Exim - Kurzer Konfigurationsüberblick

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
v1.2.1-3-a9bf9198340a	2015-04-22		HSH

Contents

Die Quelle dieses Dokuments ist: https://ssl.schlittermann.de/hg/doc/exim, es gibt auch eine HTML und eine PDF Version.

1 Konfiguration

1.1 File

- Das Konfigurationsfile wird in der Build-Konfiguration (*EDITME*) festgelegt.
- Es ist auch eine Liste von Konfigurationsfiles möglich, der erste Treffer gewinnt.

Das verwendete File läßt sich ermitteln mit:

```
exim -bP configure_file
```

Das aktive Konfigurationsfile kann auch auf der Kommandozeile festgelegt werden. Für einige Operationen muss der Pfad ein absoluter sein. Auch */dev/null* ist eine gültige Konfiguration.

```
exim -C <Pfadname> ...
```

Die Konfiguration wird öfter gelesen als vielleicht erwartet:

- Start des Daemon
- Signal *HUP*
- re-exec for verschiedene Zwecke

Caution
Inhalt der Datei *exim.conf* und eventuell über opt:.include eingelesener Files ist statisch. Dynamisch sind dann Expansionen, Lookups.

1.2 Syntax

Der Parser der Konfigurationsdatei ist ziemlich einfach.

1. Kommentarzeilen werden entfernt
2. Führende und anhängige Leerzeichen werden entfernt
3. Forsetzungszeilen („\“ am Zeilenende) werden zusammengefasst
4. einfache Syntax-Checks

Die Konfiguration besteht aus mehreren Abschnitten. Nicht alle Abschnitte sind identisch aufgebaut.

Globale Optionen
: Allgemeine Parameter wir Listendefinitionen, Optionen für den Mailempfang, Datenbank-Verbindungsparameter

Access Control Lists
: Kontrolle des Nachrichten-Empfangs

Router
: Steuerung des Nachrichten-Routings (Ermittlung des Zielhosts und des Transportmechanismus)

Transports
Konfiguration der Transportmechanismen (SMTP, LMTP, o.a.)

Rewrite
Umschreiben von Envelope und Kopfzeilen

Retry
Regeln für Wiederholungsversuche bei der Zustellung

Authentifizierung
Parameter für die SMTP-Authentifizierung (sowohl Client als auch Server)

2 Syntax der Konfiguration

2.1 Kommentare

- Alle Zeilen, die mit einem „#“ beginnen (nach optionalem Whitespace), sind Kommentarzeilen.
- An anderen Stellen der Konfiguration hat das „#“-Zeichen keine besondere Bedeutung.

2.2 Macros

- Macros werden generell groß geschrieben.
- Macros sind ein einfache statischer Textersetzungsmechanismus (im Sinne
- globaler „Konstanten“) und für das bedingte Parsen von Konfigurationsteilen.
- Macros können in der Konfiguration definiert werden, aber auch auf der Kommandozeile. (Auf der Kommandozeile gibt es jedoch wegen eventueller Sicherheitsimplikationen einige Besonderheiten zu beachten.)

Beispiel

```
LDAP_BASE = ou=ousers,o=acme
LDAP = ldap:///LDAPBASE
...
data = ${lookup ldap{LDAP?mail?sub?uid=${quote_ldap:$local_part}}}
...
.ifdef TESTING
testrouter:
   driver = dnslookup
   transport = TRANSPORT
   ...
.endif
```

Und beim Aufruf:

```
exim -DTESTING -DTRANSPORT=foo ...
```

Macros können neu definiert werden („==“ statt „=“). Macros von der Kommandozeile haben Vorrang.

2.3 Optionen

- Optionen werden generell klein geschrieben.
- Es gibt ca. 250 globale Optionen, hinzu kommen noch weitere Optionen für Router, Transports, Authenticators, usw.
- Optionen haben einen bestimmten Datentyp.

Beispiel

```
primary_hostname = foo.example.com
```

2.4 Listen

Listen bestehen aus der Angabe des Listentypes, eines Namens und dem Listeninhalt. Es gibt 4 verschiedenen Listentypen.

