• 55 - LOG Files (See english Linux Magazine September 2000 Page 80)

System logging Daemons:

- `klogd`	- Kernel logging daemon (client to `syslogd`)
- `syslogd`	- System wide (all programs) logging Daemon

These files are started by the Script `/etc/init.d/syslog`
Location of standard sytem and kernel log : `/var/log/messages`
Config file for both daemons: `/etc/syslog.conf`

- The `syslodg` Daemon.
  This daemon process runs in the background, receives log information from the kernel and from other applications or Daemons and, according to the configuration in `/etc/syslog.conf`, it distributes the log information either to files or send it to a console or even can send it to a syslog server via network.
  - Each log information contains only a one line message.
  - The same log information can be sent to many files or other destinations.
  - The log information is received with the following content:
    - Facility: `auth,authpriv,cron,daemon,kern,lpr,mail,news,` `local`*`N`*`,user`
    - Priority Level: `debug,info,notice,warning,err,crit,alert,emerg`
    - Tag: (Title) and maybe the PID of process.
    - Log Text : (actual message).
- - The saved or sent information(one line) is having the following content:
  `Date Time Hostname Tag [Process ID] Message`

- **Format of `/etc/syslog.conf`:**

```
facility.level;[facility.level];...... destination
```

IMPORTANT: Remember to use TABs and NOT spaces in your `syslog.conf` file. Otherwise it won't work.

Facilities: (For multiple facilities separate by ',' eg. `auth,cron.*`)

`*`	All Facilities
`auth`	General authentications
`authpriv`	Login authentication
`cron`	cron subsystem
`daemon`	System server processes
`kern`	Linux Kernel
`ftp`	FTP server
`lpr`	Spooling subsystem
`mail`	Mail subsystem
`news`	News subsystem
`uucp`	Unix-To-Unix copy service/Program
`user`	Users system messages
`syslog`	Syslogd daemon
`local`*`N`*	Locally defined syslog facilities *N*=0 to 7
`mark`	Regular mark for reference purposes

Levels: (Priority) From the least to the most important level

	`none`	Exclude messages of this facility eg. `mail.none` = no mail messages
	`*`	All messages
most important	`emerg`	High emergency
^	`alert`	Very critical
|	`crit`	Critical
|	`err`	Errors
|	`warning`	
|	`notice`	
|	`info`	Information
least important	`debug`	Lots of messages. For debugging purposes

Destinations:

- files	eg. `/var/log/cron.msgs`
- devices	eg. `/dev/tty6` (to virtual console 6)
- usernames	eg. `root` (to root) or `*` (to all logged-in users)
- computer	eg. `@moon` (to 'moon' host....syslog server)
- named pipes	eg. `|/dev/xconsole` (to the virtual x console) The `xconsole` can be then started with the command: `kdesu 'xconsole -file /dev/xconsole'`

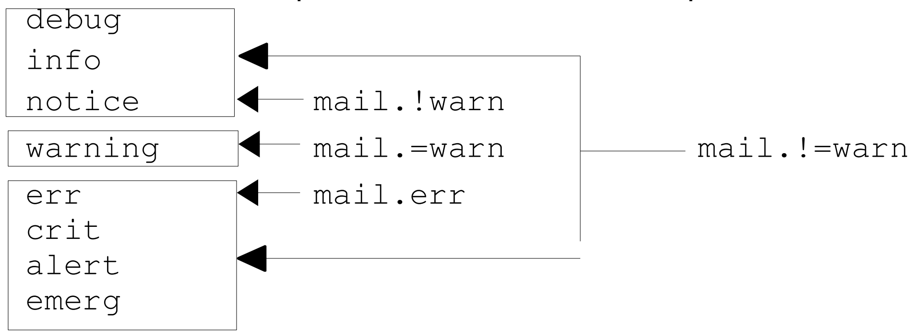
'–' before the filename means: buffered before writing.

```
eg:  *.*;mail.none;auth.none  -/var/log/messages
```

Messages exclusive logging: (examples)

`mail.warn`	log the mail warning or more serious messages
`mail.!warn`	log ONLY the messages less serious than warn
`mail.=warn`	log ONLY the mail warning messages
`mail.!=warn`	log all mail messages except the warning ones

Here are few examples of the above selection possibilities:



General Examples:

```
kern.=warn;*.err;authpriv.none  -/var/log/warnings
```

- send Kernel warnings only
- error messages from all facilities
- BUT none from the `authrpiv`
- to the file `/var/log/warnings`

('-' before the filename means buffered before writing to disk)

```
*.emerg;user.none   *
```

- send to whosoever is now logged-in about real emergencies
- BUT not the messages concerning users

Syslog server (hosts messages logging center)

```
*.info          @mainlogger.gdf.local
```

Send all info level messages and more serious to the host 'mainlogger' using:
Prot: `UDP` port: `514`

The host `mainlogger.gdf.local` will log these messages according to its `/etc/syslog.conf` configuration file as if coming from is local processes. Messages will include the IP of sending host though.

