SelfLinux-0.13.1

syslog-ng

Autor: Winfried Müller (*wm@reintechnisch.de*)
Formatierung: Florian Frank (*florian@pingos.org*)

Inhaltsverzeichnis

1 Einführung

Logfiles werden gerne übersehen und sind doch so extrem wichtig und hilfreich. Einer der großen Vorzüge von Linux ist, dass es fast alles im System protokollieren kann. So kann man immer nachvollziehen, was wann wo passiert oder eben schief läuft.

Logging wird gerade aus Sicherheitsgründen immer wichtiger. Hierüber lässt sich früh erkennen, wann wo im System ein Einbruchsversuch läuft. Neben dem Logging sind vor allem gute Logfile-Analyse-Werkzeuge notwendig.

Der Standard syslog Daemon unter Linux und vielen Unix Varianten ist in vielerlei Hinsicht sehr eingeschränkt. Er konnte mir nicht das liefern, was ich brauchte, um Logfiles vernünftig und einfach auszuwerten.

Ich bin nicht der einzige, dem der syslog Daemon ziemlich angestaubt und unzulänglich vorkam. Und so machte sich dann *Balazs Scheidler* 1998 auf den Weg, einen neuen besseren syslog zu schreiben. Diesen nannte er `syslog-ng`. Das **ng** steht für **New Generation**.

`syslog-ng` ist mittlerweile ein fest etablierter und stabil laufender Ersatz für den syslog. Alle großen Distributionen haben ihn mit an Board, jedoch muss er meist noch aktiviert werden. Es spricht eigentlich alles dafür, auf `syslog-ng` umzusteigen, sobald man Möglichkeiten braucht, die über den normalen syslog hinausreichen.

Diese Beschreibung bezieht sich auf die `syslog-ng` Version 1.5.15, wie sie in Debian Woody enthalten ist.

2 Ein Anwendungsfall

Als ich anfing, mir ein Logfile-Analyse-Werkzeug zu bauen, merkte ich, dass der Standard syslog Dämon von Linux und vielen Unix-Systemen doch erhebliche Einschränkungen mit sich bringt. Ich wollte ein Logfile in einer ähnlichen Form, wie es moderne Anwendungen wie cups, samba oder Apache machen. Das Datum sollte mit Jahresangabe sein, Priorität, Facility und das Programm, was loggt, sollten ebenfalls auftauchen. Eine Zeile sollte etwa so aussehen:

```
[2003/10/13 12:00:42] info mail fetchmail fetchmail[30924]: No mail for ...
```

Zuerst das Datum in einer samba-like Form, dann die Log-Priorität, die Facility (also der Systembereich, der loggt), das Programm und die Message. Alle erzeugten Logs des Systems, die über syslog laufen, sollten in einer Logdatei landen. Hierzu fügte ich in der syslog-ng.conf folgende Zeilen hinzu:

```
destination mysyslog {
  file("/var/log/mysyslog"
  owner("root")
  group("adm")
  perm(0640)
  template( "[$YEAR/$MONTH/$DAY $HOUR:$MIN:$SEC] $PRIORITY
             $FACILITY $PROGRAM $MESSAGE\n")); };

log { source(src);  destination(mysyslog); };
```

Mit **destination** wird ein ▶ destination-Objekt festgelegt, welches **mysyslog** heißt und mit der Datei /var/log/mysyslog verbunden wird. Das Ausgabe-Format beschreibt ein Template. Über die Zeile **log** wird dann ein zuvor definiertes ▶ source-Objekt und mein ▶ destination-Objekt zu einer Log-Aktion zusammengefügt.

3 Konfigurationsdetails

Die Konfigurations-Syntax ist an C/Java/PHP angelehnt und sehr einfach gehalten. Man sollte jedoch aufpassen, Semikolons immer richtig zu setzen.

Leider gehört es zum Charakter von Linux, dass fast jeder Autor eines Paketes sich in einer eigenwilligen Konfigurations-Syntax verewigt. Wie schön wäre es, wenn gerade bei der Konfigurations-Sprache sich ein Standard durchsetzt, an den sich alle halten. Manche großen Pakete wie samba, cups, apache machen schon gute Vorgaben, die andere Paketentwickler übernehmen. Der Einstieg in neue Pakete wird so wesentlich vereinfacht, weil man nur noch wissen muss, was man konfigurieren will und nicht mehr, wie man überhaupt konfigurieren kann. Ein vielversprechender Ansatz ist http://config4gnu.sourceforge.net/docs/article.html.