Domains
Liste von Domain-Namen. Es gibt einige spezielle Tokens, die in einer solchen Liste auftauchen dürfen.

```
domainlist local_domains = @ : schlittermann.de : @mx_primary
```

Hosts
Hosts oder IP-Adressen, z.B. für vertrauenswürdige Absender-Netze

```
hostlist trusted_hosts = 192.168.0.0/24 : 127.0.0.1
```

Mail-Adressen
Mailadressen

```
addresslist blocked_senders = ad@bestholiday.de : *@mailgun.com
```

Local Parts
Local Parts von Mail-Adressen, z.B. von ACL-Checks ausgenommene lokale Empfänger

```
localpartlist rfc = postmaster : abuse
```

Listen-Lookups werden nur gecacht, wenn die Liste keine Expansions-Items entält (also kein *$*). Um ein Caching zu erwzingen, kann dem Listentyp ein `_cache` nachgesetzt werden:

```
domainlist_cache local_domain = ${lookup ...}
```

Beispiel Debugging

```
exim -bP +local_domains
```

2.5 Instanzen von Routern, Transports, Authentificators

- Router, Transports und Authenificators werden in einzelnen „Funktionsblöcken“ definiert.
- Diese Blöcke haben selbstgewählte Namen und eine Liste von Optionen, die das Verhalten des jeweiligen Routers, Transports oder Authenticators bestimmen.

Beispiel

```
begin transports
...
remote_smtp:
   driver = smtp
   command_timeout = 20s
```

2.6 ACL, Retry- und Rewrite-Regeln

Diese Teile der Konfiguration haben ihre jeweils eigene Syntax.

3 Werte und ihre Typen

ALle Optionen der Konfigurationsdatei haben einen spezifischen Daten-Typ.

- *bool*, *integer*, *fixed-point*, *time* und *string*
- Listen sind immer String-Listen
- Werte, die der Expansion unterliegen, sind immer vom Typ *string*

3.1 Bool

Werte vom Typ *bool* sind einfache Schalter.

Format

```
<option> = true|yes
<option> = false|no
<option>
no_<option>
not_<option>
```

Beispiel

```
split_spool_directory
not_split_spool_directory
split_spool_directory = true
split_spool_directory = no
```

3.2 Integer

Zahlen. Ganze Zahlen.

Format

```
... = <digits>[<suffix>]
      = 0<oct digts>[<suffix>]
      = 0x<hex digts>[<suffix>]
```

Erlaubte Suffixe sind *K* (2^{10}) und *M* (2^{20}).

Beipiel

```
check_spool_space = 10M
```

3.3 Fixed-Point

Fixkomma-Zahlen, maximal 3 Nachkommastellen

Format

```
... = <digits>[.<max 3 digits>]
```

Diese Werte erlauben **keine** Einheitensuffixe.

Beispiel

```
deliver_queue_load_max = 2.5
```

3.4 Zeitintervalle

Zeitintervalle sind Zeitintervalle.

Format:

```
... = <digits><suffix>[<digits><suffix>]...
```

Erlaubte Suffixe sind *s*, *m*, *h*, *d*, *w*, also Intervalle, die sich zweifelsfrei in Sekunden umrechnen lassen.

Beispiel

```
auto_thaw = 3d12h
```

Caution
Bei der Ausgabe von Intervall-Optionen werden die Zeiten normalisiert.

3.5 Zeichenketten

- Zeichenketten gibt es in zwei Formen: Literale Zeichenketten und gequotete Zeichenketten..
- Zeichenketten, die später noch expandiert werden, werden *immer* als gequotete Zeichenketten behandelt.

Format

```
... = <string>
... = "<string>"
```

Die gequotete Form ist notwendig, wenn die Zeichenkette mit Leerzeichen enden oder beginnen soll, oder wenn eine der folgenden Sequenzen interpretiert werden soll: \\, *n*, *r*, *t*, *ddd* (oktaler Zeichenwert), *xdd* (hexadezimaler Zeichenwert).

Beispiel

```
bounce_message_file = /etc/exim/bounce-message.txt
bounce_message_text = "Sorry.\nI'm really sorry.\n"
message = User $local_part does not exist.
```

Caution
Bei der Ausgabe von String-Optionen werden eventuell vorhandene Steuerzeichen durch korrenspondierende Escape-Sequenzen ersetzt.

3.6 Listen

- Listen sind immer zuerst eine *einzige* Zeichenkette, unterliegen also den Zeichenketten-Interpretationsregeln.
- Der Feldtrenner ist standardmäßig ein ∴. Wird der Feldtrenner als Teil eines Listenelements benötigt, muss er verdoppelt werden (oder es wird ein anderer Trenner gewält.)
- Umschließender Leerraum an den einzelnen Elementen wird ignoriert.
- Leere Elemente in der Liste und am *Ende* der Liste werden ignoriert.

