuern.
2. **Socket-Units** ([systemd.socket](#)), die lokale IPC- oder Netzwerk-Sockets in dem System kapseln, nützlich für Socket-basierte Aktivierung.
3. **Target-Units** ([systemd.target](#)) sind für die Gruppierung von Units nützlich. Sie stellen während des Systemstarts auch als Runlevel bekannte Synchronisationspunkte zur Verfügung.

4. **Geräte-Units** (`systemd.device`) legen Kernel-Geräte (alle Block- und Netzwerkgeräte) in systemd offen und können zur Implementierung Geräte-basierter Aktivierung verwandt werden.
5. **Mount-Units** (`systemd.mount`) steuern Einhängpunkte im Dateisystem.
6. **Automount-Units** (`systemd.automount`) stellen Selbsteinhänge-Fähigkeiten bereit, für bedarfsgesteuertes Einhängen von Dateisystemen sowie parallelisiertem Systemstart.
7. **Zeitgeber-Units** (`systemd.timer`) sind für das Auslösen der Aktivierung von anderen Units basierend auf Zeitgebern nützlich.
8. **Auslagerungs-Units** (`systemd.swap`) sind ähnlich zu Einhäng-Units und kapseln Speicherauslagerungspartitionen oder -dateien des Betriebssystems.
9. **Pfad-Units** (`systemd.path`) können zur Aktivierung andere Dienste, wenn sich Dateisystemobjekte ändern oder verändert werden, verwandt werden.
10. **Slice-Units** (`systemd.slice`) können zur Gruppierung von Units, die Systemprozesse (wie Dienste- und Bereichs-Units) in einem hierarchischen Baum aus Ressourcenverwaltungsgründen verwalten, verwandt werden.
11. **Scope-Units** (`systemd.scope`) sind ähnlich zu Dienste-Units, verwalten aber fremde Prozesse, statt sie auch zu starten.

1.2.2 systemd im Dateisystem

Die Unit-Dateien, die durch den Paketverwalter der Distribution installiert wurden, befinden sich im Verzeichnis `/lib/systemd/system/`. Selbst erstellte Unit-Dateien legen wir im Verzeichnis `/usr/local/lib/systemd/system/` ab. (Ggf. ist das Verzeichnis zuvor mit dem Befehl `mkdir -p /usr/local/lib/systemd/system/` anzulegen.)

Die Steuerung des Status (enabled, disabled) einer Unit erfolgt über Symlink im Verzeichnis `/etc/systemd/system/`.

Das Verzeichnis `/run/systemd/system/` beinhaltet zur Laufzeit erstellte Unit-Dateien.

1.2.3 Weitere Funktionen von systemd

Systemd bietet noch weitere Funktionen. Eine davon ist [logind](#) als Ersatz für das nicht mehr weiter gepflegte *ConsoleKit*. Damit steuert systemd Sitzungen und Energiemanagement. Nicht zuletzt bietet systemd eine Menge an weiteren Möglichkeiten wie beispielsweise das Aufspannen eines Containers (ähnlich einer Chroot) mittels [systemd-nspawn](#) und viele weitere. Ein Blick in die Linkliste auf [Freedesktop](#) ermöglicht weitere Entdeckungen, unter anderem auch die ausführliche Dokumentation von Lennart Poettering zu systemd.

1.2.4 Handhabung von Diensten

Einer der Jobs von systemd ist es Dienste zu starten, zu stoppen oder sonstwie zu steuern. Dazu dient der Befehl `systemctl`.

- `systemctl --all` - listet alle Units, aktive und inaktive.
- `systemctl -t [NAME]` - listet nur Units des bezeichneten Typ.
- `systemctl list-units` - listet alle aktiven Units.
- `systemctl start [NAME...]` - startet eine oder mehrere Units.
- `systemctl stop [NAME...]` - stoppt eine oder mehrere Units.
- `systemctl restart [NAME]` - stoppt eine Unit und startet sie sofort wieder. Wird z.B. verwendet um die geänderte Konfiguration eines Dienstes neu einzuladen.
- `systemctl status [Name]` - zeigt den derzeitigen Status einer Unit.
- `systemctl is-enabled [Name]` - zeigt nur den Wert "enabled" oder "disabled" des Status einer Unit.

Die beiden folgenden Befehle integrieren bzw. entfernen die Unit anhand der Konfiguration ihrer Unit-Datei. Dabei werden Abhängigkeiten zu anderen Unit beachtet und ggf. Standardabhängigkeiten hinzugefügt damit systemd die Dienste und Prozesse fehlerfrei ausführen kann.

- `systemctl enable [NAME]` - gliedert eine Unit in systemd ein.
- `systemctl disable [NAME]` - entfernt eine Unit aus systemd.

Oft ist es nötig, `systemctl start` und `systemctl enable` für eine Unit durchzuführen, um sie sowohl sofort als auch nach einem Reboot verfügbar zu machen. Beide Optionen vereint der Befehl:

- `systemctl enable --now [NAME]`

Nachfolgend zwei Befehle deren Funktion unsere Handbuchseite [Systemd-Target](#) beschreibt.

- `systemctl reboot` – Führt einen Reboot aus
- `systemctl poweroff` - Führt das System herunter und schaltet den Strom, sofern technisch möglich, aus.

Beispiel

Anhand von Bluetooth demonstrieren wir die Funktionalität des systemd.

Zuerst die Statusabfrage im Kurzformat.

```
# systemctl is-enabled bluetooth.service
enabled
```

Nun Suchen wir nach den Unit-Dateien, dabei kombinieren wir `systemctl` mit `grep`:

```
# systemctl list-unit-files | grep blue
bluetooth.service          enabled          enabled
dbus-org.bluez.service     alias           -
bluetooth.target           static          -
```

Anschließend deaktivieren wir die Unit `“bluetooth.service”`.

```
# systemctl disable bluetooth.service
Synchronizing state of bluetooth.service with SysV service script
with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install disable bluetooth
Removed /etc/systemd/system/dbus-org.bluez.service.
Removed /etc/systemd/system/bluetooth.target.wants/bluetooth.
service.
```

In der Ausgabe ist gut zu erkennen, dass die Link (nicht die Unit-Datei selbst) entfernt wurden. Damit startet der `“bluetooth.service”` beim Booten des PC/Laptop nicht mehr automatisch. Zur Kontrolle fragen wir den Status nach einem Reboot ab.

```
# systemctl is-enabled bluetooth.service
disabled
```

Um eine Unit nur zeitweise zu deaktivieren, verwenden wir den Befehl

```
# systemctl stop <unit>
```

Damit bleibt die Konfiguration in systemd erhalten. Mit dem entsprechenden “start”-Befehl aktivieren wir die Unit wieder.

1.2.5 Quellen systemd

[Deutsche Manpage ‘systemd’](#)

[Deutsche Manpage ‘systemd.unit’](#)

[Deutsche Manpage ‘systemd.syntax’](#)

Seite zuletzt aktualisiert 2021-04-06

% Systemd - Unit-Datei

ANFANG INFOBEREICH FÜR DIE AUTOREN

Dieser Bereich ist vor der Veröffentlichung zu entfernen !!!

Status: RC2

Änderungen 2021-04:

- Neu “systemd-unit”
- Für die Verwendung mit pandoc optimiert.

ENDE INFOBEREICH FÜR DIE AUTOREN

1.3 systemd unit-Datei

Die grundlegenden und einführenden Informationen zu Systemd enthält die Handbuchseite [Systemd-Start](#)

In der vorliegenden Handbuchseite erklären wir den Aufbau der **Unit-Dateien** und die generischen Sektionen “[Unit]” und “[Install]”.

Die Unit-Datei ist eine reine Textdatei im INI-Format. Sie enthält Konfigurationsanweisungen von der Art “*Schlüssel=Wert*” in verschiedene Sektionen. Leere Zeilen und solche, die mit “#” oder “;” beginnen, werden ignoriert. Alle Unit-Dateien müssen eine Sektion entsprechend des Unit Typ enthalten. Die generischen Sektionen “[Unit]” am Beginn und “[Install]” am Ende der Datei sind optional, wobei die Sektion “[Unit]” dringend empfohlen wird.

1.3.1 Ladepfad der Unit-Dateien

Die Ausgabe zeigt die Reihenfolge der Verzeichnisse, aus denen die Unit-Dateien geladen werden.

```
# systemd-analyze unit-paths
/etc/systemd/system.control
/run/systemd/system.control
/run/systemd/transient
/run/systemd/generator.early
/etc/systemd/system
/etc/systemd/system.attached
/run/systemd/system
/run/systemd/system.attached
/run/systemd/generator
/usr/local/lib/systemd/system
/lib/systemd/system
/usr/lib/systemd/system
/run/systemd/generator.late
```

Unit-Dateien, die in früher aufgeführten Verzeichnissen gefunden werden, setzen Dateien mit dem gleichen Namen in Verzeichnissen, die weiter unten in der Liste aufgeführt sind, außer Kraft. So hat eine Datei in “/etc/systemd/system” Vorrang vor der gleichnamigen in “/lib/systemd/system”.

Nur ein Teil der zuvor aufgeführten Verzeichnisse existiert per default in siduction. Die Verzeichnisse

- **/lib/systemd/system/**
beinhalten System-Units, die durch den Paketverwalter der Distribution installiert wurden und ggf. vom Administrator erstellte Unit-Dateien.
- **/etc/systemd/system/**
beinhalten Symlinks auf Unit-Dateien in */lib/systemd/system/* für aktivierte Units und ggf. vom Administrator erstellte Unit-Dateien.
- **/usr/local/lib/systemd/system/**
dieses Verzeichnis muss erstellt werden und ist für vom Administrator erstellte Unit-Dateien vorgesehen.
- **/run/systemd/**
beinhalten Laufzeit-Units und dynamische Konfiguration für flüchtige Units. Für den Administrator hat dieses Verzeichnis ausschließlich informellen Wert.

Wir empfehlen eigene Unit-Dateien in */usr/local/lib/systemd/system/* abzulegen.

1.3.2 Aktivierung der Unit-Datei

Um systemd die Konfiguration einer Unit zugänglich zu machen, muss die Unit-Datei aktiviert werden. Dies geschieht mit dem Aufruf:

```
# systemctl daemon-reload
# systemctl enable --now <UNIT_DATEI>
```

Der erste Befehl lädt die komplette Daemon-Konfiguration neu, der zweite startet die Unit sofort (Option “–now”) und gliedert sie in systemd ein, sodass sie bei jedem Neustart des PC ausgeführt wird.

Der Befehl

```
# systemctl disable <UNIT_DATEI>
```

bewirkt, dass sie nicht mehr bei jedem Neustart des PC ausgeführt wird. Sie kann aber weiterhin manuell mit dem Befehl `systemctl start <UNIT_DATEI>` gestartet und mit `systemctl stop <UNIT_DATEI>` gestoppt werden.

Falls eine Unit-Datei leer ist (d.h. die Größe 0 hat) oder ein Symlink auf */dev/null* ist, wird ihre Konfiguration nicht geladen und sie erscheint mit einem Ladezustand “masked” und kann nicht aktiviert werden. Dies ist eine wirksame Methode um eine

Unit komplett zu deaktivieren und es auch unmöglich zu machen, sie manuell zu starten.

1.3.3 Sektionen der Unit-Datei

Die Unit-Datei besteht in der Regel aus der Sektionen [Unit], der Typ spezifischen Sektion und der Sektion [Install]. Die Typ spezifische Sektion fließt als Suffix in den Dateinamen ein. So besitzt zum Beispiel eine Unit-Datei, die einen Zeitgeber konfiguriert, immer die Endung *.timer* und muss "[Timer]" als Typ spezifische Sektion enthalten.

1.3.3.1 Sektion [Unit] Diese Sektion enthält allgemeine Informationen über die Unit, definiert Abhängigkeiten zu anderen Units, wertet Bedingungen aus und sorgt für die Einreihung in den Bootprozess.

1. Allgemeine Optionen

a. "Description="

Identifiziert die Unit durch einen menschenlesbaren Namen, der von systemd als Bezeichnung für die Unit verwandt wird und somit im systemjournal erscheint ("Starting *description*...") und dort als Suchmuster verwandt werden kann.

b. "Documentation="

Ein Verweis auf eine Datei oder Webseite, die Dokumentation für diese Unit oder ihre Konfiguration referenzieren. Z. B.: "Documentation=man:cupsd(8)" oder "Documentation=http://www.cups.org/doc/man-cupsd.html".

2. Bindungsabhängigkeiten zu anderen Units

a. "Wants="

Hier aufgeführte Units werden mit der konfigurierten Unit gestartet.

b. "Requires="

Ähnlich zu *Wants=*, erklärt aber eine stärkerere Bindung an die aufgeführten Units.

Wenn diese Unit aktiviert wird, werden die aufgeführten Units ebenfalls aktiviert.

Schlägt die Aktivierung einer der anderen Units fehl **und** die Ordnungsabhängigkeit *After=* ist auf die fehlgeschlagene Unit gesetzt, dann wird

diese Unit nicht gestartet.

Falls eine der anderen Units inaktiv wird, bleibt diese Unit aktiv, nur wenn eine der anderen Units gestoppt wird, wird diese Unit auch gestoppt.

c. *"Requisite="*

Ähnlich zu *Requires=*. Der Start dieser Unit wird sofort fehlschlagen, wenn die hier aufgeführten Units noch nicht gestartet wurden. *Requisite=* sollte mit der Ordnungsabhängigkeit *After=* kombiniert werden, um sicherzustellen, dass diese Unit nicht vor der anderen Unit gestartet wird.

d. *"BindsTo="*

BindsTo= ist der stärkste Abhängigkeitstyp: Es bewirkt zusätzlich zu den Eigenschaften von *Requires=*, dass die gebundene Unit im aktiven Status sein muss, damit diese Unit auch aktiv sein kann.

Beim Stoppen oder inaktivem Zustand der gebundenen Unit wird diese Unit immer gestoppt.