Die gesamte Konfiguration erfolgt über die Datei syslog-ng.conf, die bei Debian Linux unter /etc/syslog-ng/syslog-ng.conf liegt.

In dieser Konfigurations-Datei gibt es prinzipiell folgende Einträge:

```
options { option; option; .. };
```

Hiermit können globale Optionen festgelegt werden.

```
source <identifier> { source-driver(params); ... };
```

Mit jeder source Zeile wird eine syslog-Quelle festgelegt.

```
filter <identifier> { expression; ... };
```

Hiermit können beliebig viele Filter-Objekte angelegt werden, die später in der log-Zeile benutzt werden können. Expression beschreibt die Filterregel.

```
destination <identifier> {dest-driver(params); ... };
```

Hiermit kann man Ziel-Objekte anlegen, wohin also geloggt werden soll und mit welchen Einstellungen.

```
log { source(s1); source(s2); ...; filter(f1); filter(f2); ...; destination (d1);
      destination(d2); ...; flags( flag1; ...) }
```

Hiermit werden log-Objekte angelegt und log-Objekte sind die einzigen, die auch Aktionen ausführen, wenn sie definiert sind. In dem Log-Objekt werden zuvor definierte Objekte (source, filter, destination) zusammengefügt und das Logging somit aktiviert.

Nochmal zum Verständnis: Man kann beliebig viele source, filter und destination-Objekte anlegen, die von sich

aus gar nichts tun. Erst das Einbinden in ein log-Objekt, aktiviert eine Logausgabe. Das log-Objekt ist das einzige, was irgendwie aktiv wird.

Dieser objektorientierte Aufbau ist sehr leicht überschaubar und wartbar. Nach ein wenig Praxis wird man spüren, wie angenehm diese Form zu handeln ist.

3.1 Das options Objekt

Hier kann man globale Einstellungen vornehmen. Das meiste ist für erste Gehversuche schon korrekt eingestellt, es geht hauptsächlich um Feintuning. Einige wichtige Einstellungen sind:

* **sync(n-lines)**
  Hiermit kann man festlegen, wieviele Log-Zeilen auflaufen sollen, bis gesynct wird, bis also die Daten tatsächlich auf der Platte abgelegt werden und nicht noch in irgendeinem Puffer im Ram stecken. Wer sicher gehen will und auch keine Performance-Probleme hat, sollte hier 0 einstellen - es wird dann sofort jede Zeile geschrieben.
* **log_fifo_size(n-lines)**
  Anzahl der Zeilen, die zwischengepuffert werden können. Ist wichtig, wenn mehr Logs einlaufen, als syslog-ng wegschreiben kann. Das kann z. B. passieren, wenn eine kurzzeitige Logzeilen-Flut hereinbricht. Ein typischer Wert ist 1000.

3.2 Das source Objekt

Hier kann man die Quellen angeben, woher `syslog-ng` Meldungen empfangen soll. Unter debian Linux reicht für das normale Logging folgende Zeile:

```
source src { unix-stream("/dev/log"); internal(); };
```

Es wird hier ein neues Source-Objekt namens **src** angelegt. Dieses Source-Objekt wird nun mit zwei Quellen verbunden, zum einen mit /dev/log, zum anderen mit **internal**. Der Typ **internal** steht für die Messages, die `syslog-ng` selber erzeugt. Die sollte man also immer mit aufnehmen. /dev/log ist vom Typ unix-stream, ein Stream mit unix-style Übertragung, was man typischerweise unter Linux nutzt. Ein ähnlicher Typ ist unix-dgram, der in BSD Systemen verwendet wird. Unter Linux funktioniert generell auch unix-dgram, es ist aber nicht so sicher, weil bei zu hoher Systemlast Logs verloren gehen können.