Format

```
... = <item> : <item> ...
... = <item> : <i::tem> ...
... = <<separator> <item> ; <item> ...
```

Beispiel

```
local_interfaces = 127.0.0.1 : ::::1
local_interfaces = <; 127.0.0.1 ; ::1
domains = <\n ${lookup mysql{...}}
senders =   ❶
senders = : ❷
```

❶ leere Liste

❷ Liste mit einem leeren Element

Beispiel Debugging

```
exim -be '${map{:a:b:c,}{<$item>}}'
exim -be '${map{<,,a,b,c,}{<$item>}}'
exim -C <(echo domainlist local_domains = 'a:b::c:d') -be '${listnamed: ↩
    local_domains}'
exim -bP +local_domains
```

3.7 Reguläre Ausdrücke

- Reguläre Ausdrücke werden durch *libpcre* ausgewertet, sie sind also kompatibel mit den von Perl bekannten regulären Ausdrücken.
- Reguläre Ausdrücke werden immer zuerst als String evaluiert, eventuelle Backslash-Sequenzen müssen also ggf. geschützt werden.

Format

```
... = ^<regexp>
```

Beispiel

```
domains = example.com : ^\\d{3}
domains = example.com : \N^\d{3}\N
```

Im Beispiel wird der Backslash verdoppelt, da die `domains` Option zuerst expandiert wird.

4 Expansion von Zeichenketten

- Viele Stellen der Konfiguration verwenden String-Expansion (Typ *string** im *spec.txt*). String-Expansion wird von *$* getriggert. Das Ergebnis ist dann ein neuer String oder *forced failure*.
- Die Expansion der Strings erfolgt in der Regel erst im Bedarfsfall („late binding“).

Caution
Es stehen aber nicht in jeder Phase alle Expansions-Items zur Verfügung. (z.B. wird der Wert der globalen Option `message_size_limit` expandiert, aber bereits während der Antwort auf das SMTP-HELO. In dieser Phase gibt es noch keinen *`$local_part`*).

Debugging

```
exim -be [<string>]
exim -bem <message-file> [<string>]
exim -be -Mset <spool-id> [<string>]
```

Beispiel Debugging

```
exim -be '$primary_hostname'
exim -be '$tod_full'
exim -bem <(echo 'Subject: Hi') '$h_subject:'
```

4.1 Expansions-Items

- Aus Sicht des Expanders ist alles, was mit einem *$* beginnt, ein Expansions-Item.

Variablen
Das sind operative Parameter, Information über die aktuelle Nachricht, den aktuellen Systemzustand, Rückgabewerte. spec

```
$local_part
${local_part}
```

Operatoren
Einfache Funktionen wie die Transformation einer Zeichenkette von Klein- auf Großbuchstaben. spec

```
${hex2b64:<string>}
${length_3:<string>}
```

Funktionen
Komplexere Umwandlungen mit mehreren Eingangsparametern spec

```
${readsocket{<socket>}{<request>}}
${length{<string>}{<string>}}
```

Bedingungen
Fluss-Steuerung spec

```
${if <condition>...}
```

Lookups
Informationsgewinnung aus externen Quellen (Files, Datenbanken) spec

```
${lookup{<item>}lsearch{<file>}}
lsearch;<file>
```

5 Lookups

- Mit Lookups werden Daten aus externen Quellen gewonnen.
- Lookups gibt es in zwei Syntaxvarianten und in zwei Stilen.

Syntax
Es gibt implizite und explizite Loookups.

Stil
Es gibt Single-Key und Query-Style Lookups.

5.1 Syntax

5.1.1 Explizite Syntax

- Der Key oder die Frage wird ausdrücklich formuliert, es ist eine gewöhnliche String-Expansion. Als Resultat wird das Ergebnis des Lookups zurückgeliefert.
- Analogie: `value(<$key>)`

Format

```
... = ${lookup{<key>}<type>{<file>}}
... = ${lookup <type>{<query>}}
```

Beispiel

```
domainlist local_domains = ${lookup{$domain}dsearch{/etc/vmail/domains}} ❶
localpartlist users = ${lookup ldap{ldap:///o=foo?uid?sub?uid=${quote_ldap: ↩
    $local_part}}}
data = ${lookup{$local_part}lsearch{/etc/aliases}{$value}{root@localhost}}
data = ${lookup{$local_part}lsearch{/etc/aliases}{$value}\
       {${lookup{...}}}}
```

❶ Im Beispiel wird der Umstand genutzt, dass in vielen Fällen lediglich die aktuell behandelte Adresse in der Liste vorhanden sein muss.