Um zu verhindern, dass der Start dieser Unit fehlschlägt, wenn die gebundene Unit nicht, oder noch nicht, in einem aktiven Zustand ist, sollte *BindsTo=* am besten mit der Ordnungsabhängigkeit *After=* kombiniert werden.

e. *"PartOf="*

Ähnlich zu *Requires=*, aber begrenzt auf das Stoppen und Neustarten von Units.

Wenn Systemd die hier aufgeführten Units stoppt oder neustartet, wird die Aktion zu dieser Unit weitergeleitet.

Das ist eine Einwegeabhängigkeit. Änderungen an dieser Unit betreffen nicht die aufgeführten Units.

f. *"Conflicts="*

Deklariert negative Anforderungsabhängigkeiten. Die Angabe einer durch Leerzeichen getrennten Liste ist möglich.

Conflicts= bewirkt, dass die aufgeführte Unit gestoppt wird, wenn diese Unit startet und umgekehrt.

Da *Conflicts=* keine Ordnungsabhängigkeit beinhaltet, muss eine Abhängigkeit *After=* oder *Before=* erklärt werden, um sicherzustellen, dass die in Konflikt stehende Unit gestoppt wird, bevor die andere Unit gestartet wird.

3. Ordnungsabhängigkeiten zu anderen Units

a. “*Before=*”

Diese Einstellung konfiguriert Ordnungsabhängigkeiten zwischen Units. *Before=* stellt sicher, dass die aufgeführte Unit erst mit dem Starten beginnt, nachdem der Start der konfigurierte Unit abgeschlossen ist. Die Angabe einer durch Leerzeichen getrennten Liste ist möglich.

b. “*After=*”

Diese Einstellung stellt das Gegenteil von *Before=* sicher. Die aufgeführte Unit muss vollständig gestartet sein, bevor die konfigurierte Unit gestartet wird.

c. “*OnFailure=*”

Units, die aktiviert werden, wenn diese Unit den Zustand »failed« einnimmt.

4. Bedingungen

Unit-Dateien können auch eine Reihe von Bedingungen enthalten.

Bevor die Unit gestartet wird, wird Systemd nachweisen, dass die festgelegten Bedingungen wahr sind. Falls nicht, wird das Starten der Unit (fast ohne Ausgabe) übersprungen.

Fehlschlagende Bedingungen führen nicht dazu, dass die Unit in den Zustand »failed« überführt wird.

Falls mehrere Bedingungen festgelegt sind, wird die Unit ausgeführt, falls alle von ihnen zutreffen.

In diesem Abschnitt führen wir nur Bedingungen auf, die uns für selbst erstellte Units hilfreich erscheinen, denn viele Bedingungen dienen dazu, um Units zu überspringen, die auf dem lokalen System nicht zutreffen.

Der Befehl `systemd-analyze verify <UNIT_DATEI>` kann zum Testen von Bedingungen verwandt werden.

a. “*ConditionVirtualization=*”

Prüft, ob das System in einer virtualisierten Umgebung ausgeführt wird und testet optional, ob es eine bestimmte Implementierung ist.

b. “*ConditionACPower=*”

Prüft, ob das System zum Zeitpunkt der Aktivierung der Unit am Netz hängt oder ausschließlich über Akku läuft.

c. “*ConditionPathExists=*”

Prüft auf die Existenz einer Datei. Mit einem Ausrufezeichen (“!”) vor

dem Pfad wird der Test negiert.

- d. *“ConditionPathExistsGlob=”*
Wie zuvor, nur dass ein Suchmuster angegeben wird. Mit einem Ausrufezeichen (“!”) vor dem Pfad wird der Test negiert.
- e. *“ConditionPathIsDirectory=”*
Prüft auf die Existenz eines Verzeichnisses. Mit einem Ausrufezeichen (“!”) vor dem Pfad wird der Test negiert.
- f. *“ConditionPathIsSymbolicLink=”*
Überprüft ob ein bestimmter Pfad existiert und ein symbolischer Link ist. Mit einem Ausrufezeichen (“!”) vor dem Pfad wird der Test negiert.
- g. *“ConditionPathIsMountPoint=”*
Überprüft ob ein bestimmter Pfad existiert und ein Einhängepunkt ist. Mit einem Ausrufezeichen (“!”) vor dem Pfad wird der Test negiert.
- h. *“ConditionPathIsReadWrite=”*
Überprüft ob das zugrundeliegende Dateisystem les- und schreibbar ist. Mit einem Ausrufezeichen (“!”) vor dem Pfad wird der Test negiert.
- i. *“ConditionDirectoryNotEmpty=”*
Überprüft ob ein bestimmter Pfad existiert und ein nicht leeres Verzeichnis ist. Mit einem Ausrufezeichen (“!”) vor dem Pfad wird der Test negiert.
- j. *“ConditionFileNotEmpty=”*
Überprüft ob ein bestimmter Pfad existiert und sich auf eine normale Datei mit einer von Null verschiedenen Größe bezieht. Mit einem Ausrufezeichen (“!”) vor dem Pfad wird der Test negiert.
- k. *“ConditionFileIsExecutable=”*
Überprüft ob ein bestimmter Pfad existiert und sich auf eine normale, als ausführbar gekennzeichnete Datei bezieht. Mit einem Ausrufezeichen (“!”) vor dem Pfad wird der Test negiert.

Die vollständige Dokumentation zu allen Optionen der Sektion “[Unit]” bitte in der [Deutschen Manpage, systemd.unit](#) nachlesen.

1.3.3.2 Typ spezifische Sektion Diese Sektion enthält die speziellen Optionen der elf möglichen Typen. Ausführliche Beschreibungen enthalten die verlinkten

Handbuchseiten, oder ersatzweise die jeweilige deutsche Manpage.

- [\[Service\]](#) konfiguriert einen Dienst
- [\[Socket\]](#) konfiguriert ein Socket
- [\[Device\]](#) konfiguriert ein Gerät
- [\[Mount\]](#) konfiguriert einen Einhängpunkt
- [\[Automount\]](#) konfiguriert einen Selbsteinhängpunkt
- [\[Swap\]](#) konfiguriert eine Auslagerungsdatei oder -partition
- [\[Target\]](#) konfiguriert ein Startziel
- [\[Path\]](#) konfiguriert einen überwachten Dateipfad
- [\[Timer\]](#) konfiguriert einen von systemd gesteuerten und überwachten Zeitgeber
- [\[Slice\]](#) konfiguriert eine Ressourcenverwaltungsscheibe
- [\[Scope\]](#) konfiguriert eine Gruppe von extern erstellten Prozessen.

1.3.3.3 Sektion **[Install]** Unit-Dateien können diese Sektion enthalten.

Die Optionen der *[Install]*-Sektion werden von den Befehlen `systemctl enable <UNIT_DATEI>` und `systemctl disable <UNIT_DATEI>` während der Installation einer Unit verwandt.

Unit-Dateien ohne *[Install]*-Sektion lassen sich manuell mit dem Befehl `systemctl start <UNIT_DATEI>`, oder von einer anderen Unit-Datei starten.

Beschreibung der Optionen:

- “*Alias=*”
Eine Liste von zusätzlichen Namen, unter der diese Unit installiert werden soll. Die hier aufgeführten Namen müssen die gleiche Endung wie die Unit-Datei haben.
- “*WantedBy=*”
Diese Option kann mehrfach verwendet werden oder eine durch Leerzeichen getrennte Liste enthalten.
Im *.wants/*-Verzeichnis jeder der aufgeführten Units wird bei der Installation ein symbolischer Link erstellt. Dadurch wird eine Abhängigkeit vom Typ *Wants=* von der aufgeführten Unit zu der aktuellen Unit hinzugefügt. Das

Hauptergebnis besteht darin, dass die aktuelle Unit gestartet wird, wenn die aufgeführte Unit gestartet wird.

Verhält sich wie die Option *Wants=* in der Sektion *[Unit]*.

Beispiel:

WantedBy=graphical.target

Das teilt systemd mit, die Unit beim Starten von graphical.target (früher "init 5") hereinzuziehen.

- *"RequiredBy="*
Diese Option kann mehrfach verwendet werden oder eine durch Leerzeichen getrennte Liste enthalten.
Im *.requires/-*Verzeichnis jeder der aufgeführten Units wird bei der Installation ein symbolischer Link erstellt. Dadurch wird eine Abhängigkeit vom Typ *Requires=* von der aufgeführten Unit zu der aktuellen Unit hinzugefügt. Das Hauptergebnis besteht darin, dass die aktuelle Unit gestartet wird, wenn die aufgeführte Unit gestartet wird.
Verhält sich wie die Option *Requires=* in der Sektion *[Unit]*.
- *"Also="*
Zusätzliche Units, die installiert/deinstalliert werden sollen, wenn diese Unit installiert/deinstalliert wird.
- *"DefaultInstance="*
Diese Option zeigt nur bei Vorlagen-Unit-Dateien Wirkung.
Deklariert, welche Instanz der Unit freigegeben werden soll. Die angegebene Zeichenkette muss zur Identifizierung einer Instanz geeignet sein.

Hinweis: Um die Konfiguration einer Unit-Datei zu prüfen, eignet sich der Befehl **systemd-analyze verify <UNIT_DATEI>**.

1.3.4 Beispiel cupsd

Der *cupsd*, Auftragsplaner (Scheduler) für das Common UNIX Printing System, wird von systemd mit seinen drei Unit Dateien "*cups.socket*", "*cups.service*" und "*cups.path*" gesteuert und eignet sich gut, um die Abhängigkeiten zu verdeutlichen. Hier die drei Dateien.

```
Datei /lib/systemd/system/cups.service:  
  
[Unit]
```

```
Description=CUPS Scheduler
Documentation=man:cupsd(8)
After=network.target sssd.service ybind.service nslcd.service
Requires=cups.socket
    After=cups.socket (nicht in der Datei, da implizit vorhanden.)
    After=cups.path (nicht in der Datei, da implizit vorhanden.)

[Service]
ExecStart=/usr/sbin/cupsd -l
Type=notify
Restart=on-failure

[Install]
Also=cups.socket cups.path
WantedBy=printer.target
```

Datei /lib/systemd/system/cups.path:

```
[Unit]
Description=CUPS Scheduler
PartOf=cups.service
    Before=cups.service (nicht in der Datei, da implizit vorhanden,
    .)

[Path]
PathExists=/var/cache/cups/org.cups.cupsd

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

Datei /lib/systemd/system/cups.socket:

```
[Unit]
Description=CUPS Scheduler
PartOf=cups.service
    Before=cups.service (nicht in der Datei, da implizit vorhanden,
    .)

[Socket]
ListenStream=/run/cups/cups.sock

[Install]
WantedBy=sockets.target
```

Die Sektion [Unit]

enthält für alle drei Dateien die gleiche Beschreibung. Die Dateien *cups.path* und *cups.socket* zusätzlich die Bindungsabhängigkeit *PartOf=cups.service*, was bedeutet, dass diese zwei Units abhängig von *cups.service* gestoppt oder neu gestartet werden.

Die socket-Unit ebenso wie die path-Unit schließen die Ordnungsabhängigkeit “Before=” zu ihrer namensgleichen service-Unit ein. Deshalb ist es nicht notwendig in der *cups.service*-Unit die Ordnungsabhängigkeiten “After=cups.socket” und “After=cups.path” einzutragen. (Siehe unten die Ausgabe von “systemd-analyze dump” mit dem Vermerk “destination-implicit”.) Beide Abhängigkeiten gemeinsam bewirken, dass unabhängig davon welche Unit zuerst startet, immer alle drei Units starten und die *cups.service*-Unit erst, nachdem der Start der *cups.path*-Unit und der *cups.socket*-Unit erfolgreich abgeschlossen wurde.

Die vollständige Konfiguration der Units erhalten wir mit dem Befehl **systemd-analyze dump**, der eine sehr, sehr lange Liste (> 32000 Zeilen) des systemd Serverstatus ausgibt.

```
# systemd-analyze dump
[...]
-> Unit cups.service:
  Description: CUPS Scheduler.service
  [...]
  WantedBy: printer.target (destination-file)
  ConsistsOf: cups.socket (destination-file)
  ConsistsOf: cups.path (destination-file)
  Before: printer.target (destination-default)
  After: cups.socket (destination-implicit)
  After: cups.path (destination-implicit)
[...]
-> Unit printer.target:
  Description: Printer
  [...]
  Wants: cups.service (origin-file)
  After: cups.service (origin-default)
[...]
```

Die Sektion [Install]

der *cups.service*-Unit enthält mit der Option “Also=cups.socket cups.path” die Anweisung, diese beiden Units auch zu installieren und alle drei Units haben unterschiedliche “WantedBy=” Optionen:

- cups.socket: WantedBy=sockets.target

- `cups.path`: `WantedBy=multi-user.target`
- `cups.service`: `WantedBy=printer.target`

Um zu verstehen, warum unterschiedliche Werte für die Option “WantedBy=” Verwendung finden, benötigen wir zusätzliche Informationen, die wir mit den Befehlen `systemd-analyze dot` und `systemd-analyze plot` erhalten.