Weitere Möglichkeiten von Typen einer Datenquelle sind **file**, **pipe**, **udp**, **tcp**. Hiermit kann man verrückte Sachen anstellen. Im Normalfall dürfte lediglich udp/tcp noch interessant sein, mit dem `syslog-ng` als Loghost übers Netz fungieren kann. Er verhält sich dann genauso, wie ein syslog, der per udp lauscht. Eine vollständige udp Quelle sieht so aus:

```
udp( ip(0.0.0.0) port(514) )
```

Wobei **ip** und **port** auch weggelassen werden können, wenn die hier angegebenen Defaultparameter genommen werden sollen. Ip steht für die IP-Adresse, auf der gelauscht werden soll. Hat ein Rechner also mehrere IP-Adressen, kann man hier Einschränkungen machen. Default ist 0.0.0.0, was auf allen Adressen lauscht. Man kann nicht angeben, das z. B. nur Logs von einem bestimmten Host angenommen werden. Port gibt den Port an, auf dem gelauscht werden soll. Default ist lt. altem syslog der Port 514.

UDP ist ein verbindungsloses Protokoll, es kann also passieren, dass bei verlorenen Paketen auch Logs verloren gehen, weil diese nicht erneut angefordert werden. Besser ist es daher, tcp einzusetzen. Konfiguriert wird es genauso, man kann auch den gleichen Port nutzen, insofern man auf das sonst dort sitzende rshell verzichten kann (seit ssh sollte die eh überflüssig sein). Bei tcp gibt es zusätzlich noch den Parameter **max-connections(n)**, der die Anzahl der Verbindungen limitiert.

Wer allerdings von alten syslogs aus Logs entgegennehmen muss, der ist auf udp angewiesen, weil die nur udp können.

Es gibt noch eine weitere Quelle, die man evtl. nicht vergessen darf - die Kernel-Messages. Wenn man auf seinem Linux-System einen `klogd` laufen hat, dann holt dieser sich alle Kernelmessages aus /proc/kmsg und reicht sie an den Syslog- Daemon (hier: `syslog-ng`) weiter. Somit hat man diese Messages automatisch. Das dürfte bei den meisten Distributionen der Standard-Fall sein. Der `klogd` hat auch den Vorteil, das er manche Logs noch weiter aufbereitet, z. B. Funktionsnamen über die .map Kernel Files dekodiert. Dadurch können Probleme leichter gefunden werden.

Läuft kein `klogd`, so muss man diese Kernel-Messages selber abholen. Eine komplette Sourcezeile könnte dann so aussehen:

```
source src { unix-stream("/dev/log"); internal(); file("/proc/kmsg");};
```

Manche Distributionen benutzen hier auch **pipe()** statt **file()** für /proc/kmsg.

3.3 Das destination Objekt

Mit dem Destination Objekt kann man Ziele festlegen, wohin ein Log-Stream gehen soll. Normal ist dies eine Datei. Ein einfaches Ziel könnte so aussehen:

```
destination syslog { file("/var/log/syslog" owner("root") group("adm") perm (0640)); };
```

Hier wird das Ziel mit dem Namen **syslog** angelegt, wobei es mit der Datei /var/log/syslog verbunden wird. Diese Datei soll mit dem Benutzer root.adm angelegt werden und 0640er Rechte haben.

Neben den **file()**-Optionen **owner()**, **group()**, **perm()** gibt es weitere, die man bei speziellen Ansprüchen nutzen kann. Eine sehr interessante Option ist die Möglichkeit, ein Ausgabeformat mit **template()** festzulegen. Dies hatte ich weiter oben an einem Beispiel schon verdeutlicht. Hiermit kann man sich nahezu beliebige Logfile-Formate erstellen. Das Template ist ein String, in dem Makros mit einem $MakroName eingefügt werden können.

Eine typisches Template könnte so aussehen:

```
template( "[$YEAR/$MONTH/$DAY $HOUR:$MIN:$SEC] $PRIORITY $FACILITY $MESSAGE\n")
```

Hier wird zuerst das Datum in einer typischen Form zusammengefügt, dann die Priorität, die Facility und die Message ausgegeben.

Folgende Makros sind verfügbar:

FACILITY	Facility der Message. Das ist eine der vordefinierten Gruppen des Subsystems, von der eine Message kommt. Möglich sind hier auth, auth-priv, cron, daemon, ftp, kern, lpr, mail, mark, news, syslog, user, uucp, local0 - local7. Was welche bedeuten und welches Programm welche Facility benutzt, ist nicht immer einfach herauszufinden. Ein Beobachten der syslogs hilft am besten. Viele Programme können vor dem Kompilieren eingestellt werden, unter welcher Facility sie loggen. Das Facility-System kann man als starr und veraltet betrachten, weil es für die meisten Filteraufgaben zu grob und unflexibel ist.
PRIORITY oder LEVEL	Die Priorität der Message. Hier gibt es: debug, info, notice, warn, err, crit, alert, emerg.
TAG	Die Priorität und Facility als 2-Zeichen Hexzahl codiert.
DATE, FULLDATE, ISODATE	Das Datum in verschiedenen standardisierten Formaten.
YEAR	Das Jahr 4-stellig.
MONTH	Der Monat 2-stellig numerisch, ggf. führende 0.
DAY	Der Tag 2-stellig numerisch, ggf. führende 0.
WEEKDAY	Wochentag 3 Buchstaben, wie unter Unix gewohnt. (Mon, Tue...)
HOUR	Die Stunde 2-stellig, ggf. führende 0.
MIN	Die Minute 2-stellig, ggf. führende 0.
SEC	Die Sekunden 2-stellig, ggf. führende 0.
FULLHOST	Vollständig qualifizierter Host, also host.domain
HOST	Hostname ohne Domainzusatz.
PROGRAM	Das Programm, welches die Messages abgesetzt hat. Hierüber lässt sich oft flexibler filtern, als über die Facilitys.
MSG oder MESSAGE	Die eigentliche Message. Da es im alten syslog keine Möglichkeit gab, das Programm oder den Prozess mitzuloggen, hat es sich als Standard herausgebildet, in die Message als erstes das Programm mit Prozessnummer anzugeben (Programm[ps-id]:), welches die Message ausgab. Bei Unterprozessen eines Programmes, wird normal Programm/Unterprozess[ps-id]: verwendet. Erst hinter dem Doppelpunkt beginnt die eigentliche Message. Verlassen kann man sich jedoch auf diese Regel nicht, Ausnahmen gibt es immer wieder.

Es gibt bei **file()** einen leistungsfähigen Mechanismus, die Makro-Substitution für den Dateinamen. So kann man sich dynamische Logfilenamen generieren. Hier kann man die gleichen Makros wie für **template()** benutzen. Ein

```
destination mylog { file("/var/log/syslog-$HOST" owner("root") group("adm") perm(0640)); };
```

loggt z. B. jeden Host in in eine getrennte Datei in der Form syslog-MeinErsterHost, syslog-NochEinHost. Natürlich muss man hier auch vorsichtig sein, um DoS- Attacken nicht Tür und Tor zu öffnen. Wer jedoch spezielle Möglichkeiten des Loggens braucht, kommt mit diesem Feature vielleicht weiter.

Neben file gibt es noch folgende Ziel-Typen bzw. Ziel-Treiber: tcp, udp, unix- stream, unix-dgram, fifo, usertty, program.

Udp oder tcp nutzt man ähnlich, wie schon bei dem Source-Objekt beschrieben. Als Ziel angegeben, schickt der `syslog-ng` nun diesen Log-Stream zu einem anderen Rechner per tcp oder udp. Ein Beispiel:

```
destination a_udp { udp( "192.168.0.12" port(514) ); };
```

Hier sollen an den udp-Port 514 des Rechners mit der IP 192.168.0.12 Messages verschickt werden. Port 514 ist sowieso Default, könnte hier also auch weggelassen werden.

Hier gilt auch: Udp nimmt man aus Kompatibilitätsgründen, weil sich dann `syslog-ng` wie ein altes syslog verhält. Tcp ist das sicherere Verfahren, weil bei diesem verbindungsorientierten Protokoll Pakete nicht verloren gehen können.

Auch udp ist relativ sicher, es werden nicht regelmäßig Packete verloren gehen, vor allem nicht im lokalen Netzwerk. Nur wenn Hardware oder Leitungen defekt sind, kommt es zu Datenverlusten. Aber auch tcp kann durch einen kaputte Leitung nichts mehr schicken, es kann lediglich bei einer kurzen Störung erneut versenden. Man sollte sehen, dass udp Jahrzehnte erfolgreich für diese Aufgabe eingesetzt wird.

Mit usertty kann man Meldungen auf dem Terminal eines Benutzers ausgeben, der natürlich eingeloggt sein muss. Hier ein Beispiel:

```
destination admin_tty { usertty(admin); };
```

Sollen Meldungen an alle eingeloggten Benutzer gehen, nimmt man:

```
destination warn_to_all { usertty(*); };
```

3.4 Das filter Objekt

Filter Objekte legen fest, wie Meldungen von einem Source-Objekt gefiltert werden sollen. Hiermit lassen sich also gewünschte Messages aus dem gesamten Datenstrom eines Source-Objektes herauspicken. Und das ist gut, wenn man z.B. in einem Ziel nur bestimmte Meldungen loggen möchte. Auch der alte syslog hat eine einfache Filtersprache, um Logausgaben in verschiedene Logdateien zu schreiben.