Die explizite Syntax kann natürlich auch eine komplette Liste zurückliefern, diese muss dann eventuell massiert werden, damit die Feldtrenner den Erwartungen von Exim entsprechen.

5.1.2 Implizite Syntax

- Der Key ergibt sich aus dem Kontext, das Lookup wird dennoch durch den String-Expander behandelt. Wenn das Lookup erfolgreich ist, wird als Resultat der Key zurückgeliefert, nicht ein eventuell vorhandener Wert!
- Analogie: `exists($key) ?$key :"`

Format

```
... = <type>;<file>
... = <type>;<query>
```

Beispiel

```
domains = lsearch;/etc/vmail/domainlist
domains = mysql;SELECT COUNT(*) FROM domains WHERE domain = ${quote_mysql:$domain}
```

5.2 Style

5.2.1 Single Key

Single-Key Lookups beziehen sich auf einfach (und meist schnelle) Key/Value Daten. Das sind Dateien im Format der */etc/aliases*, das sind Verzeichnisse und DBM-Files. spec

WICHTIGE SINGLE KEY LOOKUPS

lsearch
: Lineare Suche in einer Datei. Suche nach einem Default-Wert mit *lsearch**, Suche nach Teilzeichenketten (Domain) mit *partial-lsearch*.

dsearch
: Verzeichnis-Lookup: Suche nach einem spezifischen Verzeichniseintrag

iplsearch
: Suche nach IP-Adressen oder Netzen unter Beachtung von Netzmasken

dbm
: Suche in Berkeley-DBM Files

Beispiel

```
data = ${lookup{$local_part}lsearch*{/etc/aliases}}
```

5.2.2 Query Style

Komplexe Anfragen im Sinne einer „Query“. spec

WICHTIGE QUERY-STYLE LOOKUPS

mysql
: Eben MySQL...

```
${lookup mysql{SELECT ... WHERE ... like ${quote_mysql:$local_part}}
```

dnsdb
: Pseudo-Queries

```
${lookup dnsdb{a=$sender_host}}
dnsdb;mxh=example.com
```

ldap
: Anfragen an LDAP. Erwartet wird ein Object (*ldap*) oder mehrere Objekte (*ldapm*).

```
${lookup ldap{ldap:///ou=foo?mail?sub?uid=${quote_ldap:$local_part}}}
```

5.3 Überblick

Es gibt also vier Möglichkeiten, Daten aus externen Quellen zu gewinnen.

	single key	**query style**
explizit	`${lookup{<key>}<type>{<file>}`	`${lookup <type> {<query>}}`
implizit	`<type>;<file>`	`<type>;<query>`

6 Acess Control Lists

ACL wirken für alle Phasen der SMTP-Session. Im globalen Teil werden die Einstiegspunkte mit `acl_smtp_connect` usw. definiert. Im ACL-Abschnitt der Konfiguration muss dann ein entsprechend genannter Block existieren.

Die häufigsten Einstiegspunkte sind

`acl_smtp_rcpt`
: Überprüfung beim `RCPT TO` Kommando.

`acl_smtp_data`
: Überprüfung am Ende der `DATA` Phase, also Content-Scan.

Eine ACL ist eine Folge von Bedingungsblöcken. Sind *alle* Bedingungen eines Blocks erfüllt, gilt die für diesen Block festgelegte Aktion. Die Bedingungen werden in der Reihenfolge ihres Auftretens bearbeitet. Sobald eine Bedingung nicht zutrifft, wird der nächste Block untersucht.