```
$ systemd-analyze dot --to-pattern='*.target' --from-pattern=\
    '*.target' | dot -Tsvg > targets.svg

$ systemd-analyze plot > bootup.svg
```

Der erste liefert uns ein Flussdiagramm mit den Abhängigkeiten der verschiedenen *Targets* zueinander und der zweite eine graphisch aufbereitete Auflistung des Bootprozesses mit den Zeitpunkten wann ein Prozess gestartet wurde, welche Zeit er beanspruchte und seinen Aktivitätszustand.

Der *targets.svg* und der *bootup.svg* entnehmen wir, dass

1. **sysinit.target**
aktiviert wird und
2. **basic.target**
erst startet, wenn *sysinit.target* erreicht wurde.
 1. **sockets.target**
von *basic.target* angefordert wird,
 1. **cups.socket**
und alle weiteren *.socket*-Units von *sockets.target* hereingeholt werden.
 2. **paths.target**
von *basic.target* angefordert wird,
 1. **cups.path**
und alle weiteren *.path*-Units von *paths.target* hereingeholt werden.
3. **network.target**
erst startet, wenn *basic.target* erreicht wurde.

4. **cups.service**
erst startet, wenn *network.target* erreicht wurde.
5. **multi-user.target**
erst startet, wenn *network.target* erreicht wurde.
6. **multi-user.target**
erst dann erreicht wird, wenn *cups.service* erfolgreich gestartet wurde.
(Genau genommen liegt es daran, dass der *cups-browsed.service*, der vom *cups.service* abhängt, erfolgreich gestartet sein muss.)
7. **printer.target**
wird erst aktiv, wenn Systemd dynamisch Geräte-Units für die Drucker generiert.
Dazu müssen die Drucker angeschlossen und eingeschaltet sein.

Weiter oben stellten wir fest, dass der Start einer *cups.xxx*-Unit ausreicht, um alle drei Units hereinzuholen. Betrachten wir noch einmal die "WantedBy"-Optionen in der [Install]-Sektion, so haben wir die *cups.socket*-Unit, die über das *sockets.target* bereits während des *basic.target* hereingeholt wird, die *cups.path*-Unit, die während des *multi-user.target* hereingeholt wird und den *cups.service*, der vom *printer.target* hereingeholt wird.

Während des gesamten Bootprozesses werden die drei *cups.xxx*-Units wiederholt bei systemd zur Aktivierung angefordert. Das härtet den *cupsd* gegen unvorhergesehene Fehler, spielt für systemd aber keine Rolle, denn es ist unerheblich wie oft ein Service angefordert wird, wenn er sich in der Warteschlange befindet.

Zusätzlich fordert immer dann das *printer.target* den *cups.service* an, wenn ein Drucker neu von systemd erkannt wird.

1.3.5 Werkzeuge

Systemd beinhaltet einige nützliche Werkzeuge für die Analyse, Prüfung und Bearbeitung der Unit-Dateien.

Bitte auch die Manpages [systemd-analyze](#) und [systemctl](#) zu Rate ziehen.

- edit

```
# systemctl edit <UNIT_DATEI>
# systemctl edit --full <UNIT_DATEI>
# systemctl edit --full --force <UNIT_DATEI>
```

systemctl edit öffnet die ausgewählte Unit-Datei im konfigurierten Editor.

systemctl edit erstellt unterhalb */etc/systemd/system/* ein neues Verzeichnis mit dem Namen "*<UNIT_DATEI>.d*" und darin die Datei "override.conf", die ausschließlich die Änderungen gegenüber der ursprünglichen Unit-Datei enthält. Dies gilt für alle Unit-Dateien in den Verzeichnissen, die in der [Hierarchie der Ladepfade](#) inklusive */etc/systemd/system/* abwärts eingetragen sind.

systemctl edit -full erstellt eine neue, namensgleiche Datei im Verzeichnis */etc/systemd/system/*. Dies gilt für alle Unit-Dateien in den Verzeichnissen, die in der [Hierarchie der Ladepfade](#) unterhalb */etc/systemd/system/* eingetragen sind. Dateien, die sich bereits im Verzeichnis */etc/systemd/system/* befinden, werden überschrieben.

systemctl edit -full -force erstellt eine neue Datei im Verzeichnis */etc/systemd/system/*. Ohne die Option *-full* würde nur eine Datei "override.conf" im neuen Verzeichnis */etc/systemd/system/<UNIT_DATEI>.d* generiert, der die zugehörige Unit-Datei fehlt.

Wird der Editor beendet, so führt *systemd* automatisch den Befehl **systemctl daemon-reload** aus.

- revert

```
# systemctl revert <UNIT_DATEI>
```

macht die mit *systemctl edit* und *systemctl edit -full* vorgenommenen Änderungen an Unit-Dateien rückgängig. Dies gilt nicht für geänderte Unit-Dateien die sich bereits im Verzeichnis */etc/systemd/system/* befanden. Zusätzlich bewirkt der Befehl die Rücknahme der mit *systemctl mask* vorgenommenen Änderungen.

- daemon-reload

```
# systemctl daemon-reload
```

Lädt die Systemverwalterkonfiguration neu. Dies führt alle Generatoren neu aus, lädt alle Unit-Dateien neu und erstellt den gesamten Abhängigkeitsbaum neu.

- cat

```
$ systemctl cat <UNIT_DATEI>
```

Gibt entsprechend des Konsolebefehls *cat* den Inhalt der Unit-Datei und aller zugehörigen Änderungen aus.

- analyze verify

```
$ systemd-analyze verify <UNIT_DATEI>
```

überprüft die Konfigurationseinstellungen einer Unit-Datei und gibt Hinweise aus. Dies ist ein sehr hilfreicher Befehl um die Konfiguration selbst erstellter oder geänderter Unit-Dateien zu prüfen.

- systemd-delta

```
$ systemd-delta
```

präsentiert in der Ausgabe Unit-Dateien und die vorgenommenen Änderungen an ihnen. Das Schlüsselwort am Anfang der Zeile definiert die Art der Änderung bzw. Konfiguration.

Hier ein Beispiel:

```
$ systemd-delta --no-pager
[MASKED]      /etc/sysctl.d/50-coredump.conf → /usr/lib/sysctl.d/50-
coredump.conf
[OVERRIDDEN] /etc/tmpfiles.d/screen-cleanup.conf → /usr/lib/
tmpfiles.d/screen-cleanup.conf
[MASKED]      /etc/systemd/system/NetworkManager-wait-online.service
→ /lib/systemd/system/NetworkManager-wait-online.service
[EQUIVALENT] /etc/systemd/system/tmp.mount → /lib/systemd/system/
tmp.mount
[EXTENDED]   /lib/systemd/system/rc-local.service → /lib/systemd/
system/rc-local.service.d/debian.conf
[EXTENDED]   /lib/systemd/system/systemd-locale.service → /lib/
systemd/system/systemd-locale.service.d/locale-gen.conf

6 overridden configuration files found.
```

- analyze dump

```
$ systemd-analyze dump > systemd_dump.txt
```


erstellt die Textdatei *systemd_dump.txt* mit der vollständigen Konfiguration aller Units des systemd. Die sehr lange Textdatei gibt Aufschluss über alle Konfigurationseinstellungen aller systemd-Units und lässt sich mit einem Texteditor und unter Verwendung von RegEx-Pattern gut durchsuchen.

- analyze plot

```
$ systemd-analyze plot > bootup.svg
```

erstellt die Datei *bootup.svg* mit der zeitlichen Abfolge des Bootprozesses. Es ist eine graphisch aufbereitete Auflistung des Bootprozesses mit den Start- und Endzeitpunkten aller Units, welche Zeit sie beanspruchten und ihren Aktivitätszuständen.

- analyze dot