Mit `syslog-ng` kann man wesentlich erweitert und verfeinert filtern.

Ein typisches Filterobjekt könnte so definiert werden:

```
filter f_cnews { level(notice, err, crit) and facility(news); };
```

Hier werden alle Meldungen durchgelassen, die vom Level oder der Priority auf notice, err oder crit stehen und die die Facility news haben.

Filterfunktionen lassen sich mit and, or, not verknüpfen und auch klammern. Somit kann man sehr leistungsfähige Filterkonstrukte erstellen.

Wer nicht genau weiß, wie and, or, not aufgelöst werden, sollte besser einmal zuviel klammern, als sich später zu wundern, warum was merkwürdiges bei heraus kommt. Das hilft auch anderen, die die Konfiguration verstehen

wollen und auch nicht so genau bescheid wissen. Kurz gesagt bindet and mehr als or mehr als not. Ganz ähnlich wie Punktrechnung vor Strichrechnung kommt. a or b and c ist was anderes wie (a or b) and c.

Folgende Filterfunktionen gibt es:

* **facility(facility1,facility2,...)**
  Lass alle Messages durch, die dieser Facility entsprechen.
* **level(prio1, prio2,...) (oder synonym priority())**
  Lass alle Messages durch, die der angegebenen Priorität/Level entsprechen.
* **program(regexp)**
  Alle Meldungen, die vom Programm kommen, worauf regexp passt, werden durchgelassen. Hierbei ist regexp ein regulärer Ausdruck. Hat man Whitespaces im Programmnamen, sollte man den Ausdruck zwischen doppelte Anführungsstriche setzen.
* **host(regexp)**
  Filterung nach Host, woher die Message kommt, ebenfalls regulärer Ausdruck. Ein Tipp, wenn du nichts von regulären Ausdrücken verstehst: Nimm einfach den Hostnamen, z. B. so: host(myhost). Genau genommen müsstest du host("^myhost$") schreiben.
* **match(regexp)**
  Dies lässt nur Meldungen durch, wo die eigentliche Message auf dieses Muster passt. Dies ist ein sehr leistungsfähiger Mechanismus, lassen sich doch so ganz gezielt Meldungen herausfischen. Es gibt fast nichts, was man nicht mit regulären Ausdrücken erschlagen könnte.
* **filter(filter_name)**
  Um kompliziertere Filterregeln zu erstellen, kann man mehrere Filter zu einem neuen Filterkonstrukt zusammenfügen. Hiermit kann man also andere Filter in einem neuen Filter aufrufen. Damit kann man verschachtelte Filterkonstrukte erstellen. Es ist oft auch für die Lesbarkeit besser, zuerst mehrere Teilfilter zu definieren, die man dann in einer weiteren Filterregel zusammenfügt.

Mach keine Meisterschaft daraus, möglichst komplizierte Konfigurationen zu produzieren. Durch komplizierte Filterregeln kann man das hier durchaus schaffen. Das zeigt zwar deine intellektuellen Fähigkeiten, produziert aber schwer wartbare Konfig-Dateien. Mach es so einfach wie möglich, andere werden es dir danken.

3.5 Das log Objekt

Alle bisherigen Objekte waren Vorarbeiten, um jetzt Zeilen zu generieren, die wirklich Aktionen auslösen. Denn ohne die log-Objekte würde gar nichts passieren. Die anderen Objekte sind nur Daten-Definitionen. Die log-Objekte führen das eigentliche Logging aus, in dem sie die zuvor definierte Source-, Destination- und Filter-Objekte zu einer Log-Aktion verbinden. Eine typische Zeile sieht so aus:

```
log { source(src); filter(f_syslog); destination(syslog); };
```

Es wird hier also vom Source src gelesen, diese Messages durch den filter **f_syslog** geschickt und dann zum Ziel **syslog** geschrieben.

Mehrere Sourcen bindet man mit mehreren **source()**-Statements ein.

```
log { source(src); source(src1); filter(f_syslog); destination(syslog); };
```

Mehrere Filter und sogar mehrere Ziele lassen sich nach gleichem Schema einbinden.