IMPORTANT: The receiver host must start the `syslogd` daemon with the `-r` option (`/usr/sbin/syslogd -r`) in the `/sbin/init.d/syslog` script to enable the receiving logging messages on port `514` from other hosts.

Special for SuSE distribution 8.0 and up:
in `/etc/sysconfig/syslog` the parameter `SYSLOGD_PARAMS` should include the " `-r` " (`SYSLOGD_PARAMS="-r "`)
or use YAST (`/etc/sysconfig/` editor) and Search for `SYSLOGD`

Using `syslog-ng` as remote log collector:

Sending all logs to a log collector:

```
destination loghost { tcp("192.168.0.2" port(5140)); };
```

Receiving all logs from local and log sender:

All logs from remote hosts will go into a separate file named for each host in */var/log. $HOST is a variable that gets translated into the sending hostname.*

```
source r_src { tcp(ip("192.168.0.2") port(5140)); };
destination r_all { file("/var/log/$HOST"); };
log { source(r_src); destination(r_all); };
```

Real-time watching the content of a log file.

The following command allows to watch real-time the content of a log file.
eg. `tail -f /var/log/messages`
or `less +F /var/log/messages`
Other GUIs like `xtail` and `xlogmaster` can also do the same more elegantly.

Generating log messages from command line or scripts:

```
logger -p facility.level -i -t "MessageTitle" "Message"
```

('`*`' not allowed as facility or level here) (`-i` option for adding the process PID)

Command to show the very start of kernel mesages at boot-up:

```
dmesg | less      All kernel messages (including booting messages)
cat /proc/kmsg    ""     ""     ""          ""     ""     ""
```

Stop generation of the `-----MARK-----` Lines in the log files

(good for laptops: to stop the frequent harddisk access in idle)
start the `syslogd` with the option `-m 0`

- **The `logrotate` program**
- This program allows to save and compress the log files regularly based on their age or their size. These parameters are defined in its configuration file: `/etc/logrotate.conf` The parameters in this file will decide which files will be backed-up and according to which criteria.
(The content of this configuration file is not needed for the LPI 102 exam.)

Example of rotation and compression instructions:

```
compress
/var/log/messages {
   rotate 5     Make 5 weekly rotations of the file before deleting old ones.
   weekly       Rotate weekly.
   postrotate   Run the following script after rotating
          /sbin/killall -HUP syslogd
   endscript
}
```

Other possible log files(SuSE only):

`/var/log/boot.msg`	System hardware initialization log at bootup. It records lilo and kernel boot messages till end of default runlevel initialization.
`Ctrl-Alt-F10`	Shows the kernel modules messages.

Exampes of default configured log files:

`/var/log/messages`	All messages except mail and auth
`/var/log/faillog`	System failures log file
`/var/log/warn`	System warnings log file

Log files viewer under X-Windows (kde):

```
xterm -e "tail -f /var/log/messages"
```

`xtail`	Very good
`xlogmaster`	Very good
`kwatch`	Good. Mixed log information with log file source titles

Logrotate Programm (in German)

Mit dem `Logrotate` Programm kann man sich darum kümmern, dass alte Logfiles nach einer einstellbaren Zeit - oder wenn sie eine bestimmte Größe erreicht haben - weiterverarbeitet werden.

Dabei sind verschiedene Parameter in der **`logrotate.conf`** dafür zuständig, was mit den Dateien passieren soll. Auf den ersten Blick fallen dabei die Optionen `compress` und `nocompress` auf, die dafür zuständig sind, ob die rotierte Logdatei komprimiert wird oder nicht. Was aber, wenn man die Dateien gar nicht aufheben will? - Eine `delete` oder `remove` Option sucht man vergeblich.