```
$ systemd-analyze dot --to-pattern='*.target' --from-pattern=\
'*.target' | dot -Tsvg > targets.svg
Color legend: black      = Requires
                dark blue = Requisite
                dark grey = Wants
                red       = Conflicts
                green     = After
```

erstellt das Flussdiagramm *targets.svg*, dass die Abhängigkeiten der im Bootprozess verwendeten Targets darstellt. Die Beziehungen der *.target*-Units werden zur besseren Übersicht farblich dargestellt.

Die hier genannten Hilfsmittel stellen nur einen Teil der mit systemd ausgelieferten Werkzeuge dar. Bitte entnehme den man-Pages die vollständige Dokumentation.

1.3.6 Quellen systemd-unit-Datei

[Deutsche Manpage, systemd.unit](#)
[Deutsche Manpage, systemd.syntax](#)
[Deutsche Manpage, systemd.device](#)
[Deutsche Manpage, systemd.scope](#)
[Deutsche Manpage, systemd.slice](#)
[Deutsche Manpage, systemd.socket](#)
[Deutsche Manpage, systemd.swap](#)
[Deutsche Manpage, systemd-analyze](#)
[Deutsche Manpage, systemctl](#)

Dank an Helge Kreuzmann für die deutschen Übersetzungen.

Seite zuletzt aktualisiert 2021-04-06

% Systemd - service

ANFANG INFOBEREICH FÜR DIE AUTOREN

Dieser Bereich ist vor der Veröffentlichung zu entfernen !!!

Status: RC2

Änderungen 2021-04:

- Neu "systemd-service"
- Für die Verwendung mit pandoc optimiert.

ENDE INFOBEREICH FÜR DIE AUTOREN

1.4 systemd-service

Die grundlegenden und einführenden Informationen zu Systemd enthält die Handbuchseite [Systemd-Start](#). Die alle Unit-Dateien betreffenden Sektionen *[Unit]* und *[Install]* behandelt unsere Handbuchseite [Systemd Unit-Datei](#). In der vorliegenden Handbuchseite erklären wir die Funktion der Unit **systemd.service**. Die Unit-Datei mit der Namensendung “.service” ist der am häufigsten anzutreffende Unit-Typ in systemd.

Die Servic-Unit-Datei muss eine Sektion *[Service]* enthalten, die Informationen über den Dienst und den Prozess, den er überwacht, konfiguriert.

1.4.1 service-Unit anlegen

Selbst erstellte Unit-Dateien legen wir vorzugsweise im Verzeichnis `/usr/local/lib/systemd/system/` ab. (Ggf. ist das Verzeichnis mit dem Befehl `mkdir -p /usr/local/lib/systemd/system/` anzulegen.) Das hat den Vorteil, dass sie Vorrang gegenüber den System-Units, die durch den Paketverwalter der Distribution installiert wurden, erhalten und gleichzeitig Steuerungslinks sowie Änderungsdateien, die mit `systemctl edit <UNIT_DATEI>` erzeugt wurden, im seinerseits vorrangigen Verzeichnis `/etc/systemd/system/` abgelegt werden. Siehe: [Hierarchie der Ladepfade](#).

1.4.2 Sektion *[Service]*

Für diese Sektion sind über dreißig Optionen verfügbar, von denen wir hier besonders häufig verwendete beschreiben.

Type=	PIDFile=
RemainAfterExit=	GuessMainPID=
ExecStart=	Restart=
ExecStartPre=	RestartSec=
ExecStartPost=	SuccessExitStatus=
ExecCondition=	RestartPreventExitStatus=
ExecReload=	RestartForceExitStatus=
ExecStop=	NonBlocking=
ExecStopPost=	NotifyAccess=
TimeoutStopSec=	RootDirectoryStartOnly=
TimeoutStartSec=	FileDescriptorStoreMax=

TimeoutAbortSec=	USBFunctionDescriptors=
TimeoutSec=	USBFunctionStrings=
RuntimeMaxSec=	Sockets=
WatchdogSec=	BusName=
	OOMPPolicy=

- **Type=**

Definiert den Prozessesstarttyp und ist damit eine der wichtigsten Optionen. Die möglichen Werte sind: *simple*, *exec*, *forking*, *oneshot*, *dbus*, *notify* oder *idle*.

Der Standard *simple* wird verwendet, falls *ExecStart=* festgelegt ist, aber weder *Type=* noch *BusName=* gesetzt sind.

- **simple**

Eine Unit vom Typ *simple* betrachtet systemd als erfolgreich gestartet, sobald der mit *ExecStart=* festgelegte Hauptprozess mittels *fork* gestartet wurde. Anschließend beginnt systemd sofort mit dem Starten von nachfolgenden Units, unabhängig davon, ob der Hauptprozess erfolgreich aufgerufen werden kann.

- **exec**

Ähnelt *simple*, jedoch wartet systemd mit dem Starten von nachfolgenden Units bis der Hauptprozess erfolgreich beendet wurde. Das ist auch der Zeitpunkt, an dem die Unit den Zustand “active” erreicht.

- **forking**

Hier betrachtet systemd den Dienst als gestartet, sobald der mit *ExecStart=* festgelegte Prozess sich in den Hintergrund verzweigt und das übergeordnete System sich beendet. Dieser Typ findet oft bei klassischen Daemons Anwendung. Hier sollte auch die Option *PIDFile=* angegeben werden, damit das System den Hauptprozess weiter verfolgen kann.

- **oneshot**

Ähnelt *exec*. Die Option *Type=oneshot* kommt oft bei Skripten oder Befehlen zum Einsatz, die einen einzelnen Job erledigen und sich dann beenden. Allerdings erreicht der Dienst niemals den Zustand “active”, sondern geht sofort, nachdem sich der Hauptprozess beendet hat, vom Zustand “activating” zu “deactivating” oder “dead” über. Deshalb ist es

häufig sinnvoll diese Option mit “*RemainAfterExit=yes*” zu verwenden, um den Zustand “active” zu erreichen.

- **dbus**

Verhält sich ähnlich zu *simple*, *systemd* startet nachfolgende Units, nachdem der D-Bus-Busname erlangt wurde. Units mit dieser Option, erhalten implizit eine Abhängigkeit auf die Unit “*dbus.socket*”.

- **notify**

Der *Type=notify* entspricht weitestgehend dem *Type simple*, mit dem Unterschied, dass der Daemon ein Signal an *systemd* sendet, wenn er bereitsteht.

- **idle**

Das Verhalten von *idle* ist sehr ähnlich zu *simple*; allerdings verzögert *systemd* die tatsächliche Ausführung des Dienstes, bis alle aktiven Aufträge erledigt sind. Dieser Typ ist nicht als allgemeines Werkzeug zum Sortieren von Units nützlich, denn er unterliegt einer Zeitüberschreitung von 5 s, nach der der Dienst auf jeden Fall ausgeführt wird.

- **RemainAfterExit=**

Erwartet einen logischen Wert (Standard: *no*), der festlegt, ob der Dienst, selbst wenn sich alle seine Prozesse beendet haben, als aktiv betrachtet werden sollte. Siehe *Type=oneshot*.

- **GuessMainPID=**

Erwartet einen logischen Wert (Standard: *yes*). *Systemd* verwendet diese Option ausschließlich, wenn *Type=forking* gesetzt und *PIDFile=* nicht gesetzt ist, und versucht dann die Haupt-PID eines Dienstes zu raten, falls es sie nicht zuverlässig bestimmen kann. Für andere Typen oder mit gesetzter Option *PIDFile=* ist die Haupt-PID immer bekannt.

- **PIDFile=**

Akzeptiert einen Pfad zur PID-Datei des Dienstes. Für Dienste vom *Type=forking* wird die Verwendung dieser Option empfohlen.

- **BusName=**

Hier ist der D-Bus-Busname, unter dem dieser Dienst erreichbar ist, anzugeben. Die Option ist für Dienste vom *Type=dbus* verpflichtend.

- **ExecStart=**

Enthält Befehle mit ihren Argumenten, die ausgeführt werden, wenn diese

Unit gestartet wird. Es muss genau ein Befehl angegeben werden, außer die Option *Type=oneshot* ist gesetzt, dann kann *ExecStart=* mehrfach verwendet werden. Der Wert von *ExecStart=* muss den in der deutsche Manpage [systemd.service](#) detailliert beschriebenen Regeln entsprechen.

- **ExecStop=**
Kann mehrfach verwendet werden und enthält Befehle, die dem Stoppen eines mittels *ExecStart=* gestarteten Dienstes, dienen. Die Syntax ist identisch zu *ExecStart=*.
- **ExecStartPre=, ExecStartPost=, ExecStopPost=**
Zusätzliche Befehle, die vor bzw. nach dem Befehl in *ExecStart=* oder *ExecStop* gestartet werden. Auch hier ist die Syntax identisch zu *ExecStart=*. Es sind mehrere Befehlszeilen erlaubt und die Befehle werden seriell einer nach dem anderen ausgeführt. Falls einer dieser Befehle (dem nicht “-” vorangestellt ist) fehlschlägt, wird die Unit sofort als fehlgeschlagen betrachtet.
- **RestartSec=**
Bestimmt die vor dem Neustart eines Dienstes zu schlafende Zeit. Eine einheitenfreie Ganzzahl definiert Sekunden, eine Angabe von “3min 4s” ist auch möglich.
Die Art der Zeitwertdefinition gilt für alle zeitgesteuerten Optionen.
- **TimeoutStartSec=, TimeoutStopSec=, TimeoutSec=**
Bestimmt die Zeit, die auf das Starten bzw. Stoppen gewartet werden soll. *TimeoutSec=* vereint die beiden zuvor genannten Optionen.
TimeoutStopSec= konfiguriert zusätzlich die Zeit, die, soweit vorhanden, für jeden *ExecStop=-*Befehl gewartet werden soll.
- **Restart=**
Konfiguriert, ob der Dienst neu gestartet werden soll, wenn der Dienstprozess sich beendet, getötet oder eine Zeitüberschreitung erreicht wird. Wenn der Tod des Prozesses das Ergebnis einer Systemd-Aktion ist, wird der Dienst nicht neu gestartet.
Die erlaubten Werte sind: no, always, on-success, on-failure, on-abnormal, on-abort oder on-watchdog.
Folgende Tabelle zeigt den Effekt der *Restart=* Einstellung auf die Exit-Gründe.

		on	on	on	on	on
► Restart= ►	always	success	failure	abnormal	abort	watchdog

▼ Exit-Grund ▼					
Sauberer Exit	X	X			
Unsauberer Exit	X		X		
Unsauberes Signal	X		X	X	X
Zeitüberschreitung	X		X	X	
Watchdog	X		X	X	X

Die bei Bedarf gesetzten Optionen *RestartPreventExitStatus=* und *RestartForceExitStatus=* ändern dieses Verhalten.

Beispiele

Einige selbst erstellte Service-Units finden sich auf unseren Handbuchseiten

[service-Unit für systemd Timer](#)

[service-Unit für systemd Path](#)

und mit der bevorzugten Suchmaschine im Internet.

[LinuxCommunity, Systemd-Units selbst erstellen](#)

1.4.3 Quellen systemd-service

[Deutsche Manpage, systemd.service](#)

[LinuxCommunity, Systemd-Units selbst erstellen](#)

Seite zuletzt aktualisiert 2021-04-07

% Systemd - target

ANFANG INFOBEREICH FÜR DIE AUTOREN

Dieser Bereich ist vor der Veröffentlichung zu entfernen !!!

Status: RC2

Änderungen 2021-02:

- Neu "Init-System - .target Ziel-Unit"
- Für die Verwendung mit pandoc optimiert.

ENDE INFOBEREICH FÜR DIE AUTOREN

1.5 systemd.target - Ziel-Unit (Runlevel)

Die grundlegenden und einführenden Informationen zu Systemd enthält die Handbuchseite [Systemd-Start](#)

Jetzt erklären wir die Funktion der Unit **systemd.target**, die den allgemein bekannten Runleveln ähneln, etwas ausführlicher.

Die verschiedenen Runlevel, in die gebootet oder gewechselt wird, beschreibt systemd als **Ziel-Unit**. Sie besitzen die Erweiterung **.target**.

Die alten sysvinit-Befehle werden weiterhin unterstützt. (Hierzu ein Zitat aus *man systemd*: "... wird aus Kompatibilitätsgründen und da es leichter zu tippen ist, bereitgestellt.")

Ziel-Unit	Beschreibung
emergency.target	Startet in eine Notfall-Shell auf der Hauptkonsole. Es ist die minimalste Version eines Systemstarts, um eine interaktive Shell zu erlangen. Mit dieser Unit kann der Bootvorgang Schritt für Schritt begleitet werden.
rescue.target	Startet das Basissystem (einschließlich Systemeinhängungen) und eine Notfall-Shell. Im Vergleich zu multi-user.target könnte dieses Ziel als single-user.target betrachtet werden.
multi-user.target	Mehrbenutzersystem mit funktionierendem Netzwerk, ohne Grafikserver X. Diese Unit wird verwendet, wenn man X stoppen bzw. nicht in X booten möchte. Auf dieser Unit wird eine Systemaktualisierung (dist-upgrade) durchgeführt .
graphical.target	Die Unit für den Mehrbenutzermodus mit Netzwerkfähigkeit und einem laufenden X-Window-System.
default.target	Die Vorgabe-Unit, die Systemd beim Systemstart startet. In siduction ist dies ein Symlink auf graphical.target (außer NoX).

Ein Blick in die Dokumentation "*man SYSTEMD.SPECIAL(7)*" ist obligatorisch um die Zusammenhänge der verschiedenen *.target-Unit* zu verstehen.

1.5.1 Besonderheiten

Bei den Ziel-Unit sind drei Besonderheiten zu beachten:

1. Die Verwendung auf der **Kernel-Befehlszeile** beim Bootvorgang.
Um im Bootmanager Grub in den Editiermodus zu gelangen, muss man beim Erscheinen der Bootauswahl die Taste `e` drücken. Anschließend hängt man an die Kernel-Befehlszeile das gewünschte Ziel mit der folgenden Syntax: `"systemd.unit=xxxxxxx.target"` an. Die Tabelle listet die Kernel-Befehle und ihre noch gültigen Entsprechungen auf.

Ziel-Unit	Kernel-Befehl	Kernel-Befehl alt
emergency.target	systemd.unit=emergency.target	-
rescue.target	systemd.unit=rescue.target	1
multi-user.target	systemd.unit=multi-user.target	3
graphical.target	systemd.unit=graphical.target	5

Die alten Runlevel 2 und 4 verweisen auf multi-user.target

2. Die Verwendung im **Terminal** während einer laufenden Sitzung. Vorausgesetzt man befindet sich in einer laufenden graphischen Sitzung, kann man mit der Tastenkombination **CTRL + ALT + F2** zum virtuellen Terminal tty2 wechseln. Hier meldet man sich als User **root** an. Die folgende Tabelle listet die **Terminal-Befehle** auf, wobei der Ausdruck *isolate* dafür sorgt, dass alle Dienste die die Ziel-Unit nicht anfordert, beendet werden.

Ziel-Unit	Terminal-Befehl	init-Befehl alt
emergency.target	systemctl isolate emergency.target	-
rescue.target	systemctl isolate rescue.target	init 1
multi-user.target	systemctl isolate multi-user.target	init 3
graphical.target	systemctl isolate graphical.target	init 5

3. Ziel-Unit, die **nicht direkt aufgerufen** werden sollen.
Eine ganze Reihe von Ziel-Unit sind dazu da während des Bootvorgangs oder des .target-Wechsels Zwischenschritte mit Abhängigkeiten zu gruppieren. Die folgende Liste zeigt drei häufig verwendete Kommandos die **nicht** mit der Syntax `"isolate xxxxxxx.target"` aufgerufen werden sollen.

Ziel	Terminal-Befehl	init-Befehl alt
halt	systemctl halt	-
poweroff	systemctl poweroff	init 0
reboot	systemctl reboot	init 6

halt, *poweroff* und *reboot* holen mehrere Unit in der richtigen Reihenfolge herein, um das System geordnet zu beenden und ggf. einen Neustart auszuführen.

1.5.2 Quellen systemd-target

[Manpage systemd.target, de](#)

Seite zuletzt aktualisiert 2021-02-14

% Systemd - mount

ANFANG INFOBEREICH FÜR DIE AUTOREN

Dieser Bereich ist vor der Veröffentlichung zu entfernen !!!

Status: RC2

Änderungen 2021-02 bis 03:

- Neu “systemd-mount Unit”
- Für die Verwendung mit pandoc optimiert.

Änderungen 2021-04

- Durch “systemd-unit-datei” erforderliche Anpassungen

ENDE INFOBEREICH FÜR DIE AUTOREN

1.6 systemd-mount

Die grundlegenden und einführenden Informationen zu Systemd enthält die Handbuchseite [Systemd-Start](#). Die alle Unit-Dateien betreffenden Sektionen *[Unit]* und *[Install]* behandelt unsere Handbuchseite [Systemd Unit-Datei](#).

In der vorliegenden Handbuchseite erklären wir die Funktion der systemd-Unit **.mount** und **.automount**. Mit ihnen verwaltet systemd Einhängpunkte für Laufwerke und deren Partitionen, die sowohl lokal als auch über das Netzwerk erreichbar sein können.

Die **.mount**-Unit ist eine Konfigurationsdatei, die für systemd Informationen über einen Einhängpunkt bereitstellt.

Die **.automount**-Unit überwacht das Dateisystem und aktiviert die gleichnamige **.mount**-Unit, wenn das darin bezeichnete Dateisystem verfügbar ist.

Für unmittelbar im PC verbaute Laufwerke und deren Partitionen verwenden wir nur die **.mount**-Unit. Sie wird aktiviert (enabled) und gestartet um die Laufwerke bei jedem Boot einzuhängen.

Bei Netzwerk-Dateisystemen bietet die **.mount**-Unit den Vorteil, Abhängigkeiten deklarieren zu können, damit die Unit erst aktiv wird, wenn das Netzwerk bereit steht. Auch hier benutzen wir nur die **.mount**-Unit und aktivieren und starten sie, um das Netzwerk-Dateisystemen bei jedem Boot einzuhängen. Die **.mount**-Unit unterstützt alle Arten von Netzwerk-Dateisystemen (NFS, SMB, FTP, WEBDAV, SFTP, SSH).

Entfernbare Geräte, wie USB-Sticks, und Netzwerk-Dateisysteme, die nicht permanent erreichbar sind, müssen immer an eine **.automount**-Unit gekoppelt werden. In diesem Fall darf die **.mount**-Unit nicht aktiviert werden und sollte auch keine *[Install]*-Sektion enthalten.

.mount- und **.automount**-Units müssen nach dem Einhängpunkt, den sie steuern, benannt sein. Beispiel: Der Einhängpunkt `/home/musteruser` muss in einer Unit-Datei `home-musteruser.mount`, bzw. `home-musteruser.automount`, konfiguriert werden.

Die in der `/etc/fstab` deklarierten Geräte und ihre Einhängpunkte übersetzt systemd in der frühen Bootphase mit Hilfe des `systemd-fstab-generators` in native **.mount**-Units.

1.6.1 Inhalt der mount-Unit

Die `.mount`-Unit verfügt über die folgenden Optionen in der zwingend erforderlichen [Mount]-Sektion:

- **What=** (Pflicht)
Enthält den absoluten Pfad des eingehängten Geräts, also z.B. Festplatten-Partitionen wie `/dev/sda8` oder eine Netzwerkfreigabe wie NFSv4 oder Samba.
- **Where=** (Pflicht)
Hier wird der Einhängepunkt (mount point) festgelegt, d.h. der Ordner, in den die Partition, das Netzlaufwerk oder Gerät eingehängt werden soll. Falls dieser nicht existiert, wird er beim Einhängen erzeugt.
- **Type=** (optional)
Hier wird der Typ des Dateisystems angegeben, gemäß dem mount-Parameter `-t`.
- **Options=** (optional)
Enthält alle verwendeten Optionen in einer Komma getrennten Liste, gemäß dem mount-Parameter `-o`.
- **LazyUnmount=** (Standard: off)
Wenn der Wert auf `true` gesetzt wird, wird das Dateisystem wieder ausgehängt, sobald es nicht mehr benötigt wird.
- **SloppyOptions=** (Standard: off)
Falls `true`, erfolgt eine entspannte Auswertung der in `Options=` festgelegten Optionen und unbekannte Einhängeoptionen werden toleriert. Dies entspricht dem mount-Parameter `-s`.
- **ReadWriteOnly=** (Standard: off)
Falls `false`, wird bei dem Dateisystem oder Gerät, das read-write eingehängt werden soll, das Einhängen aber scheitert, versucht es read-only einzuhängen. Falls `true`, endet der Prozess sofort mit einem Fehler, wenn die Einhängung read-write scheitert. Dies entspricht dem mount-Parameter `-w`.
- **ForceUnmount=** (Standard: off)
Falls `true`, wird das Aushängen erzwungen wenn z. B. ein NFS-Dateisystem nicht erreichbar ist. Dies entspricht dem mount-Parameter `-f`.
- **DirectoryMode=** (Standard: 0755)

Die, falls notwendig, automatisch erzeugten Verzeichnisse von Einhängepunkten, erhalten den deklarierten Dateisystemzugriffsmodus. Akzeptiert einen Zugriffsmodus in oktaler Notation.

- **TimeoutSec=** (Vorgabewert aus der Option *DefaultTimeoutStartSec=* in *systemd-system.conf*)
Konfiguriert die Zeit, die auf das Beenden des Einhängebefehls gewartet wird. Falls ein Befehl sich nicht innerhalb der konfigurierten Zeit beendet, wird die Einhängung als fehlgeschlagen betrachtet und wieder heruntergefahren. Akzeptiert einen einheitenfreien Wert in Sekunden oder einen Zeitdauerwert wie »5min 20s«. Durch Übergabe von »0« wird die Zeitüberschreitungslogik deaktiviert.

1.6.2 Inhalt der automount-Unit

Die *.automount*-Unit verfügt über die folgenden Optionen in der zwingend erforderlichen [Automount]-Sektion:

- **Where=** (Pflicht)
Hier wird der Einhängepunkt (mount point) festgelegt, d.h. der Ordner, in den die Partition, das Netzlaufwerk oder Gerät eingehängt werden soll. Falls dieser nicht existiert, wird er beim Einhängen erzeugt.
- **DirectoryMode=** (Standard: 0755)
Die, falls notwendig, automatisch erzeugten Verzeichnisse von Einhängepunkten erhalten den deklarierten Dateisystemzugriffsmodus. Akzeptiert einen Zugriffsmodus in oktaler Notation.
- **TimeoutIdleSec=** (Standard: 0)
Bestimmt die Zeit der Inaktivität, nach der systemd versucht das Dateisystem auszuhängen. Akzeptiert einen einheitenfreien Wert in Sekunden oder einen Zeitdauerwert wie »5min 20s«. Der Wert "0" deaktiviert die Option.

1.6.3 Beispiele

Systemd liest den Einhängepunkt aus dem Namen der *.mount*- und *.automount*-Units. Deshalb müssen sie nach dem Einhängepunkt, den sie steuern, benannt sein.

Dabei ist zu beachten, keine Bindestriche "-" in den Dateinamen zu verwenden,

denn sie deklarieren ein neues Unterverzeichnis im Verzeichnisbaum. Einige Beispiele:

- unzulässig: /data/home-backup
- zulässig: /data/home_backup
- zulässig: /data/home\x2dbackup

Um einen fehlerfreien Dateinamen für die *.mount*- und *.automount*-Unit zu erhalten, verwenden wir im Terminal den Befehl “systemd-escape”.