Hinter **destination()** lässt sich auch noch **flags()** angeben, mit dem man erweitere Funktionalitäten festlegen kann.

4 Beispieldatei

Hier ist ein Beispiel einer kompletten Konfig-Datei, wie ich sie unter Debian Woody verwende. Die Kernel-Messages hole ich hier direkt ohne den `klogd`. Dieser darf also nicht gestartet sein. Es gibt unter Debian Woody noch einige Probleme im Zusammenspiel `klogd`/`syslog-ng`.

```
#
# Syslog-ng configuration file, compatible with default Debian syslogd
# installation. Originally written by anonymous (I can't find his name)
# Revised, and rewrited by me (SZALAY Attila sasa@debian.org)

# First, set some global options.
options { long_hostnames(off); sync(0); stats(3600); };

#
# This is the default behavior of sysklogd package
# Logs may come from unix stream, but not from another machine.
#
source src { unix-stream("/dev/log"); internal(); file("/proc/kmsg"); };

#
# If you wish to get logs from remote machine you should uncomment
# this and comment the above source line.
#
# source src { unix-dgram("/dev/log"); internal(); udp(); };


# After that set destinations.

# First some standard logfile
#
destination authlog { file("/var/log/auth.log" owner("root")
                      group("adm") perm(0640)); };
destination syslog  { file("/var/log/syslog" owner("root")
                      group("adm") perm(0640)); };
destination cron    { file("/var/log/cron.log" owner("root")
                      group("adm") perm(0640)); };
destination daemon  { file("/var/log/daemon.log" owner("root")
                      group("adm") perm(0640)); };
destination kern    { file("/var/log/kern.log" owner("root")
                      group("adm") perm(0640)); };
destination lpr     { file("/var/log/lpr.log" owner("root")
                      group("adm") perm(0640)); };
destination mail    { file("/var/log/mail.log" owner("root")
                      group("adm") perm(0640)); };
destination user    { file("/var/log/user.log" owner("root")
                      group("adm") perm(0640)); };
destination uucp    { file("/var/log/uucp.log" owner("root")
                      group("adm") perm(0640)); };


# This files are the log come from the mail subsystem.
#
destination mailinfo { file("/var/log/mail.info" owner("root")
                       group("adm") perm(0640)); };
destination mailwarn { file("/var/log/mail.warn" owner("root")
                       group("adm") perm(0640)); };
destination mailerr  { file("/var/log/mail.err" owner("root")
                       group("adm") perm(0640)); };

# Logging for INN news system
#
destination newscrit   { file("/var/log/news/news.crit" owner("root")
                         group("adm") perm(0640)); };
destination newserr    { file("/var/log/news/news.err" owner("root")
                         group("adm") perm(0640)); };
destination newsnotice { file("/var/log/news/news.notice" owner("root")
                         group("adm") perm(0640)); };
```

```
# Some `catch-all' logfiles.
#
destination debug      { file("/var/log/debug" owner("root")
                         group("adm") perm(0640)); };
destination messages   { file("/var/log/messages" owner("root")
                         group("adm") perm(0640)); };


# The root's console.
#
destination console { usertty("root"); };

# Virtual console.
#
destination console_all { file("/dev/tty8"); };

# The named pipe /dev/xconsole is for the nsole' utility.  To use it,
# you must invoke nsole' with the -file' option:
#
#    # xconsole -file /dev/xconsole [...]
#
destination xconsole { pipe("/dev/xconsole"); };

destination ppp { file("/var/log/ppp.log" owner("root")
                 group("adm") perm(0640)); };

# Here's come the filter options. With this rules, we can set which
# message go where.

filter f_authpriv  { facility(auth, authpriv); };
filter f_syslog    { not facility(auth, authpriv); };
filter f_cron      { facility(cron); };
filter f_daemon    { facility(daemon); };
filter f_kern      { facility(kern); };
filter f_lpr       { facility(lpr); };
filter f_mail      { facility(mail); };
filter f_user      { facility(user); };
filter f_uucp      { facility(uucp); };

filter f_news      { facility(news); };

filter f_debug     { level(debug) and not
                     facility(auth, authpriv, mail, news); };
filter f_messages  { level(info .. warn) and not
                     facility(auth, authpriv, cron, daemon, mail, news); };
filter f_emergency { level(emerg); };

filter f_info      { level(info); };
filter f_notice    { level(notice); };
filter f_warn      { level(warn); };
filter f_crit      { level(crit); };
filter f_err       { level(err); };

filter f_cnews     { level(notice, err, crit) and facility(news); };
filter f_cother    { level(debug, info, notice, warn) or
                     facility(daemon, mail); };

filter ppp         { facility(local2); };


log { source(src); filter(f_authpriv); destination(authlog); };
log { source(src); filter(f_syslog); destination(syslog); };
#log { source(src); filter(f_cron); destination(cron); };
log { source(src); filter(f_daemon); destination(daemon); };
log { source(src); filter(f_kern); destination(kern); };
log { source(src); filter(f_lpr); destination(lpr); };
log { source(src); filter(f_mail); destination(mail); };
log { source(src); filter(f_user); destination(user); };
log { source(src); filter(f_uucp); destination(uucp); };
log { source(src); filter(f_mail); filter(f_info);
      destination(mailinfo); };
log { source(src); filter(f_mail); filter(f_warn);
```

```
      destination(mailwarn); };
log { source(src); filter(f_mail); filter(f_err);
      destination(mailerr); };
log { source(src); filter(f_news); filter(f_crit);
      destination(newscrit); };
log { source(src); filter(f_news); filter(f_err);
      destination(newserr); };
log { source(src); filter(f_news); filter(f_notice);
      destination(newsnotice); };
log { source(src); filter(f_debug); destination(debug); };
log { source(src); filter(f_messages); destination(messages); };
log { source(src); filter(f_emergency); destination(console); };

#log { source(src); filter(f_cnews); destination(console_all); };
#log { source(src); filter(f_cother); destination(console_all); };


log { source(src); filter(f_cnews); destination(xconsole); };
log { source(src); filter(f_cother); destination(xconsole); };

log { source(src); filter(ppp); destination(ppp); };
```

