# E-MAIL SERVICE



## Index

# BIBLIOGRAPHY


- Basic:
  - "TCP/IP Protocol Suite", 3a Ed. B. Forouzan, McGraw-Hill, 2003. (Chapter 20).
  - "TCP/IP Tutorial and Technical Overview". IBM Redbook. 2006. (Chapter 15).
- Advanced:
  - "Computer Networking: A Top-Down Approach Featuring the Internet", 3a Ed. J.F. Kurose and K. W. Ross. Addison-Wesley. 2005. (Chapter 2).
  - "Internet e-mail-protocols, standards, and implementation". L. Hughes. Artech House Publishers, 1998.
- RFCs:
  - RFC 821: Simple Mail Transfer Protocol.
  - RFC 822: Standard for the Format of ARPA Internet Text Messages.
  - RFC 1939: Post Office Protocol - Version 3.
  - RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One.
  - RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two.
  - RFC 3501: Internet Message Access Protocol - Version 4rev1

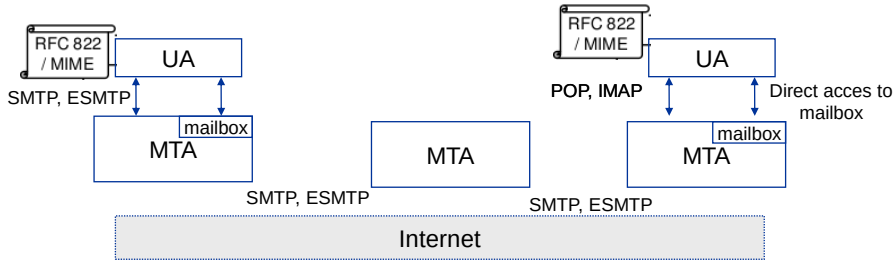



# Introduction

- Exchange of messages among users
  - Very popular application
  - Increase of productivity 30 % (estimation from '92)
- History:
  - Since more that 20 years
    - Email extension in multiuser systems
  - At the beginning it was file transfer, with an initial line added with the destination address
    - Limitations: Uncomfortable to send the same email to several recipients; files had no internal structure, impossible to send a mixture of text, voice, video; no confirmation of receipt; poor user interfaces
  - 1982: ARPANET email proposal
    - RFC 822: message format
    - RFC 821: transmission protocol
  - 1984: CCITT X.400 draft
    - 1988: modifications towards MOTIS (OSI)
  - 1990: RFC 822/821 "wins" X.400 (disappeared)
  - Responsible of the first Internet "boom"
    - (Cáceres 1991): 50% TCP connections due to email
    - Today, 10% traffic is email
  - 90s: improvements, extensions, new protocols
  - Today: spam crises?
  - We will concentrate on Internet email (RFC 821, 822)

# Architecture

- Two subsystems
  - User Agent (UA)
    - Sends and receives user messages: mail, mutt, MS Outlook,Thunderbird...
  - Message Transfer Agent (MTA)
    - Transports messages across machines: sendmail, exim, postfix, MS Exchange...
- Different standards:
  - Message format (RFC 822, MIME).
  - Destination server (DNS MX).
  - Message transfer (SMTP, ESMTP)
  - Final delivery (POP3, IMAPv4).






# Envelopes, headers, and Body

- Electronic mail is composed of three parts:
  - Envelope (RFC 821)
    - Used by MTAs for delivery
    - Specified by SMTP commands
  - Headers (RFC 822 and MIME)
    - Used by UAs
    - MTAs may add trace headers
  - Body (RFC 822 and MIME)
    - What the user writes, attaches, etc.

# Message format: RFC 822

- RFC 822: “Standard format for text messages in Internet”; i.e., ASCII
  - 1982
  - Exchange format (can be stored in a different format in queues and mailboxes)
- Structure:
  - Headers
    - format: header_name: header_value<CR><LF>
    - All characters are NVT ASCII
    - In principle each header in one line, though they may be fold
      - Replace whitespace by <CR><LF>+whitespace
    - Order unspecified
    - Generated by UA, or added by traversed MTAs
    - Only interpreted at the destination UA (not MTAs!)
  - White line (<CR><LF><CR><LF>)
  - Message body
    - NVT ASCII, only text
      - Later MIME extension allowed other content types
      - No compression
      - Maximum of 1000 chars per line (limit imposed by DATA command in RFC 821)



Delivered-To: pepe@acme.com

Received: by 10.42.246.9 with SMTP id lw9csp157724icb; Mon, 5 Nov 2012 04:34:19 -0800 (PST)

Received: by 10.180.7.197 with SMTP id l5mr13139065wia.13.1352118858931; Mon, 05 Nov 2012 04:34:18 -0800 (PST)

Return-Path: <mlperez@gerund.es>

Message-ID: <5097B248.4000503@gerund.es>

Date: Mon, 05 Nov 2012 13:34:16 +0100

From: =?ISO-8859-1?Q?Marisa_P=E9rez?= <mlperez@gerund.es>

User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:16.0) Gecko/20121026 Thunderbird/16.0.2

Reply-To: <mlperez@ieee.org>

To: pepe@acme.com

Subject: mensaje de prueba

X-Mailer: Mozilla 4.51 [es] (Win98; U)

This is a test

With several lines

# Some headers in RFC822

- Trace fields (added by traversed MTA):
  - Received: ["from" sending host] [by receiving host] *("with" mail protocol)["id" msg-id] ["for" address] ";" date-time received
- Reference fields (origin MTA):
  - Message-ID: <Unique message id>
- Between UAs:
  - Date: date and time when message was sent
  - From: source address
    - There may be several addresses if a Sender header is present
  - Rply-To: address for replies (if not present, From header will be used)
  - To: Cc: message destinations (one or several addresses separated by commas)
  - Bcc: blind list of destinations
  - Subject
  - Users, Uas, or MTAs, may invent their own experimental headers (X-...)
  - Adress formats may be
    - address@domain
    - <address@domain>
    - Name FamilyName <address@domain>
    - "Name FamilyName" address@domain



# Limitations of RFC822

- Message body limited to NVT ASCII
  - Most languages unsupported: latin and non-latin...
  - Other media unsupported: images, sound, video
- Line size limited to 1000 chars
- Limitations imposed by implementations:
  - Total size limited
    - How to fragment/split a message?
  - Whitespace removal at the end of the message
  - Lines > 76 chars folded
  - <CR> replaced by <CR><LF>
  - Some replace <TAB> by several whitespaces
- MIME includes mechanisms to solve these problems
  - Originally RFC 1341 (1992).
  - RFC 2045 and following MIME related RFCs

# MIME: new headers for RFC822

- Defined for compatibility with RFC822
  - Headers and message NVT ASCII
- MIME defines 5 new headers
  - MIME-Version:MIME-Version: 1.0
  - Content-Description: Human readable string
  - Content-Id: Unique Identifier (for referal)
  - **Content-Type**: Nature of the message; e.g. video/mpeg
  - **Content-Transfer-Encoding**: How the body is wrapped from transmission (see next slides)
- MIME is extensively used in many other protocols: HTTP, IRC, SIP, etc.