```
$ systemd-escape -p --suffix=mount "/data/home-backup"  
data/home\x2dbackup.mount
```

Festplatten-Partition

Eine Partition soll nach jedem Systemstart unter “/disks/TEST” erreichbar sein. Wir erstellen mit einem Texteditor die Datei “disks-TEST.mount” im Verzeichnis “/usr/local/lib/systemd/system/”. (Ggf. ist das Verzeichnis zuvor mit dem Befehl `mkdir -p /usr/local/lib/systemd/system/` anzulegen.)

```
[Unit]  
Description=Mount /dev/sdb7 at /disks/TEST  
After=blockdev@dev-disk-by\x2duuid-a7af4b19\x2df29d\x2d43bc\x2d3b12\x2d87924fc3d8c7.target  
Requires=local-fs.target  
Wants=multi-user.target  
  
[Mount]  
Where=/disks/TEST  
What=/dev/disk/by-uuid/a7af4b19-f29d-43bc-3b12-87924fc3d8c7  
Type=ext4  
Options=defaults,noatime  
  
[Install]  
WantedBy=multi-user.target
```

Anschließend aktivieren und starten wir die neue *.mount*-Unit.

```
# systemctl enable --now disks-TEST.mount
```

NFS

Das “document-root”-Verzeichnis eines Apache Webserver im heimischen Netzwerk soll in das Home-Verzeichnis des Arbeitsplatz-Rechners mittels NFS eingehängt werden.

Wir erstellen mit einem Texteditor die Datei “home-<user>-www_data.mount” im Verzeichnis “/usr/local/lib/systemd/system/”.

“<user>” bitte mit dem eigenen Namen ersetzen.

```
[Unit]
Description=Mount server1/var/www/ using NFS
After=network-online.target
Wants=network-online.target

[Mount]
What=192.168.3.1:/
Where=/home/<user>/www_data
Type=nfs
Options=nfsvers=4,rw,users,soft
ForceUnmount=true
```

Diese Datei enthält keine [Install]-Sektion und wird auch nicht aktiviert. Die Steuerung übernimmt die nun folgende Datei “home-<user>-www_data.automount” im gleichen Verzeichnis.

```
[Unit]
Description=Automount server1/var/www/ using NFS
ConditionPathExists=/home/<user>/www_data
Requires=NetworkManager.service
After=network-online.target
Wants=network-online.target

[Automount]
Where=/home/<user>/www_data
TimeoutIdleSec=60

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

Anschließend:

```
# systemctl enable --now home-<user>-www_data.automount
```

Jetzt wird das “document-root”-Verzeichnis des Apache Webserver eingehangen, sobald wir in das Verzeichnis “/home/<user>/www_data” wechseln.

Die Statusabfrage bestätigt die Aktion.

```
# systemctl status home-<user>-www_data.mount --no-pager●
```

```

home-<user>-www_data.mount - Mount server1/var/www/ using NFS
  Loaded: loaded (/usr/local/lib/systemd/system/home-<user>-www_data.mount; disabled; vendor preset: enabled)
  Active: active (mounted) since Wed 2021-03-10 16:27:58 CET; 8 min ago
TriggeredBy: ● home-<user>-www_data.automount
  Where: /home/<user>/www_data
  What: 192.168.3.1:/
  Tasks: 0 (limit: 4279)
  Memory: 120.0K
  CPU: 5ms
  CGroup: /system.slice/home-<user>-www_data.mount
[...]

# systemctl status home-<user>-www_data.automount --no-pager●
home-<user>-www_data.automount - Automount server1/var/www/ using NFS
  Loaded: loaded (/usr/local/lib/systemd/system/home-<user>-www_data.automount; enabled; vendor preset: enabled)
  Active: active (running) since Wed 2021-03-10 16:27:58 CET; 8 min ago
  Triggers: ● home-<user>-www_data.mount
  Where: /home/<user>/www_data
[...]
```

Der Journalauszug protokolliert anschaulich die Funktion von “TimeoutIdleSec=60” zum Aushängen des Dateisystems und das wieder Einhängen durch den Start des Dateimanagers Thunar sowie einen Aufruf von “/home/<user>/www_data” im Terminal.

```

# journalctl -f -u home-<user>-www_data.*
[...]
Mär 10 17:56:14 pc1 systemd[1]: Mounted Mount server1/var/www/ using NFS.
Mär 10 17:57:34 pc1 systemd[1]: Unmounting Mount server1/var/www/ using NFS...
Mär 10 17:57:35 pc1 systemd[1]: home-<user>-www_data.mount: Succeeded.
Mär 10 17:57:35 pc1 systemd[1]: Unmounted Mount server1/var/www/ using NFS.
Mär 10 17:58:14 pc1 systemd[1]: home-<user>-www_data.automount: Got automount request for /home/<user>/www_data, triggered by 2500 (Thunar)
Mär 10 17:58:14 pc1 systemd[1]: Mounting Mount server1/var/www/ using NFS...
```



```
Mär 10 17:58:14 pc1 systemd[1]: Mounted Mount server1/var/www/ ↵  
using NFS.  
Mär 10 18:00:15 pc1 systemd[1]: Unmounting Mount server1/var/www/ ↵  
using NFS...  
Mär 10 18:00:15 pc1 systemd[1]: home-<user>-www_data.mount: ↵  
Succeeded.  
Mär 10 18:00:15 pc1 systemd[1]: Unmounted Mount server1/var/www/ ↵  
using NFS.  
Mär 10 18:00:30 pc1 systemd[1]: home-<user>-www_data.automount: Got ↵  
automount request for /home/<user>/www_data, triggered by 6582 ↵  
(bash)  
Mär 10 18:00:30 pc1 systemd[1]: Mounting Mount server1/var/www/ ↵  
using NFS...  
Mär 10 18:00:30 pc1 systemd[1]: Mounted Mount server1/var/www/ ↵  
using NFS.  
Mär 10 18:01:51 pc1 systemd[1]: Unmounting Mount server1/var/www/ ↵  
using NFS...  
Mär 10 18:01:51 pc1 systemd[1]: home-<user>-www_data.mount: ↵  
Succeeded.  
Mär 10 18:01:51 pc1 systemd[1]: Unmounted Mount server1/var/www/ ↵  
using NFS.  
[...]
```

Weitere Beispiele

Im Internet finden sich mit Hilfe der favorisierten Suchmaschine vielerlei Beispiele für die Anwendung der *.mount*- und *.automount*-Unit. Das Kapitel "Quellen" enthält einige Webseiten mit eine ganze Reihe weiterer Beispiele. Dringend empfohlen sind auch die man-Pages.

1.6.4 Quellen sytemd-mount

[Deutsche Manpage, systemd.mount](#)

[Deutsche Manpage, mount](#)

[Manjaro Forum, systemd.mount](#)

[Manjaro Forum, Use systemd to mount ANY device](#)

[Linuxnews, nfs per systemd](#)

[Debianforum, Netzlaufwerke einbinden](#)

[Ubuntuusers, Mount-Units](#)

Seite zuletzt aktualisiert 2021-04-06

% Systemd - timer

ANFANG INFOBEREICH FÜR DIE AUTOREN

Dieser Bereich ist vor der Veröffentlichung zu entfernen !!!

Status: RC3

Änderungen 2021-02:

- Neu “systemd-timer Unit”
- Für die Verwendung mit pandoc optimiert.

Änderungen 2021-02 + Review

Änderungen 2021-03 + Minuten -> Sekunden + geringe Layout Änderung

Änderungen 2021-04

- Durch “systemd-unit-datei” erforderliche Anpassungen

ENDE INFOBEREICH FÜR DIE AUTOREN

1.7 systemd-timer

Die grundlegenden und einführenden Informationen zu Systemd enthält die Handbuchseite [Systemd-Start](#). Die alle Unit-Dateien betreffenden Sektionen *[Unit]* und *[Install]* behandelt unsere Handbuchseite [Systemd Unit-Datei](#). In der vorliegenden Handbuchseite erklären wir die Funktion der Unit **systemd.timer**, mit der zeitgesteuert Aktionen ausgelöst werden können.

Die *“.timer”*-Unit wird meist eingesetzt, um regelmäßig anfallende Aktionen zu erledigen. Dazu ist eine gleichnamige *“.service”*-Unit notwendig, in der die Aktionen definiert sind. Sobald der Systemzeitgeber mit der in der *“.timer”*-Unit definierten Zeit übereinstimmt, aktiviert die *“.timer”*-Unit die gleichnamige *“.service”*-Unit. Bei entsprechender Konfiguration können verpasste Läufe, während die Maschine ausgeschaltet war, nachgeholt werden. Auch ist es möglich, dass eine *“.timer”*-Unit die gewünschten Aktionen nur ein einziges Mal zu einem vorher definierten Termin auslöst.

1.7.1 Benötigte Dateien

Die **systemd-timer**-Unit benötigt zwei Dateien mit dem gleichen Basename im Verzeichnis `/usr/local/lib/systemd/system/` für ihre Funktion. (Ggf. ist das Verzeichnis zuvor mit dem Befehl `mkdir -p /usr/local/lib/systemd/system/` anzulegen.) Das sind die

- Timer-Unit-Datei (xxxxx.timer), welche die Zeitsteuerung und den Auslöser für die Service-Unit enthält und die
- Service-Unit-Datei (xxxxx.service), welche die zu startende Aktion enthält.

Für umfangreichere Aktionen erstellt man als dritte Datei ein Skript in `/usr/local/bin/`, das von der Service-Unit ausgeführt wird.

Wir erstellen in dem Beispiel ein regelmäßiges Backup mit *rsync*.

1.7.2 .timer-Unit anlegen

Wir legen die Datei **backup.timer** im Verzeichnis `/usr/local/lib/systemd/system/` mit folgendem Inhalt an.

```
[Unit]
```

```
Description="Backup my home directory"
```

```
[Timer]
```

```
OnCalendar=*-*-* 19:00:00
```

```
Persistent=true
```

```
[Install]
```

```
WantedBy=timers.target
```

Erklärungen

Die *.timer-Unit* muss zwingend die Sektion *[Timer]* enthalten, in der festgelegt wird wann und wie die zugehörige *.service-Unit* ausgelöst wird.

Es stehen zwei Timer-Typen zur Verfügung:

1. Realtime timers,
die mit der Option *onCalendar=* einen Echtzeit- (d.h. Wanduhr-)Zeitgeber definiert
(das Beispiel "*OnCalendar=*-*-* 19:00:00*" bedeutet "täglich um 19:00 Uhr"),
und
2. Monotonic timers,
die mit den Optionen *onActiveSec=*, *onBootSec=*, *onStartupSec=*, *onUnitActiveSec=*, *onUnitInactiveSec=* einen zu der Option relativen Zeitgeber definiert.
"*onBootSec=90*" bedeutet "90 Sekunden nach dem Booten" und
"*onUnitActiveSec=1d*" bedeutet "Einen Tag nachdem der Zeitgeber letztmalig aktiviert wurde".
Beide Optionen zusammen lösen die zugehörige *.service-Unit* 90 Sekunden nach den Booten und dann genau im 24 Stunden-Takt aus, solange die Maschine nicht heruntergefahren wird.

Die im Beispiel enthaltene Option "*Persistent=*" speichert den Zeitpunkt, zu dem die *.service-Unit* das letzte Mal ausgelöst wurde, als leere Datei im Verzeichnis */var/lib/systemd/timers/*. Dies ist nützlich, um verpasste Läufe, als die Maschine ausgeschaltet war, nachzuholen.

