The Modern Mail Handler

Markus Schnalke

Master's Thesis

advised by

Prof. Dr. Franz Schweiggert
and Dr. Andreas Borchert

Ulm University

2012

At the heart of the UNIX philosophy is the idea that the power of a system comes more from the relationship among programs than from the programs themselves.

— Brian W. Kernighan and Rob Pike —

CONTENTS

PREFACE

I have discovered the mail client *nmh* in Fall 2009. At that time I used *mutt*, as many advanced Unix users do. When I read about nmh, its concepts convinced me at once. The transition from mutt to nmh was similar to beginning with file management in the Unix shell when being used to the *midnight commander*, or like starting with vi when being used to modeless editors. Such a change is not trivial, but, in being convinced by the concepts and by having done similar transitions for file management and editing already, it was not too difficult. In contrast, setting up nmh to a convenient state became a tedious task that took several months. Once having nmh arranged to a convenient state, I enjoyed using it because of its conceptional elegance and its scripting capabilities. Nevertheless, it still was inconvenient for handling attachments, non-ASCII character encodings, and similar features of modern emailing. My setup demanded more and more additional configuration and helper scripts to have nmh behave the way I wanted; yet my expectations were rather common for modern emailing. In being a computer scientist and programmer, I wanted to improve the situation.

In Spring 2010, I asked on the *nmh-workers* mailing list for the possibility to offer a Google Summer of Code project for me. Participating in the development of nmh this way appeared attractive to me, because I would have been able to work full time on nmh. Although the nmh community had been generally positive on the suggestion, the administrative work for a GSoC project had been to much to have it realized. Nontheless, my proposal had activated the nmh community. In the following weeks, goals for nmh's future were discussed. In these discussions, I became involved in the question whether nmh should include mail transfer facilities [ML:MTA-MUA]. I argued for the MTA of nmh to be removed. In this fundamental question, my opinion differed from the opinion of most others. Sadly, besides the discussions, hardly any real work was done. Being unable to work on nmh in a way that would be accepted at university as part of my studies, I needed to choose another project.

Half a year later, starting in August 2010, I took one semester off to travel through Latin America. During my time in Argentina, I wanted to work on Free Software. This brought me back to nmh. Richard Sandelman, an active nmh user, cared for the official basis. Juan Granda, an argentine Free Software developer, provided a computer with Internet connection. Thanks to them, I was able to work on nmh during my three-month stay in Santiago del Estero, Argentina. Quickly it became obvious that I wouldn't succeed with my main goal, to improve the character encoding handling. (One of its ramifications is the missing transfer decoding of quoted text in replies.) As this is one of the most intricate parts of the system, the goal was simply set too high. Instead, I improved the code base as I read through it. I found minor bugs for which I proposed fixes. In the same go, I improved the documentation in minor ways. When I started with larger code changes, I had to discover that the community was reluctant to change. Its wish for compatibility was much stronger than its wish for convenient out-of-the-box setups – in contrast to my opinion. This led to long discussions, again. I came to understand their point of view, but it was different to mine. At the end of

my three-month project, I had become familiar with nmh's code base and community, I had improved the project in minor ways, and I still was convinced that I wanted to continue to do so.

Another half year later, the end of my studies came within reach. I needed a topic for my master's thesis. No question, I wanted to work on nmh. But well, not exactly on nmh, because I had accepted that the nmh community has different goals than I have. Working on nmh would result in much discussion and, in consequence, little progress. After careful thought, I decided to start an experimental version of nmh. I wanted to implement my own ideas of how an MH-like system should look like. I wanted to create a usable alternative version to be compared with the present state of nmh. Eventually, my work would be proven successful or not. In any case, the nmh project would profit from my experiences.

Focus of this Document

This document explains the design goals and implementation decisions for mmh. It discusses technical, historical, social and philosophical considerations. On the technical side, this document explains how an existing project was stream-lined by removing rough edges and exploiting the central concepts better. On the historical side, changes through time in the use cases and the email features, as well as the reactions to them, are discussed. Socially, this document describes the effects and experiences of a newcomer with revolutionary aims entering an old and matured software project. Philosophical thoughts on style, mainly based to the Unix philosophy, are present throughout the discussions. The document describes the changes to nmh, but as well, it clarifies my personal perception of the concepts of MH and Unix, and explain my therefrom resulting point of view.