# Content-Type

- Text: textual information
  - **text/plain**: text without format
  - **text/richtext**: with simple format (rfc1341)
- Multipart: composed of several parts
  - **multipart/mixed**: independent in the specified order
  - **multipart/alternative**: same message in different formats
  - **multipart/parallel**: the parts must be viewed simultaneously
  - **multipart/digest**: each part is a complete rfc822 message
- Message
  - **message/rfc822**: complete message
  - **message/partial**: message divided for transmission
  - **message/external-body**: message must be obtained from the network

# Content-Type (II)

- Image
  - **image/jpeg**
  - **image/gif**
- Audio
  - **audio/basic: mono, 8 bit, 8 KHz, mu law**
- Video
  - **video/mpeg**
- Application: other data types
  - **application/octect-stream**: non-interpreted byte sequence
  - **application/postScript**: document printable in postscript



# Content-Transfer-Encoding

- Allows sending a non NVT ASCII message body
- New header with five possible values (though only two new encodings quoted-printable and base64):
  - ASCII 7 bits, Message originally compliant with RFC822
  - 8 bits with 1000 characters as line limit (MTAs support 8-bit-MIMETransport)
  - Binary (only leagl for Content-Type: message/external-body)
  - **quoted-printable**
    - Characters represented as “=“ followed by two hexadecimal digits, representing the octet value
    - Characters (33-60, 62-126) can be directly represented in ASCII ('=' by =3D)
    - tabs and whitespace at end of line encoded by “=09” or “=20”
    - Lines larger that 76 characters are divided setting “=“ as the last character
  - **Base64 encoding**
    - each 24 bits are split in 4 groups of 6 bits
    - pad if necessary and indicate to the other end with extra '='
    - line length 76 chars

# Quoted Printable

- Text encoded should be mostly readable
  - Text with few non NVT ASCII chars (like a message in spanish)
- Encoding rules:
  - Any byte (in hex XY) is encoded as “=XY”
    - But the end of line (<CR><LF>) which is encoded as (0x0D0A)
  - Chars in range 33-126 (0x21-0x7E) are not changed
    - But char 61 corresponding to symbol '=' is encoded as “=3D”
  - Tab (0x09) and whitespace (0x20) are not changed
    - But at the end of a line “=09” “=20”
  - Line length limited to 76 chars (excluding <CR><LF>)
    - If a line is longer, include a soft end of line “=<CR><LF>” (0x3D0D0A)



# Example

Content-Type: text/plain; charset=ISO-8859-1

Content-Transfer-Encoding: quoted-printable

Buenos d=EDas:

=A0=A0=A0 Por indicaci=F3n del Rector, os convoco a una comida en su comedor de Rectorado=A0 (junto a su despacho de Getafe) el lunes d=EDa 8 de marzo a las=
14:30 horas.
=20

# Base64 encoding

- To efficiently encode binary data. Rules:
  - Add padding bytes at the end (message length multiple of 3)
  - Every 3 bytes of data is grouped in 4 groups of 6 bits
  - Every group of 6 bits is coded with a char as the table in next slide (6 bits encoded as 8 bits)
  - Every 76 chars insert end of line (<CR><LF>)
  - Add as many '=' chars at the end as the initial padding
- Examples:

Content-Type: image/gif
Content-Transfer-Encoding: base64
JVBERi0xLjIKNyAwIG9iago8PAovVHlwZS9FbmNvZGluZwovRGlmZmVyZW5jZXNbMS9kb3RhY2Nl
bnQvZmkvZmwvZnJhY3Rpb24vaHVuZ2FydW1sYXV0L0xzbGFzaC9sc2xhc2gvb2dvbmVrL3Jpbmcg
[...]
MiAwIFIKL0luZm8gMSAwIFIKPj4Kc3RhcnR4cmVmCjU2MTg3CiUlRU9GCg==



# Base64 encoding table

| 6-bit value | ASCII char | | 6-bit value | ASCII char | | 6-bit value | ASCII char | | 6-bit value | ASCII char |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | A | | 10 | Q | | 20 | g | | 30 | w |
| 1 | B | | 11 | R | | 21 | h | | 31 | x |
| 2 | C | | 12 | S | | 22 | i | | 32 | y |
| 3 | D | | 13 | T | | 23 | j | | 33 | z |
| 4 | E | | 14 | U | | 24 | k | | 34 | 0 |
| 5 | F | | 15 | V | | 25 | l | | 35 | 1 |
| 6 | G | | 16 | W | | 26 | m | | 36 | 2 |
| 7 | H | | 17 | X | | 27 | n | | 37 | 3 |
| 8 | I | | 18 | Y | | 28 | o | | 38 | 4 |
| 9 | J | | 19 | Z | | 29 | p | | 39 | 5 |
| a | K | | 1a | a | | 2a | q | | 3a | 6 |
| b | L | | 1b | b | | 2b | r | | 3b | 7 |
| c | M | | 1c | c | | 2c | s | | 3c | 8 |
| d | N | | 1d | d | | 2d | t | | 3d | 9 |
| e | O | | 1e | e | | 2e | u | | 3e | + |
| f | P | | 1f | f | | 2f | v | | 3f | / |

# Some considerations on Content-Transfer-Encoding

- This header only applies to message body
  - Does not apply to header values
    - For instance to have an 'ñ' in the Subject:
  - Multipart messages may have different encoding for each part
    - If not present the default encoding is assumed: 7bit
  - The Content-Type does not condition a given encoding



# Non ascii values in headers: RFC 1342

  - From, To, Subject...
  - Obsoleted by RFC 2047

- =?charset?encoding?encoded-text?=
  - **charset**: us-ascii, iso-8859-x, etc.
  - **encoding**: a B/Q char, denoting:
    - B: Base64 encoding
    - Q: quoted-printable encoding
      - Difference: whitespace as "_"
  - **encoded-text**: text encoded

- Examples:

```
Subject: =?ISO-8859-1?Q?Feliz_a=F1o?=
From: =?US-ASCII?Q?Keith_Moore?= <moore@cs.utk.edu>
To: =?ISO-8859-l?Q?Kelcl_J=F8rn_Simonsen?= <keld@dkuug.dk>
CC: =?ISO-8859-l?Q?Andr=E9_?= Pirard <PIRARD@vml.ulg.ac.be>
Subject: =?ISO-8859-1?B?SWYgeW911GNhbiRoaXMgeW8=?= =?ISO-8859-
2?B?dSBIbinRlcnNOYW5klHRoZSxlLg==?=
```

# Example I

```
MIME-Version: 1.0
From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Subject: A multipart example
Content-Type: multipart/mixed;
     boundary=unique-boundary-1

This is the preamble area of a multipart message.
Mail readers that understand multipart format
should ignore this preamble.
If you are reading this text, you might want to
consider changing to a mail reader that understands
how to properly display multipart messages.
--unique-boundary-1

   ...Some text appears here...
[Note that the preceding blank line means
no header fields were given and this is text,
with charset US ASCII.  It could have been
done with explicit typing as in the next part.]
```