Natürlich kann man die Dateien aber mit `Logrotate` auch einfach wegwerfen. Das passiert dann, wenn man bei der '`rotate`' Option als Parameter eine **0** angibt:

```
rotate 0
```

syslog-ng (new generation)

This syslog daemon allows for more control on the logged messages.
Its is included from Version 8.0 and on of SuSE distribution
Its configuration file has a very different but clear rule format. Its name is:
`/etc/syslog-ng/syslog-ng.conf`
Its Manual is in HTML format is located in :
`/usr/share/doc/packages/syslog-ng/html/`

Good extract from web site:

If you want to compile in TCP wrappers support, you can add the `--enable-tcp-wrapper` flag to the configure script. After *syslog-ng* is finished compiling, become root and run `make install`. This will install the *syslog-ng* binary and manpages. To configure the daemon, create the */usr/local/etc/syslog-ng* directory and then create a *syslog-ng.conf* to put in it. To start off with, you can use one of the sample configuration files in the *doc* directory of the *syslog-ng* distribution.

There are five types of configuration file entries for *syslog-ng*, each of which begins with a specific keyword. The `options` entry allows you to tweak the behavior of the daemon, such as how often the daemon will sync the logs to the disk, whether the daemon will create directories automatically, and hostname expansion behavior. `source` entries tell *syslog-ng* where to collect log entries from. A source can include Unix sockets, TCP or UDP sockets, files, or pipes. `destination` entries allow you to specify possible places for *syslog-ng* to send logs to. You can specify files, pipes, Unix sockets, TCP or UDP sockets, TTYs, or programs. Sources and destinations are then combined with filters (using the `filter` keyword), which let you select syslog facilities and log levels. Finally, these are all used together in a `log` entry to define precisely where the information is logged. Thus you can arbitrarily combine any source, select what syslog facilities and levels you want from it, and then route it to any destination. This is what makes *syslog-ng* an incredibly powerful and flexible tool.

To set up *syslog-ng* on the remote end so that it can replace the *syslogd* on the system and send traffic to a remote *syslog-ng*, you'll first need to translate your *syslog.conf* to equivalent `source`, `destination`, and `log` entries.

Here's the *syslog.conf* for a FreeBSD system:

```
*.err;kern.debug;auth.notice;mail.crit          /dev/console
*.notice;kern.debug;lpr.info;mail.crit;news.err /var/log/messages
security.*                                      /var/log/security
auth.info;authpriv.info                         /var/log/auth.log
mail.info                                       /var/log/maillog
lpr.info                                        /var/log/lpd-errs
cron.*                                          /var/log/cron
*.emerg                                         *
```

First you'll need to configure a source. Under FreeBSD, */dev/log* is a link to */var/run/log*. The following source entry tells *syslog-ng* to read entries from this file:

```
source src { unix-dgram("/var/run/log"); internal( ); };
```

If you were using Linux, you would specify `unix-stream` and */dev/log* like this:

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

The `internal()` entry is for messages generated by *syslog-ng* itself. Notice that you can include multiple sources in a source entry. Next, include destination entries for each of the actual log files:

```
destination console { file("/dev/console"); };
destination messages { file("/var/log/messages"); };
destination security { file("/var/log/security"); };
destination authlog { file("/var/log/auth.log"); };
destination maillog { file("/var/log/maillog"); };
destination lpd-errs { file("/var/log/lpd-errs"); };
destination cron { file("/var/log/cron"); };
destination slip { file("/var/log/slip.log"); };
destination ppp { file("/var/log/ppp.log"); };
destination allusers { usertty("*"); };
```

In addition to these destinations, you'll also want to specify one for remote logging to another *syslog-ng* process. This can be done with a line similar to this:

```
destination loghost { tcp("192.168.0.2" port(5140)); };
```

The port number can be any available TCP port.

Defining the filters is straightforward. You can simply create one for each syslog facility and log level, or you can create them according to those used in your *syslog.conf*. If you do the latter, you will only have to specify one filter in each log statement, but it will still take some work to create your filters.

Here are example filters for the syslog facilities:

```
filter f_auth { facility(auth); };
filter f_authpriv { facility(authpriv); };
filter f_console { facility(console); };
filter f_cron { facility(cron); };
filter f_daemon { facility(daemon); };
filter f_ftp { facility(ftp); };
filter f_kern { facility(kern); };
filter f_lpr { facility(lpr); };
filter f_mail { facility(mail); };
filter f_news { facility(news); };
filter f_security { facility(security); };
filter f_user { facility(user); };
filter f_uucp { facility(uucp); };
```

and examples for the log levels:

```
filter f_emerg { level(emerg); };
filter f_alert { level(alert..emerg); };
filter f_crit { level(crit..emerg); };
filter f_err { level(err..emerg); };
filter f_warning { level(warning..emerg); };
filter f_notice { level(notice..emerg); };
filter f_info { level(info..emerg); };
filter f_debug { level(debug..emerg); };
```