Beispiel

```
accept  domains = +local_domains
        local_parts = postmaster : abuse

accept  domains = +relay_to_domains
        verify = recipient/callout=use_sender

deny    message = sorry
```

MÖGLICHE ACL-AKTIONEN

accept
: Alles ist ok. Es geht zur nächsten Phase der Verbindung.

deny
: Fehler 5xx wird zurückgeliefert (nicht zwingend ein Ende der Verbindung!).

defer
: Fehler 4xx wird zurückgeliefert.

drop
: Fehler 5xx wird generiert und Verbindung wird abgebrochen.

warn
: Keine finale Entscheidung, NOOP.

discard
: Wie *accept*, aber Empfänger, bzw. Nachricht wird verworfen!

require
: Wenn eine der Bedingungen **nicht** erfüllt ist, Abbruch mit 5xx, andernfalls weiter zum nächsten Block. U.U. keine finale Entscheidung.

HÄUFIGE ACL-BEDINGUNGEN

`hosts`
: Sender-Host-IP wird geprüft

```
hosts = +trusted_hosts
```

domains
Empfänger-Domain wird geprüft

```
domains = +local_domains
```

senders
Absender-Mailadresse wird geprüft

```
senders = postmaster@example.com : hans@foo.bar
```

malware
Malware-Content-Scan liefert einen Treffer

```
malware = *
```

spam
SPAM-Content-Scan (*SpamAssassin*)

```
spam = nobody/true
```

verify
Empfänger- oder Absenderüberprüfung, eventuell auch mit *callout*.

```
verify = recipient/callout=use_sender,defer_ok
```

ratelimit
Limitierungen aller Art (Menge, Größe, . . .)

```
ratelimit = 10/1h/$sender_address
```

condition
Sonstige Bedingungen aller Art

```
condition = ${run{perl}{graylist}{$sender_address/$local_part@$domain}}
```

ACL-MODIFIER

message
Der Nachrichtenpuffer wird mit einem Text gefüllt.

```
message = 550 Sorry
```

log_message
Der Nachritenpuffer für das Protokoll wird mit einem Text gefüllt. Sonst identisch zu `message`.

logwrite
Sofort einen Log-Eintrag

control
Steuert das weitere Verhalten der Nachrichtenverarbeitung

```
control = submission
```

set
ACL-Variablen setzen

```
set acl_m_domain = $domain
set acl_c_host = $sender_host_address
```

add_header
Header zum Hinzufügen vormerken

remove_header
Header zum Entfernen vormerken

delay
Fügt eine kurze Verarbeitungspause ein

Ziel der ACL-Operations sollte es sein, möglichst viele Dinge zur SMTP-Zeit zu entscheiden und dann die SMTP-Verbindung mit 5xx zu beenden. Damit bleibt die Verantwortung für die Bounce beim Absender.

7 Router

Im Router-Abschnitt wird durch einzelne Blöcke das Routing-Verhalten gesteuert. Die Reihenfolge der Blöcke ist wichtig. Das Verhalten eines jeden Blocks wird durch den Treiber bestimmt.

Routing als Pseudoprogramm

```
if (check preconditions == FAIL) goto NEXT ROUTER

switch route($address)
   case ACCEPT:      schedule $address for transport
   case NEW ADDRESS: goto FIRST ROUTER
   case PASS:        goto NEXT ROUTER
   case FAIL:        generate bounce
   case DEFER:       back to queue
   case DECLINE:     if (more?) goto NEXT ROUTER
                     generate bounce
```

Beispiel

```
external:
   driver = dnslookup
   domains = !+local_domains
   transport = remote_smtp
   ignore_target_hosts = <; 127.0.0.1 : ::1
```

Eine Liste der konfigurierten Router erhält man mit:

```
exim -bP router_list
```

8 Transports

Die Transport-Blöcke werden von den Routern referenziert, ihre Reihenfolge ist egal. Auch hier wird das grundlegende Verhalten eins Blocks durch den Treiber bestimmt.

Beispiel

```
remote_smtp
  driver = smtp
  command_timeout = 10s
```

Die Liste der konfigurierten Transports:

```
exim -bP transport_list
```

9 Filter

Es gibt das *System*-Filter und *User*-Filter. Das System-Filter wird **vor** dem Routing der Nachricht aktiv. Das User-Filter dann, wenn ein Router für diesen Nutzer aktiv wird und das User-Filter auswertet.

Wenn eine Nachricht nicht zugestellt werden kann, dann wird bei einem späteren Queue-Run der Filter-Prozess wiederholt! (Es gibt aber einen `first_delivery` Test, mit dem man das behandeln kann.)

System-Filter verwenden immer Exim-Filter-Syntax. User-Filter können Exim-Filter-Syntax verwenden, oder eine Teilmenge der Sieve-Filter-Syntax.