# Example (II)

```
--unique-boundary-1
Content-type: text/plain; charset=US-ASCII

This could have been part of the previous part,
but illustrates explicit versus implicit
typing of body parts.

--unique-boundary-1
Content-Type: multipart/parallel;
     boundary=unique-boundary-2

--unique-boundary-2
Content-Type: audio/basic
Content-Transfer-Encoding: base64

   ... base64-encoded 8000 Hz single-channel
       mu-law-format audio data goes here....

--unique-boundary-2
Content-Type: image/gif
Content-Transfer-Encoding: base64

   ... base64-encoded image data goes here....

--unique-boundary-2--
```

# Example (III)

```
--unique-boundary-1
Content-type: text/richtext

This is <bold><italic>richtext.</italic></bold>
<smaller>as defined in RFC 1341</smaller>
<nl><nl>Isn't it
<bigger><bigger>cool?</bigger></bigger>

--unique-boundary-1
Content-Type: message/rfc822

From: (mailbox in US-ASCII)
To: (address in US-ASCII)
Subject: (subject in US-ASCII)
Content-Type: Text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: Quoted-printable

   ... Additional text in ISO-8859-1 goes here ...

--unique-boundary-1--
```



# Simple Mail Transfer Protocol

- From UA origin to MTA
- Between MTAs
- RFC 821
- TCP port 25
- ASCII protocol
  - After client connects, server sends welcome message
  - Source and destination exchange commands and responses
- Each command in a line
  - Ending in <CR><LF>
- Commands (source -> destination)
  - NVT ASCII as in FTP
  - Only 8 compulsory commands
- Responses (destination -> source)
  - 3 digits code + wsp + opt. Msg
    - Code has same meaning as in FTP
    - Only the code is interpreted
  - Many different:

    250 message accepted

# SMTP commands

- Hello:
  - To identify the origin to the destination
  - `HELO <domain> <CRLF>`
- Mail:
  - To start an email transaction
  - `MAIL FROM:<reverse-path> <CRLF>`
- Recipient:
  - To specify one email recipient
  - `RCPT TO:<forward-path> <CRLF>`
- Data:
  - To send the message (rfc822, MIME)
  - `DATA <CRLF>`
- Quit:
  - To tell the recipient to send OK and close the connection
  - `QUIT <CRLF>`
- Reset:
  - Aborts current email transaction
  - `RSET <CRLF>`
- Verify:
  - Asks destination if there is any user <string>
  - `VRFY <string> <CRLF>`
- NOOP:
  - No operation. The destination answers “200: OK”.
  - `NOOP <CRLF>`



# SMTP: optional commands

– Expand:
  - Asks the destination to confirm if the email list <string> exists
  - **EXPN <string> <CRLF>**

– Help:
  - Sends list of available commands or information about the command <string>
  - **HELP [<string>] <CRLF>**

– Turn:
  - Exchanges the roles of origin (client) and destination (server)
  - No new establishment of TCP connection required
  - **TURN <CRLF>**

# SMTP: optional commands (contd.)

- Rarely implemented (can be used instead of MAIL)
  - Send:
    - Sends the message to the terminal
    - **SEND FROM:<reverse-path> <CRLF>**
  - Send or mail:
    - If it is connected, send the message to the terminal, otherwise via email
    - **SOML FROM:<reverse-path> <CRLF>**
  - Send and mail:
    - To the terminal and via email
    - **SAML FROM:<reverse-path> <CRLF>**



# Example SMTP responses

```
500 Syntax error, command unrecognized
501 Syntax error in parameters or arguments
502 Command not implemented
503 Bad sequence of commands
504 Command parameter not implemented

211 System status, or system help reply
214 Help message

220 <domain> Service ready
221 <domain> Service closing transmission channel
421 <domain> Service not available, closing transmission channel

250 Requested mail action okay, completed
251 User not local; will forward to <forward-path>
450 Requested mail action not taken: mailbox unavailable [E.g., mailbox busy]
550 Requested action not taken: mailbox unavailable [E.g., mailbox not found, no access]
451 Requested action aborted: error in processing
551 User not local; please try <forward-path>
452 Requested action not taken: insufficient system storage
552 Requested mail action aborted: exceeded storage allocation
553 Requested action not taken: mailbox name not allowed [E.g., mailbox syntax incorrect]
354 Start mail input; end with <CRLF>.<CRLF>
554 Transaction failed
```

# SMTP: command sequence

- *Open TCP connection to port 25*
  - R: 220 <server name> Service ready
    - Failure: 421
- Identification:
  - C: HELO < client name>
  - R: 250 Requested mail action okay, completed
    - Error: 500. 501, 504, 421
- Sending an email
  - C: MAIL FROM:<reverse-path>
  - R:250 Requested mail action okay, completed
    - Failure : 552, 451, 452
    - Error: 500, 501, 421
  - *Recipients (see next slide)*



# SMTP: command sequence II

- Recipients. There may be several recipients. RFC 822 To: headers are not considered
  - C: RCPT TO:<forward-path>
  - R: 250 Requested mail action okay, completed
    - 251 User not local; will forward to <forward-path>
    - Failure : 550, 551, 552, 553, 450, 451, 452
    - Error: 500, 501, 503, 421
- Data sending:
  - C: DATA
  - R: 354 Start mail input; end with <CRLF>.<CRLF>
    - Failure: 451, 554
    - Error: 500, 501, 503, 421
  - Client: message RFC 822 or MIME
  - Client: **.**
  - R:250 Requested mail action okay, completed
    - Failure : 552, 554, 451, 452

# SMTP: command sequence III

- Saying goodbye
  - C: QUIT
  - R: 221 <domain> Service closing transmission channel
    - Error: 500
- *Closing of connection*



# Normal Transaction

```
S: 220 BBN-UNIX.ARPA Simple Mail Transfer Service Ready
C: HELO USC-ISIF.ARPA
S: 250 BBN-UNIX.ARPA

C: MAIL FROM:<Smith@USC-ISIF.ARPA>
S: 250 OK

C: RCPT TO:<Jones@BBN-UNIX.ARPA>
S: 250 OK

C: RCPT TO:<Green@BBN-UNIX.ARPA>
S: 550 No such user here

C: RCPT TO:<Brown@BBN-UNIX.ARPA>
S: 250 OK

C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Blah blah blah...
C: ...etc. etc. etc.
C: .
S: 250 OK

C: QUIT
S: 221 BBN-UNIX.ARPA Service closing transmission channel
```

# Example: Verify before sending

```
S: 220 SU-SCORE.ARPA Simple Mail Transfer Service Ready
C: HELO MIT-MC.ARPA
S: 250 SU-SCORE.ARPA

C: VRFY Crispin
S: 250 Mark Crispin <Admin.MRC@SU-SCORE.ARPA>

C: MAIL FROM:<EAK@MIT-MC.ARPA>
S: 250 OK

C: RCPT TO:<Admin.MRC@SU-SCORE.ARPA>
S: 250 OK

C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Blah blah blah...
C: ...etc. etc. etc.
C: .
S: 250 OK

C: QUIT
S: 221 SU-SCORE.ARPA Service closing transmission channel
```