This document is written for the community around MH-like mail systems, including developers and users. Despite the focus on MH-like systems, this document is may be precious to anyone interested in the Unix philosophy and anyone in contact to old software projects, be it code or community-related.

The reader is expected to be well familiar with Unix, C and emailing. Good Unix shell knowledge is required, because MH relies fundamentally on the shell. Without the power of the shell, MH becomes a motorbike without winding roads: boring. Introductions to Unix and its shell can be found in ''The UNIX Programming Environment'' by Kernighan and Pike [Kernighan84] or ''The UNIX System'' by Bourne [Bourne82]. The reader is assumed to be a C programmer, but the document should be understandable otherwise, too. The definitive guide to C is Kernighan and Ritchie's ''The C Programming Language'' [Kernighan88]. Some book about system-level C programming can be helpful additional literature. Rochkind and Curry have written such books [Rochkind85, Curry96]. As large parts of the source code are old, old books are likely more helpful for understanding. The reader is expected to know the format of email messages and the structure of email transfer systems, at least on a basic level. It's advisable to have cross-read the RFCs 821 and 822. Further more, basic understanding of MIME is good to have. The Wikipedia provides good introduction-level information to email.

Frequent references to the Unix philosophy will be made. Gancarz has tried to sum it up in his book ''The UNIX Philosophy'' [Gancarz95]. Even better, though less

concrete, are ''The UNIX Programming Environment'' [Kernighan84] and ''The Practice of Programming'' [Kernighan99] by Kernighan and Pike. The term paper ''Why the Unix Philosophy still matters'' [Schnalke10] by myself provides an overview on the philosophy, including a case study of MH.

Although a brief introduction to MH is provided in Chapter 1, the reader is encouraged to have a look at the *MH Book* ''MH & nmh: Email for Users & Programmers'' by Jerry Peek [Peek95]. The current version is available freely on the Internet. It is the definitive guide to MH and nmh.

This document is neither a user's tutorial to mmh nor an introduction to any of the topics covered. The technical discussions are on an advanced level. Nevertheless, as knowledge of the fundamental concepts is the most valuable information a user can acquire about some program or software system, this document may be worth a read for non-developers as well.

Organization

Which font for what use. Meaning of 'foo(1)'. RFCs.

References to source code repository commits are printed as [☞ 1a2b3c4]. They can be looked up with '`git show XXX`' on the command line or online at `http://git.marmaro.de/?p=mmh;a=commitdiff;h=XXX`, replacing '`XXX`' with the hash value. In this example: '`git show 1a2b3c4`' or `http://git.marmaro.de/?p=mmh;a=commitdiff;h=1a2b3cd`. Whereas the code repository will probably be available on the Internet forever, a website URL is always at risk to change.

This thesis is divided into XXX chapters, ...

Chapter 1 introduces ...

Chapter 2 describes ...

Chapter 3 covers ...

Acknowledgments

To be written at the very end.

Chapter 1
INTRODUCTION

MH is a set of mail handling tools with a common concept, similar to the Unix tool chest, which is a set of file handling tools with a common concept. *nmh* is the currently most popular implementation of an MH-like mail handling system. This thesis describes an experimental version of nmh, named *mmh*.

This chapter introduces MH, its history, concepts and how it is used. It describes nmh's code base and community to give the reader a better understanding of the state from which mmh started off. Further more, this chapter outlines the mmh project itself, describing the motivation for it and its goals.

1.1 MH – THE MAIL HANDLER

MH is a conceptual email system design and its concrete implementation. Notably, MH had started as a design proposal at RAND Corporation, where the first implementation followed later. In spirit, MH is similar to Unix, which influenced the world more in being a set of system design concepts than in being a specific software product. The ideas behind Unix are summarized in the *Unix philosophy*. MH follows this philosophy.