1.7.3 .service-Unit anlegen

Die *.service-Unit* wird von der *.timer-Unit* aktiviert und kontrolliert und benötigt daher keine *[Install]* Sektion. Somit reicht die Beschreibung der Unit in der Sektion

[Unit] und in der Sektion [Service] der auszuführende Befehl nach der Option *ExecStart=* aus.

Wir legen die Datei **backup.service** im Verzeichnis */usr/local/lib/systemd/system/* mit folgendem Inhalt an.

```
[Unit]
Description="Command to backup my home directory"

[Service]
Type=oneshot
ExecStart=/usr/bin/rsync -a --exclude=.cache/* /home/<user> /mnt/↵
      sdb5/backup/home/
```

Den String *<user>* bitte durch den eigenen User ersetzen.

1.7.4 .timer-Unit eingliedern

Mit dem folgenden Befehl gliedern wir die *.timer-Unit* in systemd ein.

```
# systemctl enable backup.timer
Created symlink /etc/systemd/system/timers.target.wants/backup.↵
      timer ↵
      /usr/local/lib/systemd/system/backup.timer.
```

Der analoge Befehl für die *.service-Unit* ist nicht notwendig und würde auch zu einem Fehler führen, da in ihr keine *[Install]* Sektion enthalten ist.

1.7.5 .timer-Unit manuell auslösen

Es wird nicht die *.timer-Unit*, sondern die von ihr auszulösende *.service-Unit* aufgerufen.

```
# systemctl start backup.service
```

1.7.6 .timer-Unit als cron Ersatz

“*cron*” und “*anacron*” sind die bekanntesten und weit verbreiteten Job-Zeitplaner. Systemd Timer können eine Alternative sein. Wir betrachten kurz den Nutzen von, und die Vorbehalte gegen Systemd Timer.

Nutzen

- Jobs können Abhängigkeiten haben (von anderen Systemd-Diensten abhängen).
- Timer Units werden im Systemd-Journal geloggt.
- Man kann einen Job sehr einfach unabhängig von seinem Timer aufrufen.
- Man kann Timer Units einen Nice-Wert geben oder cgroups für die Ressourcenverwaltung nutzen.
- Systemd Timer Units können von Ereignissen wie dem Booten oder Hardware-Änderungen ausgelöst werden.
- Sie können auf einfache Weise mit systemctl aktiviert oder deaktiviert werden.

Vorbehalte

- Die Konfiguration eines Cron-Jobs ist ein einfacher Vorgang.
- Cron kann E-Mails mit Hilfe der MAILTO-Variablen senden.

1.7.7 Quellen systemd-timer

[Deutsche Manpage 'systemd.timer'](#)

[Archlinux Wiki, Timers](#)

[PRO-LINUX.DE, Systemd Timer Units...](#)

Seite zuletzt aktualisiert 2021-04-06

% Systemd - path

ANFANG INFOBEREICH FÜR DIE AUTOREN

Dieser Bereich ist vor der Veröffentlichung zu entfernen !!!

Status: RC2

Änderungen 2021-02:

- Neu "systemd-path Unit"
- Für die Verwendung mit pandoc optimiert.

Änderungen 2021-04:

- Durch "systemd-unit-datei" erforderliche Anpassungen

ENDE INFOBEREICH FÜR DIE AUTOREN

1.8 systemd-path

Die grundlegenden und einführenden Informationen zu Systemd enthält die Handbuchseite [Systemd-Start](#). Die alle Unit-Dateien betreffenden Sektionen *[Unit]* und *[Install]* behandelt unsere Handbuchseite [Systemd Unit-Datei](#).

In der vorliegenden Handbuchseite erklären wir die Funktion der Unit **systemd.path**, mit der systemd Pfade überwacht und Pfad-basierte Aktionen auslöst.

Die *„.path-Unit“* ermöglicht es, bei Änderungen an Dateien und Verzeichnissen (Pfaden) eine Aktion auszulösen.

Sobald ein Ereignis eintritt, kann Systemd einen Befehl oder ein Skript über eine Service Unit ausführen. Die *„.path-Unit“* ist nicht in der Lage Verzeichnisse rekursiv zu überwachen. Es können aber mehrere Verzeichnisse und Dateien angegeben werden.

Die Pfad-spezifischen Optionen werden in dem Abschnitt *[Path]* konfiguriert.

1.8.1 Benötigte Dateien

Die **systemd-path**-Unit benötigt für ihre Funktion mindestens zwei Dateien mit vorzugsweise dem gleichen Namen, aber unterschiedlicher Namensendung, im Verzeichnis `/usr/local/lib/systemd/system/`. (Ggf. ist das Verzeichnis zuvor mit dem Befehl `mkdir -p /usr/local/lib/systemd/system/` anzulegen.) Das sind die

- Path-Unit-Datei (`<name>.path`), welche die Überwachung und den Auslöser für die Service-Unit enthält und die
- Service-Unit-Datei (`<name>.service`), welche die zu startende Aktion enthält. Für umfangreichere Aktionen erstellt man zusätzlich ein Skript in `/usr/local/bin/`, das von der Service-Unit ausgeführt wird.

1.8.2 .path-Unit Optionen

Die *.path-Unit* muss zwingend die Sektion *[Path]* enthalten, in der festgelegt wird wie und was zu überwachen ist.

Die speziellen Optionen sind:

- PathExists=

prüft, ob der betreffende Pfad existiert. Wenn es zutrifft, wird die zugehörige Unit aktiviert.

- **PathExistsGlob=**
Wie oben, unterstützt Datei-Glob-Ausdrücke.
- **PathChanged=**
beobachtet eine Datei oder einen Pfad und aktiviert die zugehörige Unit, wenn Änderungen auftreten.
Aktionsauslösende Änderungen sind:
 - Erstellen und Löschen von Dateien.
 - Attribute, Rechte, Eigentümer.
 - Schließen der zu beobachtenden Datei nach Schreibzugriff und schließen irgendeiner Datei nach Schreibzugriff bei Beobachtung des Pfades.
- **PathModified=**
wie zuvor, aber zusätzlich wird die zugehörige Unit bei einfachen Schreibzugriffen aktiviert, auch wenn die Datei nicht geschlossen wird.
- **DirectoryNotEmpty=**
aktiviert die zugehörige Unit wenn das Verzeichnis nicht leer ist.
- **Unit=**
die zu aktivierende, zugehörige Unit. Zu beachten ist auch, dass die *.path-Unit* standardmäßig die *„.service-Unit“* mit dem gleichen Name aktiviert. Nur bei Abweichungen hiervon ist die Option *Unit=* innerhalb der Sektion *[Path]* notwendig.
- **MakeDirectory=**
das zu beobachtenden Verzeichnis wird vor der Beobachtung erstellt.
- **DirectoryMode=**
legt bei Verwendung, für das zuvor erstellte Verzeichnis, den Zugriffsmodus in oktaler Notation fest. Standardmäßig 0755.

Ein Beispiel

Auf der Konfiguration des Apache-Webserver entsprechend unserer Handbuchseite [LAMP - Apache, Benutzer und Rechte](#) basierend, wollen wir das Zusammenspiel der *.path-Unit* mit anderen *systemd-Unit* verdeutlichen.

Die Abbildung *path-Unit-Funktion* stellt die Abhängigkeiten der systemd-Units unseres Beispiels dar.

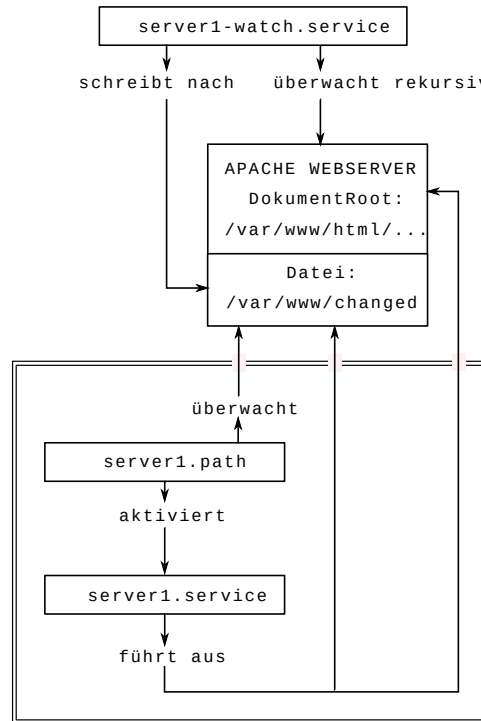


Abbildung 1: path-Unit Funktion

Der doppelt umrandete Teil in der Graphik verdeutlicht die Kernfunktion der *.path-Unit*. Die *server1.path*-Unit überwacht die Datei “/var/www/changed” und aktiviert bei Änderungen die zugehörige *server1.service*-Unit. Diese wiederum führt dann die gewünschten Aktionen im Verzeichnis “/var/www/html/” aus und stellt die Datei “/var/www/changed” zurück.

Die außerhalb der Umrandung liegende “*server1-watch.service*”-Unit übernimmt die rekursive Überwachung von *DocumentRoot* des Apache-Webservers.

1.8.3 .path-Unit anlegen

Wir legen die Datei *server1.path* im Verzeichnis */usr/local/lib/systemd/system/*, die die Datei */var/www/changed* auf Änderungen überwacht, mit folgendem Inhalt an:

```
[Unit]
Description=Monitoring "changed" file!
BindsTo=server1-watch.service
After=server1-watch.service

[Path]
PathModified=/var/www/changed

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

Erklärungen

Sektion [Unit]:

Die Option "*BindsTo*=" stellt die stärkste verfügbare Bindung zweier systemd-Einheiten aneinander dar. Falls eine von ihnen während des Starts oder des Betriebs in einen Fehlerzustand übergeht, wird die andere auch unmittelbar beendet.

Zusammen mit der Option "*After*=" wird erreicht, dass die *server1.path*-Unit erst startet, nachdem die *server1-watch.service*-Unit ihren erfolgreichen Start an systemd zurückmeldet.

Sektion [Path]:

"*PathModified*=" ist die richtige Wahl. Die Option reagiert auf Änderungen in der Datei */var/www/changed*, selbst wenn die Datei nicht geschlossen wird.

Die Option "*PathModified*=" (oder andere, siehe oben) kann mehrfach angegeben werden.

1.8.4 .service-Unit anlegen

Die *server1.service*-Unit wird von der *server1.path*-Unit aktiviert und kontrolliert und benötigt daher keine *[Install]* Sektion. Somit reichen die Beschreibung der Unit in der Sektion *[Unit]*, und in der Sektion *[Service]* die auszuführenden Befehle, aus.

Wir legen die Datei *server1.service* im Verzeichnis */usr/local/lib/systemd/system/* mit folgendem Inhalt an.

```
[Unit]
Description=Change permissions in server1 folder

[Service]
Type=oneshot
ExecStartPre=/usr/bin/truncate -s 0 /var/www/changed
```

```
ExecStart=/usr/bin/chown -R www-data /var/www/html/  
ExecStart=/usr/bin/chmod -R g+w /var/www/html/  
ExecStart=/usr/bin/chmod -R o-r /var/www/html/
```

Erklärungen

Sektion [Service]:

“*ExecStart=*”-Befehle werden nur ausgeführt, nachdem sich alle “*ExecStartPre=*”-Befehle erfolgreich beendet haben. Zuerst wird die Datei */var/www/changed* auf 0-Bite zurückgesetzt und danach der Rest ausgeführt.

1.8.4.1 Zusätzliche .service-Unit anlegen Da die *.path-Unit* Verzeichnisse nicht rekursiv überwachen kann, benötigen wir für unser Beispiel eine zusätzliche *.service-Unit*. Wir legen die Datei *server1-watch.service* im Verzeichnis */usr/local/lib/systemd/system/* mit folgendem Inhalt an.

```
[Unit]  
Description=Watching server1 folder.  
Before=server1.path  
Wants=server1.path  
  
[Service]  
Type=forking  
ExecStart=inotifywait -dqr -e move,create -o /var/www/changed /var/www/html/  
  
[Install]  
WantedBy=multi-user.target
```

Anmerkung:

Interessant ist, dass systemd intern das *inotify-API* für *.path-Unit* verwendet, um Dateisysteme zu überwachen, jedoch deren Rekursiv-Funktion nicht implementiert.

Erklärungen

Die Sektion [Unit]:

“*Before=*” und “*Wants=*” sind die entsprechenden Korrelationen zu “*BindsTo=*” und “*After=*” aus der *server1.service-Unit*.

Sektion [Service]:

inotifywait protokolliert in die Datei */var/www/changed*, die außerhalb von *DocumentRoot* des Apache-Webservers liegt.

1.8.5 .path-Unit eingliedern

Auf Grund der Abhängigkeit gliedern wir zuerst die *server1.path-Unit* und dann die *server1-watch.service-Unit* in systemd ein. Die *server1.service-Unit* benötigt und beinhaltet keine [Install]-Sektion. Bei dem Versuch sie einzugliedern erhielten wir eine Fehlermeldung.

```
# systemctl enable server1.path
Created symlink /etc/systemd/system/multi-user.target.wants/server1-
.path /usr/local/lib/systemd/system/server1.path.

# systemctl enable server1-watch.service
Created symlink /etc/systemd/system/multi-user.target.wants/server1-
-watch.service /usr/local/lib/systemd/system/server1-watch-
service.
```

Nun ist das Monitoring auch gleich aktiv, wie uns die Statusausgaben aller drei Units zeigen.

```
# systemctl status server1-watch.service●
server1-watch.service - Watching server1 folder.
  Loaded: loaded (/usr/local/lib/systemd/system/server1-watch-
service; enabled; vendor preset: enabled)
  Active: active (running) since Sun 2021-02-21 19:25:20 CET; 1
min 49s ago
  Process: 23788 ExecStart=inotifywait -dqr -e move,create -o /
var/www/changed /var/www/html/ (code=exited, status=0/
SUCCESS)
  Main PID: 23790 (inotifywait)
  Tasks: 1 (limit: 2322)
  Memory: 216.0K
  CPU: 5ms
  CGroup: /system.slice/server1-watch.service└─
23790 inotifywait -dqr -e move,create -o /var/www/
changed /var/www/html/

Feb 21 19:25:20 lap1 systemd[1]: Starting Watching server1 folder
....
Feb 21 19:25:20 lap1 systemd[1]: Started Watching server1 folder..