# Example: Reset

```
S: 220 USC-ISIF.ARPA Simple Mail Transfer Service Ready
C: HELO LBL-UNIX.ARPA
S: 250 USC-ISIF.ARPA

C: MAIL FROM:<mo@LBL-UNIX.ARPA>
S: 250 OK

C: RCPT TO:<fred@USC-ISIF.ARPA>
S: 251 User not local; will forward to <Jones@USC-ISI.ARPA>

C: RSET
S: 250 OK

C: QUIT
S: 221 USC-ISIF.ARPA Service closing transmission channel
```

# Example: mailing lists

```
 Step 1  --  Expanding the First List
S: 220 MIT-AI.ARPA Simple Mail Transfer Service Ready
C: HELO SU-SCORE.ARPA
S: 250 MIT-AI.ARPA

C: EXPN Example-People
S: 250-<ABC@MIT-MC.ARPA>
S: 250-Fred Fonebone <Fonebone@USC-ISIQ.ARPA>
S: 250-Xenon Y. Zither <XYZ@MIT-AI.ARPA>
S: 250-Quincy Smith <@USC-ISIF.ARPA:Q-Smith@ISI-VAXA.ARPA>
S: 250-<joe@foo-unix.ARPA>
S: 250 <xyz@bar-unix.ARPA>

C: QUIT
S: 221 MIT-AI.ARPA Service closing transmission channel
```



# RCPT TO determines the recipients

- TO, CC, and BCC RFC822 headers are not checked for delivery

```
S: 220 delta.aus.edu Simple Mail Transfer Service Ready
C: HELO stockholm.ibm.com
S: 250 delta.aus.edu
C: MAIL FROM:<abc@stockholm.ibm.com>
S: 250 OK
C: RCPT TO:<def@delta.aus.edu>
S: 250 OK
C: RCPT TO:<gfi@delta.aus.edu>
S: 550 No such user here
C: RCPT TO:<jkl@delta.aus.edu>
S: 250 OK
C: DATA
S: 354 Start mail input, end with <CRLF>.<CRLF>
C: Date: 23 Jan 89 18:05:23
C: From: Alex B. Carver <abc@stockholm.ibm.com>
C: Subject: Important meeting
C: To: <xyz@delta.aus.edu>
C: To: <opq@delta.aus.edu>
C: cc: <rst@delta.aus.edu>
C:
C: Blah blah blah
C: etc.....
C: .
S: 250 OK
C: QUIT
S: 221 delta.aus.edu Service closing transmission channel
```

# Mail relaying with RCPT TO

– RCPT TO may (partially) specify a delivery route (mail relays)- OBSOLETE

- RCPT TO: @host-a,@host-b:user@host-c

```
Step 1  --  Source Host to Relay Host
S: 220 USC-ISIE.ARPA Simple Mail Transfer Service Ready
C: HELO MIT-AI.ARPA
S: 250 USC-ISIE.ARPA

C: MAIL FROM:<JQP@MIT-AI.ARPA>
S: 250 OK

C: RCPT TO:<@USC-ISIE.ARPA:Jones@BBN-VAX.ARPA>
S: 250 OK

C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Date: 2 Nov 81 22:33:44
C: From: John Q. Public <JQP@MIT-AI.ARPA>
C: Subject:  The Next Meeting of the Board
C: To: Jones@BBN-Vax.ARPA
C:
C: Bill:
C: The next meeting of the board of directors will be
C: on Tuesday.
C:                                                John.
C: .
S: 250 OK

C: QUIT
S: 221 USC-ISIE.ARPA Service closing transmission channel
```



# Mail relaying (II)

```
Step 2  --  Relay Host to Destination Host
S: 220 BBN-VAX.ARPA Simple Mail Transfer Service Ready
C: HELO USC-ISIE.ARPA
S: 250 BBN-VAX.ARPA

C: MAIL FROM:<@USC-ISIE.ARPA:JQP@MIT-AI.ARPA>
S: 250 OK

C: RCPT TO:<Jones@BBN-VAX.ARPA>
S: 250 OK

C: DATA
S: 354 Start mail input; end with <CRLF>.<CRLF>
C: Received: from MIT-AI.ARPA by USC-ISIE.ARPA ; 2 Nov 81 22:40:10 UT
C: Date: 2 Nov 81 22:33:44
C: From: John Q. Public <JQP@MIT-AI.ARPA>
C: Subject:  The Next Meeting of the Board
C: To: Jones@BBN-Vax.ARPA
C:
C: Bill:
C: The next meeting of the board of directors will be
C: on Tuesday.
C:                                                John.
C: .
S: 250 OK

C: QUIT
S: 221 USC-ISIE.ARPA Service closing transmission channel
```

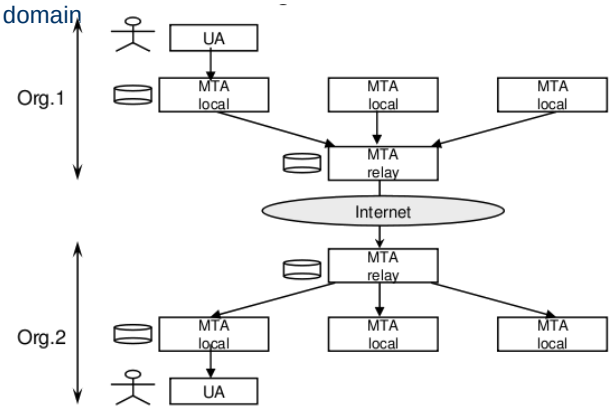
# Why mail relaying?

- Source UA may send it directly to destination MTA
  - If destination@dominio.es, query dns MX of dominio.es.
    - If no RR MX is found, query RR A
- Most often it is not send directly but through several hops
  - UAs usually configured to use a mail gateway (smarthost)
  - Usually a MTA is configured as the single entry/exit point in the organization
    - Allowed by firewall to receive connections in port 25
    - Control spamming from our domain
    - It is the destination dns MX RR
    - It delivers mail to the MTA of the intranet
      - Common view of the organization: xxx@abc.es
    - Restrict incoming spam easier (antivirus...)
- When no response from any MTA in dns MX RR
  - Keep on trying for at least 30 min, 4-5 days (RFC 1123-Requirements for Internet Host)



# Relay MTAs (intermmediate MTAs)

- Accept connections from our domain to send mail to any destination address, and
- Accept connections from any outside client to send mail to destination addresses in our domain



# More on RCPT TO

- We indicate our UA to send a message to: To: alice@abc.de, bob@def.es
- If the UA uses a relay MTA, it will set:

  RCPT TO: <alice@abc.de>

  250 OK

  RCPT TO: <bob@def.es>

  250 OK

  DATA
- The Relay MTA will query for MX RR of both domains and send them in two independent connections:

  RCPT TO: <alice@abc.de> RCPT TO: <bob@def.es>

  250 OK 250 OK

  DATA DATA
- The message in DATA will be exactly the same (but the BCC header if any of them was there)
- Each destination MTA will introduce its own Receive: (trace header)



# ESMTP

– SMTP limitations
  - Message length (some implementations msg limit 64 KB)
  - Problems with non NVT ASCII chars in message body
  - Client and server may have different timers
  - Security (authentication, confidenciality)