Now you can combine the source with the proper filter and destination within the log entries:

```
# *.err;kern.debug;auth.notice;mail.crit                /dev/console
log { source(src); filter(f_err); destination(console); };
log { source(src); filter(f_kern); filter(f_debug);
destination(console); };
log { source(src); filter(f_auth); filter(f_notice);
destination(console); };
```

```
log { source(src); filter(f_mail); filter(f_crit);
destination(console); };

# *.notice;kern.debug;lpr.info;mail.crit;news.err
/var/log/messages
log { source(src); filter(f_notice); destination(messages); };
log { source(src); filter(f_kern); filter(f_debug);
destination(messages); };
log { source(src); filter(f_lpr); filter(f_info);
destination(messages); };
log { source(src); filter(f_mail); filter(f_crit);
destination(messages); };
log { source(src); filter(f_news); filter(f_err);
destination(messages); };

# security.*
/var/log/security
log { source(src); filter(f_security); destination(security); };

# auth.info;authpriv.info                        /var/log/auth.log
log { source(src); filter(f_auth); filter(f_info);
destination(authlog); };
log { source(src); filter(f_authpriv); filter(f_info);
destination(authlog); };

# mail.info
/var/log/maillog
log { source(src); filter(f_mail); filter(f_info);
destination(maillog); };

# lpr.info                                             /var/log/lpd-
errs
log { source(src); filter(f_lpr); filter(f_info); destination(lpd-
errs); };

# cron.*                                               /var/log/cron
log { source(src); filter(f_cron); destination(cron); };

# *.emerg                                              *
log { source(src); filter(f_emerg); destination(allusers); };
```

You can set up the machine that will be receiving the logs in much the same way as if you were replacing the currently used *syslogd*.

To configure *syslog-ng* to receive messages from a remote host, you must specify a source entry:

```
source r_src { tcp(ip("192.168.0.2") port(5140)); };
```

Alternatively, you can dump all the logs from the remote machines into the same destinations that you use for your local log entries. This is not really recommended, because it can be a nightmare to manage, but can be done by including multiple source drivers in the source entry that you use for your local logs:

```
source src {
    unix-dgram("/var/run/log");
    tcp(ip("192.168.0.2") port(5140));
    internal( );
};
```

Now logs gathered from remote hosts will appear in any of the destinations that were combined with this source.

If you would like all logs from remote hosts to go into a separate file named for each host in */var/log*, you could use a destination like this:

```
destination r_all { file("/var/log/$HOST"); };
```

syslog-ng will expand the `$HOST` macro to the hostname of the system sending it logs and create a file named after it in */var/log*. An appropriate log entry to use with this would be:

```
log { source(r_src); destination(r_all); };
```

However, an even better method is to recreate all of the remote *syslog-ng* log files on your central log server. For instance, a destination for a remote machine's messages file would look like this:

```
destination fbsd_messages { file("/var/log/$HOST/messages"); };
```

Notice here that the `$HOST` macro is used in place of a directory name. If you are using a destination entry like this, be sure to create the directory beforehand, or use the `create_dirs()` option:

```
options { create_dirs(yes); };
```

syslog-ng's macros are a very powerful feature. For instance, if you wanted to separate logs by hostname and day, you could use a destination like this:

```
destination fbsd_messages {
    file("/var/log/$HOST/$YEAR.$MONTH.$DAY/messages");
};
```

You can combine the remote source with the appropriate destinations for the logs coming in from the network just as you did when configuring *syslog-ng* for local logging—just specify the remote source with the proper destination and filters.

Another neat thing you can do with *syslog-ng* is collect logs from a number of remote hosts and then send all of those to yet another *syslog-ng* daemon. You can do this by combining a remote source and a remote destination with a log entry:

```
log { source(r_src); destination(loghost); };
```

Since *syslog-ng* is now using TCP ports, you can use any encrypting tunnel you like to secure the traffic between your *syslog-ng* daemons. You can use SSH port forwarding or stunnel to create an encrypted channel between each of your servers. By limiting connections on the listening port to include only localhost (using firewall rules, as in or), you can eliminate the possibility of bogus log entries or denial-of-service attacks.

Server logs are among the most critical information that a system administrator needs to do her job. Using new tools and strong encryption, you can keep your valuable log data safe from prying eyes.