# systemctl status server1.path●
server1.path - Monitoring "changed" file!
  Loaded: loaded (/usr/local/lib/systemd/system/server1.path;
enabled; vendor preset: enabled)
  Active: active (waiting) since Sun 2021-02-21 19:25:20 CET; 3
```

```
min 27s ago
Triggers: ● server1.service

Feb 21 19:25:20 lap1 systemd[1]: Started Monitoring "changed" file
!

# systemctl status server1.service●
server1.service - Change permissions in server1 folder
  Loaded: loaded (/usr/local/lib/systemd/system/server1.service;
         static)
  Active: inactive (dead)
  TriggeredBy: ● server1.path
```

Der Status “Active: inactive (dead)” der letzten Ausgabe ist der normale Zustand für die Unit *server1.service*, denn diese Unit ist nur dann aktiv, wenn sie von *server1.path* angestoßen wurde ihre Befehlskette auszuführen. Danach geht sie wieder in den inaktiven Zustand über.

1.8.6 service-Unit manuell ausführen

Sollte es einmal hilfreich oder nötig sein die Dateirechte in *DocumentRoot* des Apache-Webserver manuell zu ändern, setzen wir einfach diesen Befehl ab:

```
# systemctl start server1.service
```

Eine erneute Statusabfrage generiert zusätzlich einige Protokollzeilen, denen wir den erfolgreichen Durchlauf der Befehlskette entnehmen können.

```
# systemctl status server1.service●
server1.service - Change permissions in server1 folder
  Loaded: loaded (/usr/local/lib/systemd/system/server1.service;
         static)
  Active: inactive (dead) since Mon 2021-02-22 17:55:36 CET; 1
         min 43s ago
  TriggeredBy: ● server1.path
    Process: 2822 ExecStartPre=truncate -s 0 /var/www/changed (code
              =exited, status=0/SUCCESS)
    Process: 2823 ExecStart=chown -R www-data /var/www/html1/ (code
              =exited, status=0/SUCCESS)
    Process: 2824 ExecStart=chmod -R g+w /var/www/html1/ (code=
              exited, status=0/SUCCESS)
    Process: 2825 ExecStart=chmod -R o-r /var/www/html1/ (code=
              exited, status=0/SUCCESS)
```

```
Main PID: 2825 (code=exited, status=0/SUCCESS)
CPU: 19ms

Feb 22 17:55:36 lap1 systemd[1]: Starting Change permissions in ↵
server1 folder...
Feb 22 17:55:36 lap1 systemd[1]: server1.service: Succeeded.
Feb 22 17:55:36 lap1 systemd[1]: Finished Change permissions in ↵
server1 folder.
```

1.8.7 Quellen systemd-path

[Deutsche Manpage 'systemd.path'](#)

Ein anders gelagertes Beispiel:

[PRO-LINUX.DE, Systemd Path Units...](#)

Seite zuletzt aktualisiert 2021-04-06

% Systemjournal

ANFANG INFOBEREICH FÜR DIE AUTOREN

Dieser Bereich ist vor der Veröffentlichung zu entfernen !!!

Status: RC2

Änderungen 2021-02 bis 03:

- Datei neu angelegt und Journald aus systemd-start_de.md)entfernt.
- Für die Verwendung mit pandoc optimiert.

ENDE INFOBEREICH FÜR DIE AUTOREN

1.9 Systemjournal

Das Systemjournal besteht aus dem *systemd-journald*, kurz **journald**, der Protokollmeldungen sammelt und speichert, und dem **journalctl**, das der Verwaltung, Abfrage und Ausgabe der gesammelten Protokollmeldungen dient.

1.9.1 journald

journald ist ein Systemdienst, der mit Hilfe der Unit *systemd-journald.service* (und seiner zugehörigen Socket-Units) Protokollmeldungen sammelt und speichert. Es erstellt und verwaltet strukturierte, indizierte Journale, basierend auf den Protokollmeldungen aus:

- Kernel-Protokollmeldungen
- Einfache Systemprotokollmeldungen
- Strukturierte Systemprotokollmeldungen über die native Journal-API
- Standardausgabe und Standardfehlerausgabe der Dienste-Units
- Audit-Aufzeichnungen, stammend aus dem Kernel-Audit-Subsystem

journald erlaubt Journal-“Namensräume”. Sie sind zum Einen ein Mechanismus zur logischen Isolation eines Protokolldatenstroms vom Rest des Systems, zum Anderen auch ein Mechanismus zur Steigerung der Leistung. Journal-Namensräume existieren gleichzeitig und nebeneinander. Jeder hat seinen eigenen, unabhängigen Protokolldatenstrom. Nach der Installation von siduction besteht nur der Vorgabe-Namensraum des Systems.

Der *journald* speichert die Protokolldaten standardmäßig dauerhaft unter */var/log/journal/MASCHINENKENNUNG*.

Protokolldaten für andere Namensräume befinden sich in */var/log/journal/MASCHINENKENNUNG.NAMENSRAUM*.

Der Befehl **systemd-cat** bietet zwei Möglichkeiten um unabhängig von systemd-Units Daten eines Prozesses an das Journal weiterzugeben.

1. **systemd-cat** <Programm> <Option(en)>

Mit einem Programmaufruf oder Befehl verwendet, leitet *systemd-cat* alle Standardeingaben, Standardausgaben und Standardfehlerausgaben eines Prozesses zum Journal um.

2. In einer Pipe verwendet,

dient *systemd-cat* als Filterwerkzeug, um die zuvor erstellte Ausgabe an das Journal zu senden.

Falls kein Parameter übergeben wurde, wird *systemd-cat* alles, was es von der Standardeingabe liest, an das Journal schicken. Die man-page [systemd-cat.1.de](#) bietet weitere Informationen.

1.9.2 journald über das Netzwerk

Die *systemd-journal*-Module *upload*, *remote* und *gatewayd* ermöglichen das Versenden und Empfangen von Systemprotokolldaten zwischen verschiedenen Rechnern über das Netzwerk. Mit ihrer Hilfe lassen sich entfernte Rechner fortlaufend überwachen. In dieser Konstellation ist es sinnvoll auf dem Remoterechner Namensräume für die Protokolldaten der entfernten Rechner einzurichten.

Für weitere Informationen bitte die man-pages [journal upload](#), [journal remote](#) und [journal gatewayd](#) lesen.

1.9.3 journald.conf

Die folgenden Dateien konfigurieren verschiedene Parameter des Systemd-Journal-Dienstes.

- */etc/systemd/journald.conf*
- */etc/systemd/journald.conf.d/*.conf*
- */etc/systemd/journald@NAMENSRAUM.conf* (optional)
- */run/systemd/journald.conf.d/*.conf* (optional)
- */usr/lib/systemd/journald.conf.d/*.conf* (optional)

Der Vorgabe-Namensraum, den der *systemd-journald.service* (und seine zugehörigen Socket-Units) verwaltet, wird in */etc/systemd/journald.conf* und zugeordneten Ergänzungen konfiguriert.

Die Konfigurationsdatei enthält die Vorgaben als auskommentierten Hinweis für den Administrator. Um lokal Einstellungen zu ändern, genügt es diese Datei zu bearbeiten.

Instanzen, die andere Namensräume verwalten, werden nur benötigt, wenn von den Vorgaben **abgewichen werden muss**. Deren Konfigurationsdatei ist nach dem Muster `“etc/systemd/journald@NAMENSRAUM.conf”` zu erstellen.

Einem bestimmten Journal-Namensraum können Dienste-Units mittels der Unit-Dateieinstellung `“LogNamespace=”` zugeordnet werden.

Standardmäßig sammelt nur der Vorgabe-Namensraum Kernel- und Auditprotokollnachrichten.

1.9.4 Rangfolge

Wenn Pakete die Konfiguration anpassen müssen, können sie Konfigurationsschnipsel in `/usr/lib/systemd/*.conf.d/` oder `/usr/local/lib/systemd/*.conf.d/` installieren.

Die Hauptkonfigurationsdatei wird vor jeder anderen aus den Konfigurationsverzeichnissen gelesen und hat die niedrigste Priorität. Einträge in einer Datei in jedem der Konfigurationsverzeichnisse setzen Einträge in der Hauptkonfigurationsdatei außer Kraft. Dateien in den Unterverzeichnissen `*.conf.d/` werden nach ihrem Dateinamen sortiert, unabhängig davon, in welchem Unterverzeichnis sie sich befinden. Sofern eigene Konfigurationsdateien nötig sind, wird empfohlen, allen Dateinamen in diesen Unterverzeichnissen eine zweistellige Zahl und einen Bindestrich voranzustellen, um die Sortierung der Dateien zu vereinfachen.

1.10 journalctl

journalctl dient der Abfrage des von systemd-journald erstellten Journals.

Beim Aufruf ohne Parameter wird der gesamte Inhalt aus allen zugreifbaren Quellen des Journals angezeigt, beginnend mit dem ältesten Eintrag. Die für die Ausgabe herangezogenen Journal-Dateien können mit den Optionen `-user`, `-system`, `-directory` und `-file` verändert werden.

Die Ausgabe wird seitenweise durch *less* geleitet. Lange Zeilen kann man mittels der *“Pfeil-links-“* und *“Pfeil-rechts-“*Tasten betrachten. Die Option *“-no-pager“* deaktiviert die seitenweise Anzeige, wobei die Zeilen auf die Breite des Terminals verkürzt werden.

journalctl bietet zu den nachfolgend beschriebenen Optionen eine ganze Reihe weiterer Möglichkeiten der Filterung und Aufbereitung der Ausgaben. Bitte auch die man-Page [journalctl](#), [Journalabfrage](#) lesen.

1.10.1 Rechte

Dem Benutzer Root und allen Benutzern die Mitglied der Gruppen *“systemd-journal“*, *“adm“* und *“wheel“* sind, wird Zugriff auf das System-Journal und die Journale der anderen Benutzer gewährt. Siduction fügt alle konfigurierten USER der Gruppe *“systemd-journal“* zu.

Das Journal enthält vertrauenswürdige Felder, d.h. Felder, die implizit vom Journal hinzugefügt werden und durch Client-Code nicht geändert werden können. Sie beginnen mit einem Unterstrich. (z.B.: `_PID=`, `_UID=`, `_GID=`, `_COMM=`, `_EXE=`, `_CMDLINE=`)

1.10.2 Ausgabe filtern

- **Optionen:** `- -user`, `- -system`, `- -directory=`, `- -file=`, `- -namespace=`
Die Optionen begrenzen die **Quelle** der Ausgabe auf den genannten Bereich, das Verzeichnis oder die Datei.
- **Optionen:** `-b`, `-k`, `-u`, `-p`, `-g`, `-S`, `-U`
Die Ausgaben dieser Optionen verwenden alle zu Verfügung stehenden Journal-Dateien, es sei denn, eine der zuvor genannten Optionen wird zusätzlich verwendet.
 - **-b** (`- -boot=`)

Zeigt Nachrichten von einem bestimmten Systemstart. Ohne Argument werden die Protokolle für den aktuellen Systemstart angezeigt. Das Argument “-1” gibt die Meldungen des Systemstarts vor dem Aktuellen aus. Das Argument “5” präsentiert die Meldungen des fünften Systemstarts seit Beginn der Aufzeichnungen.

- **-k** (- -dmesg)
Zeigt nur Kernelnachrichten. Dies beinhaltet die Option “-b”, sodass nur Kernelmeldungen seit dem aktuellen Systemstart ausgegeben werden.
- **-u** (- -unit=)
Diese Option benötigt die Angabe einer UNIT oder eines MUSTERS. Gibt die Journaleinträge für die angegebene Systemd-Unit UNIT oder für alle Units, die auf das MUSTER passen, aus.
- **-p** (- -priority=)
Filtert die Ausgabe nach Nachrichtenprioritäten oder Prioritätsbereichen. Benötigt die Angabe einer einzelnen Protokollstufe, oder einen Bereich von Protokollstufen in der Form VON..BIS.
Die Protokollstufen sind die normalen Syslog-Protokollstufen: “emerg” (0), “alert” (1), “crit” (2), “err” (3), “warning” (4), “notice” (5), “info” (6), “debug” (7)
Als Argument können sowohl die Namen als auch die Ziffern der Protokollstufen verwendet werden. Falls eine einzelne Protokollstufe angegeben ist, werden alle Nachrichten mit dieser oder einer niedrigeren Protokollstufe angezeigt.
- **-g** (- - grep=)
Benötigt die Angabe eines PERL-kompatiblen regulären Ausdrucks, um die Ausgabe zu filtern. Der reguläre Ausdruck wird in den Journaleinträgen auf das Feld “MESSAGE=” angewendet.
- **-S** (- -since=) und **-U** (- -until=)
Die Anzeige beginnt mit neueren Einträgen ab dem angegebenen Datum oder älteren Einträgen bis zum angegebenen Datum. Das Datumsformat sollte “2012-10-30 18:17:16” sein, es können aber auch Teile davon weggelassen werden. Alternativ sind die Zeichenketten “yesterday”, “today”, “tomorrow” möglich. Das Argument “now” bezieht sich auf die aktuelle Zeit. Die Angabe relativer Zeiten ermöglichen ein vorangestelltes “-” oder “+”, die sich auf Zeiten vor bzw. nach der angegebenen Zeit beziehen.

1.10.3 Ausgabe steuern

- Optionen: -f, -n, -r, -o, -x, - --no-pager
 - **-f** (- follow)
Nur die neusten Journal-Einträge anzeigen und kontinuierlich neue Einträge ausgeben. Dies beinhaltet die Option “-n”. Die Ausgabe ist vergleichbar mit dem altbekannten Befehl “*tail -f /var/log/messages*”.
 - **-n** (- lines=)
Zeigt die neusten Journal-Einträge und begrenzt die Anzahl der zu zeigenden Ereignisse. Das Argument ist eine positive Ganzzahl. Der Vorgabewert ist 10, falls kein Argument angegeben wird.
 - **-r** (- reverse)
Die Ausgabe beginnt mit dem neusten Eintrag.
 - **-o** (- output=)
Steuert die Formatierung der angezeigten Journal-Einträge. Dieser Option sind eine ganze Reihe weiterer Optionen untergeordnet, von denen wir hier nur die Option “short-full” betrachten.
 - o short-full**
Die Ausgabe ist größtenteils identisch zu der Formatierung klassischer Syslog-Dateien. Sie zeigt eine Zeile pro Journal-Eintrag an, aber der Zeitstempel wird im Format, das die Optionen –since= und –until= akzeptieren, ausgegeben. Deshalb eignet sich diese Ausgabe sehr gut um nachfolgend eine zeitbezogene Filterung der Journaleinträge zu erstellen.
 - **-x** (- catalog)
Ergänzt Protokollzeilen mit erklärenden Hilfetexten, soweit diese verfügbar sind.
 - **--no-pager**
Die Option deaktiviert die seitenweise Anzeige, wobei die Zeilen auf die Breite des Terminals verkürzt werden. Sie zu benutzen ist nur sinnvoll, wenn für die Ausgabe nur eine geringe Anzahl an Zeilen erwartet wird.

1.10.4 journalctl steuern

Die folgenden Optionen behandeln die Verwaltung der von *journald* geschriebenen Daten.

- **- -disk-usage**
Zeigt den aktuellen Plattenplatzverbrauch aller Journal-Dateien an.
- **- -vacuum-size=, - -vacuum-time=, - -vacuum-files=**
Entfernt die ältesten archivierten Journal-Dateien, bis der Plattenplatz, den sie verwenden, unter die angegebene Größe fällt oder alle archivierten Journal-Dateien, die keine Daten älter als die angegebene Zeitspanne enthalten oder so dass nicht mehr als die angegebene Anzahl an separaten Journal-Dateien verbleiben. Die Ausführung von “*-vacuum-**” bezieht nicht die aktiven Journal-Dateien ein.
- **- -rotate**
Bittet den Journal-Daemon, die Journal-Dateien zu rotieren. Journal-Dateien-Rotation hat den Effekt, dass alle derzeit aktiven Journal-Dateien als archiviert markiert und umbenannt werden, so dass in der Zukunft niemals mehr in sie geschrieben wird. Dann werden stattdessen neue (leere) Journal-Dateien erstellt. Diese Aktion kann mit “*-vacuum-**” in einem einzigen Befehl kombiniert werden, um die “*-vacuum-**” mitgegebenen Argumente tatsächlich zu erreichen.
- **- -verify**
Prüft die Journal-Dateien auf interne Konsistenz.

1.10.5 journalctl beherrschen

Wie oben unter [Rechte](#) beschrieben, kannst du das Journal als einfacher User benutzen. Hier sind einige Beispiele:

journalctl	das volle Journal aller User, älteste Einträge zuerst
journalctl -r	wie zuvor, neueste Einträge zuerst
journalctl -b	zeigt das Protokoll des letzten Bootvorgangs
journalctl -b -1 -k	zeigt vom vorletzten Bootvorgang (-1) alle Kernmeldungen
journalctl -b -p err	limitiert auf den letzten Boot und die Priorität ERROR
journalctl --since=yesterday	zeigt das Journal seit gestern
journalctl /dev/sda	zeigt das Journal der Gerätedatei /dev/sda

<code>journalctl /usr/bin/dbus-daemon</code>	zeigt alle Logs des D-Bus-Daemon
<code>journalctl -f</code>	zeigt eine Liveansicht des Journal (früher: <code>tail -f /var/log/messages</code>)

Die Option “`--list-boots`” gibt die entsprechende Liste aus.