– ESMTP (Extended SMTP) solves many of them
  - Originally RFC 1425, then RFC 1869, 1652 (8-bit), 1870 (SIZE), 2554 (AUTH), 3207 (STARTTLS)...
  - Backwards compatible
    – new features available if client starts with EHLO, which follows multiline answer with ESMTP commands supported by server

      C: EHLO varpa.it.uc3m.es

      S: 500 Command unrecognized

      C: RSET

      S: 250 Reset state

      C: HELO varpa.it.uc3m.es

```
varpa:~>mail -v labscd@lab.it.uc3m.es
Subject: prueba de ESMTP
Envío un mail de prueba
termina con un punto.
.
EOT
labscd@lab.it.uc3m.es... Connecting to smtp.lab.it.uc3m.es via smtp...
220 lmserv2.lab.it.uc3m.es ESMTP Sendmail 8.9.3/8.9.3; Tue, 20 Apr 2004 10:29:43 +0200
>>> EHLO varpa.it.uc3m.es
250-lmserv2.lab.it.uc3m.es Hello varpa.it.uc3m.es [163.117.139.253], pleased to meet you
250-EXPN
250-VERB
250-8BITMIME
250-SIZE 10000000
250-DSN
250-ONEX
250-ETRN
250-XUSR
250 HELP
>>> MAIL From:<celeste@it.uc3m.es> SIZE=100 BODY=8BITMIME
250 <celeste@it.uc3m.es>... Sender and 8BITMIME ok; can accomodate 100 byte message
>>> RCPT To:<labscd@lab.it.uc3m.es>
250 <labscd@lab.it.uc3m.es>... Recipient ok
>>> DATA
354 Enter mail, end with "." on a line by itself
>>> .
250 KAA22824 Message accepted for delivery
labscd@lab.it.uc3m.es... Sent (KAA22824 Message accepted for delivery)
Closing connection to smtp.lab.it.uc3m.es
>>> QUIT
221 lmserv2.lab.it.uc3m.es closing connection
varpa:~>
```



# Final Delivery

- Users may read received messages with a UA
  - Running in the destination server (mailspool access)
  - Running in a different host using a final delivery protocol
    - POP3 (Post Office Protocol). RFC 1081, actual 1939. ASCII protocol (as SMTP, FTP) which allows to
      - Check messages in mailbox
      - Retrieve messages to UA for reading
      - Delete messages
    - IMAPv4 (Interactive Mail Access Protocol). RFC 1730, actual 3501
      - More sophisticated and powerful
      - Designed to work with UAs accessing from different hosts
      - Leave messages in server

# POP3

- RFC 1081, last version RFC 1939 (1996)
- ASCII protocol over 110 TCP port that allows to
  - See which messages are there in the mailbox
  - Bring messages from the MTA to the UA, to read them later on
  - Delete messages
- POP3 defines different phases:
  - Connection establishment `(S: +OK POP3 server ready)`
  - Once connected, the authorization phase is entered
  - Once the client succesfully authenticates, the transaction phase starts (and mailbox is locked!)
  - when the client sends the QUIT command, the UPDATE phase is entered, and the connection is closed
- Client sends commands and server send answers:
  - `S: + OK string`
  - `S: - ERR string`
  - Multiline answers are also possible, with the sequence "`<CR><LF>.<CR><LF>`" indicating the end of the answer (char `'.'` to be stuffed if it appears in a line alone)



# Commands

- Commands:
  - USER <name>
  - PASS <string>

```
C:    USER mrose
S:    +OK mrose is a real hoopy frood
C:    PASS secret
S:    +OK mrose's maildrop has 2 messages (320 octets)
...
C:    USER mrose
S:    +OK mrose is a real hoopy frood
C:    PASS secret
S:    -ERR unable to lock mrose's maildrop, file already
  locked
```

# Commands

- STAT
  - Number of messages and number of octets in the mailbox
  - Messages marked for deletion are not counted

```
C:    STAT
S:    +OK 2 320
```

- LIST [msg]

```
C: LIST
S: +OK 2 messages (320 octets)
S: 1 120
S: 2 200
S: .
...
C: LIST 2
S: +OK 2 200#
...
C: LIST 3
S: -ERR no such message, only 2 messages in maildrop
```



# Commands

**•RETR msg**

```
C: RETR 1
S: +OK 120 octets
S: <the POP3 server sends the entire message here>
S: .
```

**•DELE msg**

```
C: DELE 1
S: +OK message 1 deleted
...
C: DELE 2
S: -ERR message 2 already deleted
```

# Commands

- NOOP
  - LAST
    - Gives the last email accessed

```
C: STAT
S: +OK 4 320
C: LAST
S: +OK 1
C: RETR 3
S: +OK 120 octets
S: <the POP3 server sends the entire message here>
S: .
C: LAST
S: +OK 3
C: DELE 2
S: +OK message 2 deleted
C: LAST
S: +OK 2
C: RSET
S: +OK
C: LAST
S: +OK 1
```



# Commands

- RSET

  Deletion marks are removed from the maildrop

```
C: RSET
S: +OK maildrop has 2 messages (320 octets)
```

- QUIT

  Messages marked for deletion are physically deleted

```
C: QUIT
S: +OK dewey POP3 server signing off
```

# Optional commands

- `TOP msg [n]`
  - Sends the headers and the first n lines of the message msg
- `RPOP user`
  - Instead of PASS. Like .rhost
- `APOP <user> <MD5 de reto+password>`
  - Authenticate user not sending password in cleartext
    - `S: +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>`
    - `C: APOP mrose c4c9334bac560ecc979e58001b3e22fb`
    - `S: +OK maildrop has 1 message (369 octets)`
  - The final part of the welcome message is the nonce.
  - The second paramenter of APOP is MD5(nonce+password): `<1896.697170952@dbc.mtview.ca.us>tanstaaf.`
    - taanstaf is the user password
- RFC1734 allows any authentication mechanism for use with POP3



# Example

```
S: <wait for connection on TCP port 110>
C: <open connection>
S:    +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
C:    APOP mrose c4c9334bac560ecc979e58001b3e22fb
S:    +OK mrose's maildrop has 2 messages (320 octets)
C:    STAT
S:    +OK 2 320
C:    LIST
S:    +OK 2 messages (320 octets)
S:    1 120
S:    2 200
S:    .
```

```
C:   RETR 1
S:   +OK 120 octets
S:   <the POP3 server sends message 1>
S:   .
C:   DELE 1
S:   +OK message 1 deleted
C:   RETR 2
S:   +OK 200 octets
S:   <the POP3 server sends message 2>
S:   .
C:   DELE 2
S:   +OK message 2 deleted
C:   QUIT
S:   +OK dewey POP3 server signing off (maildrop empty)
C: <close connection>
S: <wait for next connection>
```



# Example cont.

```
C:   RETR 1
S:   +OK 120 octets
S:   <the POP3 server sends message 1>
S:   .
C:   DELE 1
S:   +OK message 1 deleted
C:   RETR 2
S:   +OK 200 octets
S:   <the POP3 server sends message 2>
S:   .
C:   DELE 2
S:   +OK message 2 deleted
C:   QUIT
S:   +OK dewey POP3 server signing off (maildrop empty)
C: <close connection>
S: <wait for next connection>
```