History

In 1977 at RAND Corporation, Norman Shapiro and Stockton Gaines had proposed the design of a new mail handling system, called ''Mail Handler'' (MH), to superseed RAND's old monolithic ''Mail System'' (MS). Two years later, in 1979, Bruce Borden took the proposal and implemented a prototype of MH. Before the prototype had been available, the concept was believed to be practically unusable. But the prototype had proven successful and replaced MS thereafter. In replacing MS, MH grew to an all-in-one mail system.

In the early Eighties, the University of California at Irvine (UCI) had started to use MH. Marshall T. Rose and John L. Romine became the driving force then. They took over the development and pushed MH forward. RAND had put the code into the public domain by then. MH was developed at UCI at the time when the Internet appeared, when UCB implemented the TCP/IP stack, and when Allman wrote Sendmail. MH was extended as emailing became more featured. The development of MH was closely related to the development of email RFCs. In the advent of MIME, MH was the first implementation of this new email standard.

In the Nineties, the Internet had become popular and in December 1996, Richard Coleman initiated the ''New Mail Handler'' (nmh) project. Nmh is a fork of MH 6.8.3 and bases strongly on the *LBL changes* by Van Jacobson, Mike Karels and Craig Leres. Colman intended to modernize MH and improve its portability and MIME handling

capabilities. This should be done openly within the Internet community. The development of MH at UCI stopped after the 6.8.4 release in February 1996, soon after the development of nmh had started. Today, nmh has almost completely replaced the original MH. Some systems might still provide old MH, but mainly for historical reasons.

In the last years, the work on nmh was mostly maintenance work. However, the development revived in December 2011 and stayed busy since then.

Concepts

MH consists of a set of tools, each covering a specific task of email handling, like composing a message, replying to a message, refiling a message to a different folder, listing the messages in a folder. All of the programs operate on a common mail storage.

The mail storage consists of *mail folders* (directories) and *messages* (regular files). Each message is stored in a separate file in the format it had been received (i.e. transfer format). The files are named with ascending numbers in each folder. The specific format of the mail storage characterizes MH in the same way like the format of the file system characterizes Unix.

MH tools maintain a *context*, which includes the current mail folder. Processes in Unix have a similar context, containing the current working directory, for instance. In contrast, the process context is maintained by the Unix kernel automatically, whereas MH tools need to maintain the MH context themselves. The user can have one MH context or multiple ones, he can even share it with other users.

Messages are named by their numeric filename, but they can have symbolic names, too. These are either automatically updated position names like being the next or the last message, or user-settable group names for arbitrary sets of messages. These names are called sequences. Sequences can be bound to the containing folder or to the context.

The user's *profile* is a file that contains his MH configuration. Default switches for the individual tools can be specified to adjust them to the user's personal preferences. Multiple versions of the same command with different default values can also be created very easily. Form templates for new messages or for replies are easily changeable, and output is adjustable with format files. Almost every part of the system can be adjusted to personal preference.

The system is well scriptable and extensible. New MH tools are built out of or on top of existing ones quickly. Further more, MH encourages the user to tailor, extend and automate the system. As the MH tool chest was modeled after the Unix tool chest, the properties of the latter apply to the former as well.

Using MH

It is strongly recommended to have a look at the MH Book, which introduces well into using MH [Peek95, Part II]. Rose and Romine provide a deeper and more technical though slightly outdated introduction in only about two dozens pages [Rose85].

Following is an example mail handling session. It uses mmh but is mostly compatible with nmh and old MH. Details might vary but the look'n'feel is the same.

```
$ inc
Incorporating new mail into inbox...

   1+ 2012-05-16 11:16  meillo@dream.home  Hello
   2  2012-05-16 11:17  meillo@dream.home  book

$ show
Date:    Wed, 16 May 2012 11:16:00 +0200
To:      meillo
From:    <meillo@dream.home.schnalke.org>
Subject: Hello

part       text/plain                  13
mmh is great

$ next
Date:    Wed, 16 May 2012 11:17:24 +0200
To:      meillo
From:    <meillo@dream.home.schnalke.org>
Subject: book

part       text/plain                  79
Hello meillo,

have a look at the ``Daemon book´´. You need to read that!

foo

$ rmm 1

$ scan
   2+ 2012-05-16 11:17  meillo@dream.home  book

$
```

1.2 NMH: CODE AND COMMUNITY

In order to understand the condition, goals and dynamics of a project, one needs to know the reasons. This section explains the background.