Dem Filter steht die Nachricht schon zur Verfügung, es kann also auf Variablen aus den ACL und auf die Header der Nachricht zugreifen..

9.1 Exim-Filter-Syntax

- 1. Zeile

```
# Exim filter
```

- Keyword/Value, separiert durch Whitespace oder an einigen Stellen durch *()*
- Zeilenumbrüche haben keine Sonderbedeutung
- Kommentare von <*separator*># bis zum Zeilenende
- Data-Values
  - verbatim, wenn sie keine Separatoren enthalten
  - quoted mit *"*
    - Leerzeichen und die Zeichen *n*, *r*, *t*,
    - *ddd* oder \x*dd*
    - alle anderen Backslash-Squenzen werden daruch das dem Backslash folgende Zeichen ersetzt
  - Strings-Expansion wie in der Exim-Konfiguration
  - max 1024 Zeichen **vor** der Expansion

9.2 Filter-Kommandos

`add`
: Nutzer-Variable inkrementieren. Diese Variablen stehen dann später als *`$sn1`* - *`$sn9`* in den Routern und Transports zur Verfügung

```
add 27 to to n3
```

`deliver`
: Mail weiterleiten, Vorsicht mit den Absendern (SPF). Signifikant.

```
deliver "Fred <fred@example.com>"
```

`fail`
: Mail abweisen

`finish`
: Keine weitere Bearbeitung

freeze
Mail einfrieren

headers
Header-Verarbeitung konfigurieren (Zeichensatz)

```
headers charset UTF-8
```

if
Bedingte Verarbeitung mit `if`, `then`, `elif`, `else`, `endif`.

- String-Conditions sind `begins`, `contains`, `ends`, `matches`, bzw. `does not ...`.
- Numerische Bedingungen sind `is above`, `is below` bzw. `is not ...`
- Andere Bedingunge: `is_delivered`, `error_message`, `personal`, `first_delivery`
- Listen: `forany`

logfile
Logfile definieren

logwrite
Schreiben ins Logfile

mail
Antwort-Nachricht erzeugen

```
mail text "Danke für Ihre Nachricht mit $h_subject:"
```

pipe
Nachricht per Pipeline weiterverarbeiten. Signifikant.

```
pipe "/usr/bin/mailstat"
```

save
Nachricht in File/Mailbox sichern, das Format der Sicherung hängt vom `filter_file_transport` in der Konfiguration ab. Signifikant.

```
save bcc
```

testprint
Testausgabe im Filter-Test-Modus

vacation
Urlaubsnachricht erzeugen. Sonderform von `mail`.

10 Operating

Exim legt alle wichtigen Daten in Text-Dateien ab. Trotzdem sind einige Optionen und Tools hilfreich im Umgang mit den operativen Daten des Exim.

10.1 Prozesse

Mit

```
exiwhat
```

erhält man eine gute Übersicht über die aktuellen Aktivitäten des Exim.

10.2 Queue

Alle operativen Daten liegen unter dem Spool-Directory:

```
exim -bP spool_directory
```

input
: Message-Spool. Je Message 2 Dateien.

db
: Hint-Files, Cache.

msglog
: Nachrichten-Logs, solange die Nachricht noch nicht *completed* ist.

Mit dem klassischen *mailq* lässt sich die aktuelle Queue auflisten. Oder auch mit

```
exim -bp
exim -bpc
```

Besser ist *exipick*, das verfügt über viele Möglichkeiten, Queue-Inhalte zu selektieren:

```
exipick
exipick -c
exipick -zi
exipick -f foo
```

Einzelne Nachrichten untersuchen

```
exim -Mvh <queue-id>
exim -Mvb <queue-id>
exim -Mvc <queue-id>
```

Löschen von Nachrichten

```
exim -Mrm <queue-id>
```

10.3 Protokolle

Protokolle liegen in

```
exim -bP log_file_path
```

mainlog
: Logfile über alle Transaktionen, dokumentiertes und maschinenlesbares Format

rejectlog
: Etwas mehr Details bei 5xx/4xx (Headerzeilen)

paniclog
: Wenn nichts mehr geht.

Zum Suchen und Zusammenfassen dieser Informationen eignet sich *exigrep* am besten.