# IMAP4 (RFC 1730)

- RFC 1730 - Internet Message Access Protocol - Version 4
  - Last update RFC 3501.
- TCP port 143
  - More complex protocol than POP3!
  - mailbox concept (folder of messages at the server) allows to maintain messages in the server and simultaneously access from different machines
  - Unique ID allows simultaneous access to the same mailbox
  - IMAP supports MIME: message parts may be handled/downloaded separately
  - Three modes:
    - Offline:
      - client connects to server periodically and downloads messages
      - similar to POP3
    - Online:
      - client directly access and manipulates messages at the server
    - Disconnected:
      - client connects to server, downloads messages,
      - makes changes locally,
      - and periodically connects again to resynchronize (propagate changes to server and download new messages



# Mailboxes

- Messages are stored in mailboxes
- Default input mailbox: INBOX.
- User may create and destroy mailboxes (in the server)
- Mailboxes names
  - NVT ASCII.
  - Hierarchy is allowed
    - `Mail/cursos/redes`
  - Hierarchy separator char be any
    - Typically "/" or "."
    - LIST command allows the server to communicate the client the separator character
- Mailboxes may have attributes (returned with the LIST command):
  - `\Noinferiors`: indicates a leaf in the hierarchy
  - `\Noselect`: it may not be selected (with SELECT).
  - `\Marked`: it holds new messages since last selected
  - `\Unmarked`: it does not hold any new message since last selected

# Messages

- Messages are *flagged* to indicate status
  - `\Seen`
  - `\Answered`
  - `\Flagged` (important)
  - `\Deleted`
  - `\Draft`:
  - `\Recent`: (first session since message arrived)
- Two ways to identify a message in a mailbox (for reading, deletion, etc.)
  - By its sequence number:
    - relative position in mailbox (ascendent order)
    - subject to changes within a session or between sessions
      - when a message is deleted, sequence numbers are recalculated.
    - The number of messages within a mailbox is equal to the sequence number of the last message
  - By its unique identifier (UID):
    - 32 bit unique message id + 32 bit unique mailbox id
    - Message id:
      - Assigned when the message is stored in a mailbox
      - It does not change across sessions
    - Mailbox id:
      - Sent to the client when the mailbox is selected (UIDVALIDITY)
      - Changed for each session



# Commands and responses

- When the client connects, the server send a welcome message:
  - may be OK, PREAUTH or BYE.

```
S: * OK varpa.it.uc3m.es IMAP4rev1 v12.264 server ready
```

- Client sends commands
  - Started with a unique tag and ending with <CR><LF>

```
C: A023 NOOP
```

  - Tags allow to distinguish server responses
- Server answers
  - Server data: 0 or more lines starting with '*'
  - Server completion result: 1 line with
    - tag, result (OK, NO, BAD), command, text

```
C: a047 NOOP
S: * 23 EXISTS
S: * 3 RECENT
S: A047 OK NOOP Completed
```

# Commands and responses

- The client does not need to wait for server answer to send a new command
  - There are two exceptions:
    - During the authentication (AUTHENTICATE command)
    - when a string literal is sent (ending with {length})
  - In those cases the client should wait until a '+' is received to continue sending data
  - If data is sent before receiving '+', the server will send a BAD answer
- The server may send answers in a different order from commands
- The server may send unsolicited server answers
  - are prefixed by '*'
  - do not correspond to any command



# IMAP4 states

- <Not authenticated>: connection start
- <Authenticated>: credentials accepted, pre-authenticated connection, selection error/end, (LOGIN, AUTHENTICATE, PREAUTH)
- <Selected> mailbox: normal operation state (SELECT, EXAMINE)
- <Exit>: initiated by any of both sides (LOGOUT, errors)
- Normal cycle:
  - LOGIN->Authenticated,
  - SELECT->Selected,
    - XXX, # message access (FETCH, SEARCH, etc.)
  - LOGOUT ->Exit

# Command syntax

- Atoms: sequence of characters
  - capabilities, mailbox references, authentication chains, flags, etc.
- Numbers: sequence of digits
  - UID, MSN, sizes, etc.
    - MSN ≡ Message sequence number
- Strings:
  - Literals: `{4}`
    - Useful for user data (like appending data to a message)
  - Quoted: `"HELLO"`
- Lists: sequence of data

  `(\Seen \Deleted)`
- Flags: permanent or session

  `\Seen`, `\Recent`, `\Answered`, `\Flaged`, `\Deleted`, `\Draft`



# Commands

- Universal (any state)
  `CAPABILITY`, `NOOP`, `LOGOUT`
- Not Authenticated (non authenticated state)
  `AUTHENTICATE`, `LOGIN`
- Authenticated
  `SELECT`, `EXAMINE`, `CREATE`, `DELETE`, `RENAME`, `SUBSCRIBE`, `UNSUBSCRIBE`, `LIST`, `LSUB`, `STATUS`, `APPEND`
- Selected
  `CHECK`, `CLOSE`, `EXPUNGE`, `SEARCH`, `FETCH`, `STORE`, `COPY`, `UID`, `(plus all the previous)`

# <Not authenticated> LOGIN

- Arguments:
  - user name
  - password
- Server data: none
- Server completion result: OK or BAD
  - OK - login completed, now in authenticated state
  - BAD - login failure: user name or password rejected
- The LOGIN command identifies the client to the server and carries the plaintext password authenticating this user.
- Example:

```
C: a001 LOGIN SMITH SESAME
S: a001 OK LOGIN completed
```

- Not required if PREAUTH welcome message received:

```
S: * PREAUTH IMAP4rev1 server logged in as Smith
```



# <Not authenticated> AUTHENTICATE

- `AUTHENTICATE`: requires the server an authentication mechanism. In case the server does not support it, an error is returned.
  - Arguments: <authentication_mechanism_name>.
  - "Server data": +
  - Client send data continuation
  - "Server completion result": OK, NO or BAD.
  - Example:

```
S: * OK IMAP4rev1 Server
C: A001 AUTHENTICATE GSSAPI
S: +
C: YIIB+wYJKoZIhvcSAQICAQBIIB5qADAgEFoQMCAQ6iBw
   MFACAAAACjggEmYYIBIjCCAR6aESGxB1Lndhc2hpbmd0
   [...]
S: + YGgGCSqGSIb3EgECAgIAb1kwV6ADAgEFoQMCAQ+AMC
   AQGiQgRAtHTEuOP2BXb9sBYFR4SJlDIxmRBOhXRKdDA0
   uHTCOT9Bq3OsUTXUlk0CsFLoa8j+gvGDWHPSQg==
C: YDMGCSqGSIb3EgECAgIBAAD/////3LQBHgrejpLlLImP
   wkhbfa2QteAQAgAG1yYwE=
S: A001 OK GSSAPI authentication successful
```

  - `STARTTLS`: TLS negotiation.