MH predates the Internet, it comes from times before networking was universal, it comes from times when emailing was small, short and simple. Then it grew, spread and adopted to the changes email went through. Its core-concepts, however, remained the same. During the Eighties students at UCI actively worked on MH. They added new features and optimized the code for the then popular systems. All this still was in times before POSIX and ANSI C. As large parts of the code stem from this time, today's nmh source code still contains many ancient parts. BSD-specific code and

constructs tailored for hardware of that time are frequent.

Nmh started about a decade after the POSIX and ANSI C standards had been established. A more modern coding style entered the code base, but still a part of the developers came from ''the old days''. The developer base became more diverse and thus resulted in code of different style. Programming practices from different decades merged in the project. As several peers added code, the system became more a conglomeration of single tools rather than a homogeneous of-one-cast mail system. Still, the existing basic concepts held it together. They were mostly untouched throughout the years.

Despite the tool chest approach at the surface – a collection of separate small programs – on the source code level it is much more interweaved. Several separate components were compiled into one program for efficiency reasons. This lead to intricate innards. Unfortunately, the clear separation on the outside appeared as being pretty interweaved inside.

The advent of MIME rose the complexity of email by a magnitude. This is visible in nmh. The MIME-related parts are the most complex ones. It's also visible that MIME support had been added on top of the old MH core. MH's tool chest style made this easily possible and encourages such approaches, but unfortunately, it lead to duplicated functions and half-hearted implementation of the concepts.

To provide backward-compatibility, it is a common understanding to not change the default settings. In consequence, the user needs to activate modern features explicitly to be able to use them. This puts a burden on new users, because out-of-the-box nmh remains in the same ancient style. If nmh is seen to be a back-end, then this compatibility surely is important. However, in the same go, new users have difficulties to use nmh for modern emailing. The small but matured community around nmh hardly needs much change as they have their convenient setups since decades.

1.3 MMH

I started to work on my experimental version in October 2011, at a time when there were no more than three commits to nmh since the beginning of the year. In December, when I announced my work in progress on the nmh-workers mailing list [ML:mmh_ann], nmh's community became active, too. This movement was heavily pushed by Paul Vixie's ''edginess'' comment [ML:edginess]. After long years of stagnation, nmh became actively developed again. Hence, while I was working on mmh, the community was working on nmh, in parallel.

The name *mmh* may stand for *modern mail handler*, because the project tries to modernize nmh. Personally however, I prefer to call mmh *meillo's mail handler*, emphasizing that the project follows my visions and preferences. (My login name is *meillo*.) This project model was inspired by *dwm*, which is Anselm Garbe's personal window manager – targeted to satisfy Garbe's personal needs whenever conflicts appear. Dwm had retained its lean elegance and its focused character, whereas its community-driven predecessor *wmii* had grown fat over time. The development of mmh should remain focused.

Motivation

MH is the most important of very few command line tool chest email systems. Tool chests are powerful because they can be perfectly automated and extended. They allow arbitrary kinds of front-ends to be implemented on top of them quickly and without internal knowledge. Additionally, tool chests are much better to maintain than monolithic programs. As there are few tool chests for emailing and as MH-like ones are the most popular among them they should be developed further. This keeps their conceptional elegance and unique scripting qualities available to users. Mmh will create a modern and convenient entry point to MH-like systems for new and interested users.

The mmh project is motivated by deficits of nmh and my wish for general changes, combined with the nmh community's reluctancy to change.

nmh hadn't adjusted to modern emailing needs well enough. The default setup was completely unusable for modern emailing. Too much setup work was required. Several modern features were already available but the community didn't wanted to have them as default. mmh is a way to change this.