5 Probleme und Offenes

5.1 Zusammenspiel klogd und syslog-ng

Es gibt ein paar Probleme im Zusammenspiel des `klogd` mit `syslog-ng`. Unter Debian Woody reicht der `klogd` nur Meldungen korrekt weiter, wenn unter `syslog-ng` die Methode **unix-dgram("/dev/log")** anstatt **unix-stream("/dev/log")** im Source Objekt verwendet wird. Auch gibt es Probleme, wenn `syslog-ng` neu gestartet wird, ohne gleichzeitig klogd neu zu starten. Der Datenaustausch von `klogd` an `syslog-ng` funktioniert dann nicht mehr korrekt.

Einige Distributionen sind deshalb dazu übergegangen, entweder den `klogd` nicht zu verwenden oder aber im `syslog-ng` Init-skript sowohl `syslog-ng` wie auch `klogd` zu starten und zu stoppen. Dadurch werden beide Daemons immer korrekt im Tandem hoch- und runtergefahren. Evtl. sollte man 1 Sekunde nach Start des `syslog-ng` warten, bevor man `klogd` startet.

Loggt man ohne `klogd`, fehlt der Prefix **kernel:** im Logfile, den `klogd` normal hinzufügt. Modernere Versionen von `syslog-ng` haben hierfür einen zusätzlichen Befehl, um diesen Prefix selber zu generieren. In einer Source-Zeile könnte dann z. B. stehen:

```
pipe("/proc/kmsg" log_prefix("kernel: ")).
```

Unter Debian Woody sollte man also einfach ohne `klogd` arbeiten oder aber die Init-Skripte anpassen.

5.2 Referenzen

* http://www.balabit.com/products/syslog_ng/
* Referenz Manual im Source-tar (1.5.15) unter doc/sgml/syslog-ng.txt
* http://home.datacomm.ch/prutishauser/texte/syslog-ng-de.txt
* Beispiele im Source-tar (1.5.15) unter contrib/syslog-ng.conf.*
* Beispiele im Source-tar (1.5.15) unter doc/syslog-ng.conf.*
* http://www.linux-magazin.de/Artikel/ausgabe/2003/11/tagebuch/tagebuch.html
* man syslog-ng
* man syslog (lohnt sich, weil viele Standards und Definitionen von syslog auf `syslog-ng` übernommen wurden.)
* man grep (Beschreibung reguläre Ausdrücke (regexp))