# <Authenticated> SELECT

- To open a mailbox, if success transit to <Selected> state
  - `SELECT`
    - Arguments: <mailbox_name>.
    - "Server data" (no order defined):
      - `* FLAGS (<flags>)`
      - `* <n> EXISTS`: # messages in the mailbox
      - `* <n> RECENT`: # messages flagged \Recent.
      - `* OK [UNSEEN <n>]`: sequence number of first message unread (if the server does not send it, the client should issue a SEARCH command).
      - `* OK [PERMANENTFLAGS (<list of flags>)]`: flags that the user may change permanently (not just for this session)
      - `* OK [UIDNEXT <n>]`: next UID
      - `* OK [UIDVALIDITY <n>]`: mailbox ID
    - "Server completion result": OK, NO or BAD.
    - Example:

```
C: A142 SELECT INBOX
S: * 172 EXISTS
S: * 1 RECENT
S: * OK [UNSEEN 12] Message 12 is first unseen
S: * OK [UIDVALIDITY 3857529045] UIDs valid
S: * OK [UIDNEXT 4392] Predicted next UID
S: * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S: * OK [PERMANENTFLAGS (\Deleted \Seen \*)] Limited
S: A142 OK [READ-WRITE] SELECT completed
```

  - `EXAMINE`: same as SELECT, but mailbox is open in read only mode



# <Authenticated> LIST

- To see the mailboxes available at the server:
  - `LIST`:
    - Arguments: <directory> <name>.
      - If we list "" "" server returns hierarchy separator and root name
      - Wildcards may be used in the name (* states for any string)
    - "Server data":
      - `* LIST (mailbox attributes) "separator" mailbox_name`
    - "Server completion result": OK, NO or BAD.
    - Example:

```
C: A101 LIST "" ""
S: * LIST (\Noselect) "/" ""
C: A682 LIST "" *
S: * LIST () "/" blurdybloop
S: * LIST (\Noselect) "/" foo
S: * LIST () "/" foo/bar
S: A682 OK LIST completed
```

- Other commands are also possible:
  - `SUBSCRIBE`: to subscribe a mailbox to the list of mailboxes we are interested in
  - `UNSUBSCRIBE`: the mailbox is deleted from the list
  - `LSUB`: like LIST but only returns subscribed mailboxes

# <Authenticated> DELETE, RENAME

- DELETE: mailbox is deleted
  - Arguments: <mailbox_name>.
  - "Server data": none
  - "Server completion result": OK, NO or BAD.
  - Example:

```
C: A683 DELETE blurdybloop
S: A683 OK DELETE completed
C: A684 DELETE foo
S: A684 NO Name "foo" has inferior hierarchical names
C: A685 DELETE foo/bar
S: A685 OK DELETE Completed
C: A686 LIST "" *
S: * LIST (\Noselect) "/" foo
S: A686 OK LIST completed
C: A687 DELETE foo
S: A687 OK DELETE Completed
```

- RENAME: give a mailbox a new name
  - Arguments: <mailbox name>.
  - "Server data": none
  - "Server completion result": OK, NO or BAD.
  - Example:

```
C: A683 RENAME blurdybloop sarasoop
```



# <Authenticated> CREATE, STATUS

- To create a new mailbox:
  - CREATE
    - Arguments: <mailbox_name>.
      - may use hierarchy within the name (typically "/")
    - "Server data": none
    - "Server completion result": OK, NO or BAD.
    - Example:

```
C: A003 CREATE owatagusiam/
S: A003 OK CREATE completed
C: A004 CREATE owatagusiam/blurdybloop
S: A004 OK CREATE completed
```

- To inquire about the status of a given mailbox
  - STATUS
    - Arguments: <mailbox_name> (<state>)
      - states may be MESSAGES, RECENT, UIDNEXT, UIDVALIDITY, UNSEEN
    - "Server data": * STATUS <mailbox> (<state>)
    - "Server completion result": OK, NO or BAD.
    - Example:

```
C: A042 STATUS blurdybloop (UIDNEXT MESSAGES)
S: * STATUS blurdybloop (MESSAGES 231 UIDNEXT 44292)
S: A042 OK STATUS completed
```

# <Authenticated> APPEND

- APPEND: adds new message to a mailbox
  - Arguments:
    - <mailbox_name>
    - (<flags>) (optional)
    - <day/time> (optional)
    - {length}
  - "Server data": +.
  - After receiving the '+' the client sends the message of the indicated length
  - "Server completion result": OK, NO or BAD.
  - Example:

```
C: A003 APPEND saved-messages (\Seen) {310}
S: + Ready for literal data
C: Date: Mon, 7 Feb 1994 21:52:25 -0800 (PST)
C: From: Fred Foobar <foobar@Blurdybloop.COM>
C: Subject: afternoon meeting
C: To: mooch@owatagu.siam.edu
C: Message-Id: <B27397-0100000@Blurdybloop.COM>
C: MIME-Version: 1.0 C: Content-Type: TEXT/PLAIN;
 CHARSET=US-ASCII
C:
C: Hello Joe, do you think we can meet at 3:30 tomorrow?
C:
S: A003 OK APPEND completed
```



# <Selected> SEARCH

- SEARCH:
  - Arguments: <search_criteria>
    - ALL, NEW, OLD, ANSWERED, DELETED, FROM/TO/CC/BCC <string>, BEFORE/SINCE/SENTBEFORE/SENTSINCE <date>, BODY/TEXT <string>, LARGER/SMALLER <n>...
  - "Server data": * SEARCH <n> <m> ...
    - Returns 0 or more message numbers matching the search criteria
  - "Server completion result": OK, NO or BAD.
  - Example:

```
C: A282 SEARCH SINCE 1-Feb-1994 NOT FROM "Smith"
S: * SEARCH 2 84 882
S: A282 OK SEARCH completed
C: A283 SEARCH TEXT "string not in mailbox"
S: * SEARCH
S: A283 OK SEARCH completed
C: A284 SEARCH CHARSET UTF-8 TEXT {6}
S: + Ready for additional command text
C: XXXXXX
S: * SEARCH 43
```

# <Selected> FETCH, STORE

- To download a message
  - FETCH:
    - Arguments: <message_range> <search criteria>.
      - FLAGS, INTERNALDATE, RFC822.SIZE, ENVELOPE, BODY...
      - It may be used to retrieved a given part of the MIME structure
    - "Server data": * <n> FETCH
    - "Server completion result": OK, NO or BAD.
    - Example:

```
C: A654 FETCH 2:4 (FLAGS BODY[HEADER.FIELDS (DATE FROM)])
S: * 2 FETCH ....
S: * 3 FETCH ....
S: * 4 FETCH ....
S: A654 OK FETCH completed
```

- To change the flags of a message (for instance to mark for deletion):
  - STORE:
    - Arguments: <message_range> +/-FLAGS (<flags>)
    - "Server data": * **FETCH** (FLAGS (<flags>))
    - "Server completion result": OK, NO or BAD.
    - Example:

```
C: A003 STORE 2:4 +FLAGS (\Deleted)
S: * 2 FETCH (FLAGS (\Deleted \Seen))
S: * 3 FETCH (FLAGS (\Deleted))
S: * 4 FETCH (FLAGS (\Deleted \Flagged \Seen))
S: A003 OK STORE completed
```