```
# journalctl --list-boots --no-pager
[...]
```

-50	8fc07f387...	Sun	2021-02-28	11:07:05	-CET	Sun	2021-02-28	↗
	23:01:56	CET						
-49	aa49cb3af...	Mon	2021-03-01	17:49:58	-CET	Mon	2021-03-01	↗
	20:19:59	CET						
-48	3a6e55a4a...	Tue	2021-03-02	12:18:46	-CET	Tue	2021-03-02	↗
	20:47:24	CET						
-47	a46150a19...	Wed	2021-03-03	11:06:29	-CET	Wed	2021-03-03	↗
	20:33:09	CET						
-46	d42ed8b05...	Thu	2021-03-04	10:59:56	-CET	Thu	2021-03-04	↗
	19:53:26	CET						
-45	566f65991...	Thu	2021-03-04	19:53:52	-CET	Thu	2021-03-04	↗
	19:55:38	CET						
-44	8e2da4a61...	Fri	2021-03-05	10:15:18	-CET	Fri	2021-03-05	↗
	10:20:11	CET						

```
[...]
```

Anschließend kannst du dir mit dem Befehl `journalctl -b -47` die Meldungen des Bootvorgangs vom 3.3.2021 anzeigen lassen.

Eine weitere Neuerung beim Protokollieren ist die Tab-Vervollständigung für `journalctl`. Wenn Du *journalctl* schreibst, und zwei mal die TAB-Taste drückst, erscheint eine Liste möglicher Vervollständigungen:

```
$ journalctl
_AUDIT_FIELD_APPARMOR=      ERRNO=      SEAT_ID↗
=
_AUDIT_FIELD_CAPABILITY=    _EXE=      ↗
_SELINUX_CONTEXT=
_AUDIT_FIELD_CAPNAME=      EXECUTABLE= ↗
_SESSION_ID=
_AUDIT_FIELD_DENIED_MASK=  EXIT_CODE= ↗
_SHUTDOWN=
_AUDIT_FIELD_INFO=         EXIT_STATUS= SLEEP=
_AUDIT_FIELD_NAME=         _FSUID=    ↗
_SOURCE_MONOTONIC_TIMESTAMP=
_AUDIT_FIELD_OPERATION=    _GID=      ↗
_SOURCE_REALTIME_TIMESTAMP=
```

_AUDIT_FIELD_OUID=	GLIB_DOMAIN=	↗
_STREAM_ID=		
_AUDIT_FIELD_PEER=	GLIB_OLD_LOG_API=	↗
SYSLOG_FACILITY=		
_AUDIT_FIELD_PROFILE=	_HOSTNAME=	↗
SYSLOG_IDENTIFIER=		
_AUDIT_FIELD_REQUESTED_MASK=	INVOCATION_ID=	↗
SYSLOG_PID=		
_AUDIT_FIELD_SIGNAL=	JOB_ID=	↗
SYSLOG_RAW=		
_AUDIT_ID=	JOB_RESULT=	↗
SYSLOG_TIMESTAMP=		
_AUDIT_LOGINUID=	JOB_TYPE=	↗
_SYSTEMD_CGROUP=		
_AUDIT_SESSION=	JOURNAL_NAME=	↗
_SYSTEMD_INVOCATION_ID=		
_AUDIT_TYPE=	JOURNAL_PATH=	↗
_SYSTEMD_OWNER_UID=		
_AUDIT_TYPE_NAME=	_KERNEL_DEVICE=	↗
_SYSTEMD_SESSION=		
AVAILABLE=	_KERNEL_SUBSYSTEM=	↗
_SYSTEMD_SLICE=		
AVAILABLE_PRETTY=	KERNEL_USEC=	↗
_SYSTEMD_UNIT=		
_BOOT_ID=	LEADER=	↗
_SYSTEMD_USER_SLICE=		
_CAP_EFFECTIVE=	LIMIT=	↗
_SYSTEMD_USER_UNIT=		
_CMDLINE=	LIMIT_PRETTY=	↗
THREAD_ID=		
CODE_FILE=	_LINE_BREAK=	↗
TIMESTAMP_BOOTTIME=		
CODE_FUNC=	_MACHINE_ID=	↗
TIMESTAMP_MONOTONIC=		
CODE_LINE=	MAX_USE=	↗
_TRANSPORT=		
_COMM=	MAX_USE_PRETTY=	↗
_UDEV_DEVNODE=		
COMMAND=	MESSAGE=	↗
_UDEV_SYSNAME=		
CONFIG_FILE=	MESSAGE_ID=	_UID=
CONFIG_LINE=	NM_CONNECTION=	UNIT=
CURRENT_USE=	NM_DEVICE=	↗
UNIT_RESULT=		
CURRENT_USE_PRETTY=	NM_LOG_DOMAINS=	USER_ID ↗
=		

DISK_AVAILABLE=	NM_LOG_LEVEL=	↵
USER_INVOCATION_ID=		
DISK_AVAILABLE_PRETTY=	N_RESTARTS=	↵
USERSPACE_USEC=		
DISK_KEEP_FREE=	_PID=	↵
USER_UNIT=		
DISK_KEEP_FREE_PRETTY=	PRIORITY=	

Die meisten davon sind selbsterklärend. Beispielsweise `COMM`, was für *command* steht, bedient eine Menge an Optionen:

“*journalctl* `_COMM=`” listet nach einem weiterer Druck auf `TAB` die möglichen Applikationen:

```
$ journalctl _COMM=
acpid          cups          kdm           ntp           ↵
  sensors      systemd-shutdow
acpi-fakekey   dbus-daemon   keyboard-setup ntpd          sh ↵
               systemd-udev
acpi-support   ddclient      loadcpufreq   ntpdate       ↵
  smartmontools teamviewer
alsactl        docvert-convert logger         ofono         smbld ↵
               udev-configure-
anacron        glances       login         ofonod         ssh ↵
               udisksd
apache2        gpasswd       lvm           pkexec         sshd ↵
               udisks-daemon
backlighthelper gpm           lvm2          polkitd        su ↵
               umount
bash           groupadd      mbmon         pulseaudio     sudo ↵
               uptimed
bluetoothd    hddtemp       mdadm         pywwetha       ↵
  sysstat      useradd
chfn           hdparm        mdadm-raid    pywwetha.py    ↵
  systemd      usermod
chrome         hp            mtp-probe     resolvconf     ↵
  systemd-fsck vboxdrv
console-kit-dae hpfax         mysql         rpcbind        ↵
  systemd-hostnam VBoxExtPackHelp
console-setup ifup          networking    rpc.statd       ↵
  systemd-journal vdr
cpufrequtils  irqbalance   nfs-common    samba-ad-dc     ↵
  systemd-logind winbind
cron           kbd          mbd           saned           ↵
  systemd-modules
```


Mit “*journalctl _COMM=su*” kannst du nun sehen, welcher User sich wann mit “su” root-Rechte verschafft hat.

```
# journalctl _COMM=su
-- Boot 1b5d2b3fcd9043d88d8abce665b75ed4 --
Mär 10 13:45:53 pc1 su[75259]: (to root) siduser on pts/3
Mär 10 13:45:53 pc1 su[75259]: pam_unix(su:session): session opened↵
    for user root(uid=0) by (uid=1000)
Mär 10 16:27:22 pc1 su[105197]: (to root) siduser on pts/1
Mär 10 16:27:22 pc1 su[105197]: pam_unix(su:session): session ↵
    opened for user root(uid=0) by (uid=1000)
Mär 10 17:54:33 pc1 su[105197]: pam_unix(su:session): session ↵
    closed for user root
Mär 10 17:54:42 pc1 su[75259]: pam_unix(su:session): session closed↵
    for user root
-- Boot 37b19f6321814620be1ed4deb3be467f --
Mär 10 17:56:35 pc1 su[3381]: (to root) siduser on pts/1
Mär 10 17:56:35 pc1 su[3381]: pam_unix(su:session): session opened ↵
    for user root(uid=0) by (uid=1000)
Mär 10 19:07:17 pc1 su[3381]: pam_unix(su:session): session closed ↵
    for user root
```

Ein anderes Beispiel:

Man kann die Ausgabe zusätzlich zeitlich eingrenzen.

```
# journalctl _COMM=dbus-daemon --since=2020-04-06 --until↵
    ="2020-04-07 23:40:00"
[...]
Apr 07 22:59:04 pc1 org.gtk.Private.GPhoto2VolumeMonitor[2006]: ###↵
    debug: in handle_supported
Apr 07 22:59:04 pc1 org.gtk.Private.GPhoto2VolumeMonitor[2006]: ###↵
    debug: in handle_list
Apr 07 22:59:04 pc1 org.gtk.Private.GoaVolumeMonitor[2006]: ### ↵
    debug: in handle_supported
Apr 07 22:59:04 pc1 org.gtk.Private.GoaVolumeMonitor[2006]: ### ↵
    debug: in handle_list
Apr 07 23:03:09 pc1 org.gtk.Private.GPhoto2VolumeMonitor[2006]: ###↵
    debug: Name owner ':1.4320' vanished
Apr 07 23:03:09 pc1 org.gtk.Private.GoaVolumeMonitor[2006]: ### ↵
    debug: Name owner ':1.4320' vanished
Apr 07 23:03:09 pc1 org.gtk.Private.AfcVolumeMonitor[2006]: ### ↵
    debug: Name owner ':1.4320' vanished
Apr 07 23:03:09 pc1 org.gtk.Private.MTPVolumeMonitor[2006]: ### ↵
    debug: Name owner ':1.4320' vanished
```

Viele der oben genannten Optionen lassen sich miteinander kombinieren, damit nur die gesuchten Journaleinträge angezeigt werden. Die man-page von [journalctl](#) beschreibt alle Optionen ausführlich.

1.10.6 Quellen

[systemd-journald](#)
[journald Konfiguration](#)
[journalctl, Journalabfrage](#)
[journal gatewayd](#)
[journal remote](#)
[journal upload](#)
[systemd-cat.1.de](#)

Dank an Helge Kreuzmann für die deutschen Übersetzungen.

Seite zuletzt aktualisiert 2021-03-20