Beispiel

```
exigrep 'hannes@example.com' /var/log/exim/mainlog
exigrep 'fred@foobar.de' $(ls -tr /var/log/exim/mainlog*)
```

Solange die Nachricht noch in der Queue ist:

```
exim -Mvl <spool-id>
```

10.4 Hint-Data

Die Hint-Data-Files sind Berkeley-DB-Files. Die dafür vorhandenen Werkzeuge sind eher spartanisch. Im Zweifelsfall können diese Files auch gelöscht werden.

exim_dumpdb
Textdump der jeweiligen Datenbank, Ausgabeformat ist je nach DB-Format leicht unterschiedlich

Beispiel

```
exim_dumpdb /var/spool/exim retry
```

exim_fixdb
Mit diesem Werkzeug können gezielt einzelne Datensätze einer Hint-DB gelöscht werden

Beispiel

```
# exim_fixdb /var/spool/exim4 retry
Modifying Exim hints database /var/spool/exim4/db/retry
> T:mx1.bmw.c3s2.iphmx.com:2620:101:200a:d100::e
13-Apr-2015 08:16:36
0 error number: 101 Network is unreachable
1 extra data:   77
2 first failed: 12-Apr-2015 18:09:35
3 last try:     13-Apr-2015 08:16:36
4 next try:     13-Apr-2015 10:16:36
5 expired:      no
> d
deleted
```

exim_tidydb
Wird regelmäßig über Cron aufgerufen um die Inhalte der Datenbankfiles aufzuräumen. Die Files werden dabei nicht zwangsläufig kleiner!

11 Debugging

11.1 Konfiguration

Die syntaktische Korrektheit der Konfiguration wird mit

```
exim -bV
```

geprüft. In Routern und Transports kann die `debug_print` Option verwendet werden, um während der Test-Sessions noch angepasste Ausgaben zu erhalten.

Caution
Viele Fehler treten erst später bei der Expansion auf!

Teile der Konfiguration können mit:

```
exim -bP <option>
exim -bP
exim -bP transports
exim -bP routers
```

ausgelesen werden.

11.2 Routing im Trockentest

Das nützlichste Werkzeug ist sicher der Routing-Trockentest. Mit

```
exim -bt <address>
```

wird ein Address-Test gemacht. Das entspricht dem Routing-Vorgang. Im Unterschied zum echten Routing-Vorgang stehen diesem Test aber keine Header-Daten zur Verfügung. Die Absender-Adresse kann mit *-f <sender>* eingesetzt werden:

```
exim -f <sender> -bt <address>
```

Mit

```
exim -bv <address>
```

findet eine Adress-Verification statt. Das ist normalerweise identisch zum Adresstest. Aber Router können so konfiguriert sein, dass sie nur zum Adress-Test oder nur zur Verification verwendet werden. Adress-Verification findet u.a. über das ACL-Statement `verify =recipient` statt.

Beispiel

```
exim -bt postmaster@schlittermann.de  ❶
exim -bts postmaster@schlittermann.de ❷
```

❶ Recipient-Test

❷ Sender-Test

11.3 Fake-Delivery für Filter/Router/Rewriting

Bei Fake-Deliveries werden alle Schritte vom Routing über Filter und Rewriting ausgeführt, aber es findet keine Delivery statt.

Beispiel

```
exim -N 1Yk9s3-00048G-4a
exim -N <message.eml
```

11.4 Fake-SMTP-Session zum ACL-Test

Exim kann SMTP-Sessions zum Test über STDIN/STDOUT abwickeln.

```
exim -bh <sender-ip>
exim -bhc <sender-ip> ❶
```

❶ Callouts werden durchgeführt

Dabei schaltet er automatisch in den Debug-Modus für ACL.

Beispiel

```
swaks -f hs@schlittermann.de -t fred@example.com \
      --pipe 'exim -bh 1.2.3.4'
```

11.5 Filter-Tests

Mit `exim -bf` bzw. `exim -bF` lassen sich die Filter-Scripte debuggen.

Beispiel

```
exim -f sender@example.com .de -bf ./filter <message
exim -bfd <domain> -bfl <local_part> ... -bf
```

11.6 Rewrite-Tests

Wenn Rewrite-Regeln verwendet werden, dann können diese mit

```
exim -brw <address>
```

getestet werden.

11.7 String-Expansion

Der String-Expander steht auf der Kommandozeile zur Verfügung:

```
exim -be <string>
```

Beispiel

```
exim -be '${lookup{root}lsearch{/etc/aliases}}'
```