# <Selected> EXPUNGE, COPY

- To permanently delete messages flagged with \Deleted:
  - EXPUNGE:
    - Arguments: none
    - "Server data": * <deleted_message> EXPUNGE.
    - "Server completion result": OK, NO or BAD.
    - Example:

```
C: A202 EXPUNGE
S: * 2 EXPUNGE
S: * 3 EXPUNGE
S: * 4 EXPUNGE
S: A202 OK EXPUNGE completed
```

- To copy a message in a mailbox
  - COPY:
    - Arguments: <message_range> <mailbox_name>.
    - "Server data": none
    - "Server completion result": OK, NO or BAD.
    - Example:

```
C: A003 COPY 2:4 MEETING
```

# <Selected> UID, CLOSE

- To use the UID of a message:
  - UID:
    - UID SEARCH to get the message.
    - UID COPY / FETCH / STORE to access the message by its UID.
    - Ejemplo:

```
C: A999 UID FETCH 4827313:4828442 FLAGS
S: * 23 FETCH (FLAGS (\Seen) UID 4827313)
S: * 24 FETCH (FLAGS (\Seen) UID 4827943)
S: * 25 FETCH (FLAGS (\Seen) UID 4828442)
S: A999 OK UID FETCH completed
```

- To close the mailbox
  - CLOSE:
    - Arguments: ninguno.
    - “Server data”: none
    - “Server completion result”: OK or BAD.
    - Example:

```
C: A341 CLOSE
S: A341 OK CLOSE completed
```

  - If a mailbox was open and a SELECT, EXAMINE or LOGOUT is issued, a CLOSE is automatically performed before



# <Any state> CAPABILITY, NOOP

- CAPABILITY: inquires the server about the supported functions
  - Arguments: none
  - “Server data”: * CAPABILITY <capabilities>.
  - “Server completion result”: OK or BAD.
  - Example:

```
C: abcd CAPABILITY
S: * CAPABILITY IMAP4rev1 STARTTLS AUTH=GSSAPI LOGINDISABLED
S: abcd OK CAPABILITY completed
C: efgh STARTTLS
S: efgh OK STARTLS completed
<TLS negotiation, further commands are under [TLS] layer>
C: ijkl CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=GSSAPI AUTH=PLAIN
S: ijkl OK CAPABILITY completed
```

- NOOP: no operation (incativity timer is reset).
  - Arguments: none
  - “Server data”: none, but it may provoke status information from the server
    - It may be used to poll the server about new messages arrival
  - “Server completion result”: OK or BAD.
  - Example:

```
C: a002 NOOP
S: a002 OK NOOP completed
. . .
C: a047 NOOP
S: * 23 EXISTS
S: * 3 RECENT
S: * 14 FETCH (FLAGS (\Seen \Deleted))
S: a047 OK NOOP completed
```

# <Any state> LOGOUT

- `LOGOUT`: end connection request
  - Arguments: none
  - “Server data”: `* BYE`.
  - “Server completion result”: OK or BAD.
  - Example:

```
C: A023 LOGOUT
S: * BYE IMAP4rev1 Server logging out
S: A023 OK LOGOUT completed
```



```
S:   * OK IMAP4rev1 Service Ready
C:   a001 login mrc secret
S:   a001 OK LOGIN completed
C:   a002 select inbox
S:   * 18 EXISTS
S:   * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S:   * 2 RECENT
S:   * OK [UNSEEN 17] Message 17 is the first unseen message
S:   * OK [UIDVALIDITY 3857529045] UIDs valid
S:   a002 OK [READ-WRITE] SELECT completed
C:   a003 fetch 12 full
S:   * 12 FETCH (FLAGS (\Seen) INTERNALDATE "17-Jul-1996 02:44:25 -0700"
      RFC822.SIZE 4286 ENVELOPE ("Wed, 17 Jul 1996 02:23:25 -0700 (PDT)"
      "IMAP4rev1 WG mtg summary and minutes"
      (("Terry Gray" NIL "gray" "cac.washington.edu"))
      (("Terry Gray" NIL "gray" "cac.washington.edu"))
      (("Terry Gray" NIL "gray" "cac.washington.edu"))
      ((NIL NIL "imap" "cac.washington.edu"))
      ((NIL NIL "minutes" "CNRI.Reston.VA.US")
      ("John Klensin" NIL "KLENSIN" "MIT.EDU")) NIL NIL
      "<B27397-0100000@cac.washington.edu>")
       BODY ("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 3028
       92))
S:    a003 OK FETCH completed
```

```
C:    a004 fetch 12 body[header]
S:    * 12 FETCH (BODY[HEADER] {342}
S:    Date: Wed, 17 Jul 1996 02:23:25 -0700 (PDT)
S:    From: Terry Gray <gray@cac.washington.edu>
S:    Subject: IMAP4rev1 WG mtg summary and minutes
S:    To: imap@cac.washington.edu
S:    cc: minutes@CNRI.Reston.VA.US, John Klensin <KLENSIN@MIT.EDU>
S:    Message-Id: <B27397-0100000@cac.washington.edu>
S:    MIME-Version: 1.0
S:    Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
S:
S:    )
S:    a004 OK FETCH completed
C:    a005 store 12 +flags \deleted
S:    * 12 FETCH (FLAGS (\Seen \Deleted))
S:    a005 OK +FLAGS completed
C:    a006 logout
S:    * BYE IMAP4rev1 server terminating connection
S:    a006 OK LOGOUT completed
```



# Example (string literals)

```
B LOGIN {4}  //Not completed
+ Ready for argument
pepe {6}
+ Ready for argument
passwd
B OK LOGIN completed
```

# Example (list)

```
A LIST ~/ p*
* LIST (\Noselect) "/" ~/prop
* LIST (\Noinferiors \Unmarked) "/" ~/p1
* LIST (\Noinferiors \Unmarked) "/" ~/p2
A OK LIST completed
```



# Example (delete message)

```
// Delete message 52
B102 STORE 52 +FLAGS (\Deleted)
* 63 EXISTS
* 9 RECENT
*52 FETCH (\FLAGS (\Seen \Deleted))
B102 OK STORE completed
```

# mark for deletion and logout

```
C:  a005 store 12 +flags \deleted
S:  * 12 FETCH (FLAGS (\Seen \Deleted))
S:  a005 OK +FLAGS completed
C:  a006 logout
S:  * BYE IMAP4rev1 server terminating connection
S:  a006 OK LOGOUT completed
```



# Interlaced commands

```
A12 SEARCH FROM javier@p.com BODY prueba
A13 STATUS INBOX (RECENT)
* SEARCH 52 54
* 64 EXISTS
* 10 RECENT
A12 OK SEARCH completed
A13 OK STATUS
BB102 EXPUNGE
BA1 LOGOUT
* 52 EXPUNGE
* 63 EXISTS
* 10 RECENT
BB102 OK EXPUNGE 1 Messages
* BYE server IMAP4rev1 server terminating
BA1 OK LOGOUT completed
```