In my eyes, MH's concepts could be exploited even better and the style of the tools could be improved. Both would simplify and generalize the system, providing cleaner interfaces and more software leverage at the same time. mmh is a way to demonstrate this.

In providing several parts of an email system, nmh can hardly compete with the large specialized projects that focus on only one of the components. The situation can be improved by concentrating the development power on the most unique part of MH and letting the user pick his preferred set of other mail components. Today's prepackaged software components encourage this model. mmh is a way to go for this approach.

It's worthwhile to fork nmh for the development of mmh, because the two projects focus on different goals and differ in fundamental questions. The nmh community's reluctance to change conflicts with my strong will to change. In developing a separate experimental version new approaches can easily be tried out without the need to discuss changes beforehand. In fact, revolutionary changes are hardly possible otherwise.

The mmh project provides the basis to implemented and demonstrated the listed ideas without the need to change nmh or its community. Of course, the results of the mmh project shall improve nmh, in the end.

Target Field

Any effort needs to be targeted towards a specific goal in order to be successful. Following is a description of the imagined typical mmh user. mmh should satisfy his needs. Actually, as mmh is my personal version of MH, this is a description of myself.

The target user of mmh likes Unix and its philosophy. He likes to use programs that are conceptionally appealing. He's familiar with the command line and enjoys its power. He is at least capable of shell scripting and wants to improve his productivity by scripting the mail system. He naturally uses modern email features, like attachments, non-ASCII text, and digital cryptography. He is able to setup email system components besides mmh, and actually likes the choice to pick the ones he prefers. He has a reasonably modern system that complies to standards, like POSIX and ANSI C.

The typical user invokes mmh commands directly in an interactive shell session, but as well, he uses them to automate mail handling tasks. Likely, he runs his mail setup on a server machine, to which he connects via ssh. He might also have local mmh installations on his workstations, but does rather not rely on graphical front-ends. He definitely wants to be flexible and thus be able to change his setup to suite his needs.

The typical mmh user is a programmer himself. He likes to, occasionally, take the opportunity of Free Software to put hands on and get involved in the software he uses. Hence, he likes small and clean code bases and he cares for code quality. In general, he believes that:

- Elegance – i.e. simplicity, clarity and generality – is most important.
- Concepts are more important than the concrete implementation.
- Code optimizations for anything but readability should be avoided if possible.
- Having a lot of choice is bad.
- Removed code is debugged code.

Goals

The general goals for the mmh project are the following:

Stream-lining
: Mmh should be stripped down to its core, which is the user agent (MUA). The feature set should be distilled to the ones really needed, effectively removing corner-cases. Parts that don't add to the main task of being a conceptionally appealing MUA should be removed. This includes, the mail submission and mail retrieval facilities. Choice should be reduced to the main options.

Modernizing
: Mmh's feature set needs to become more modern. Better support for attachment and digital cryptography needs to be added. MIME support needs to be integrated deeper and more naturally. The modern email features need to be readily available, out-of-the-box. And on the other hand, bulletin board support and similar obsolete facilities need to be dropped out. Likewise, ancient technologies, like hardcopy terminals, should not be supported any further.

Code style
: Mmh's source code needs to be updated to modern standards. Standardized library functions should replace non-standard versions whenever possible. Code should be separated into distinct modules when possible. Time and space optimizations should to be replaced by clear and readable code. A uniform programming style should prevail.

Homogeneity
: The available concepts need to be expanded as far as possible. A small set of concepts should prevail thoroughly throughout the system. The whole system should appear to be of-one-style. It should feel like being cast as one.

Chapter 2
DISCUSSION

This main chapter discusses the practical work done in the mmh project. It is structured along the goals to achieve. The concrete work done is described in the examples of how the general goals were achieved. The discussion compares the current version of mmh with the state of nmh just before the mmh project started, i.e. Fall 2011. Current changes of nmh will be mentioned only as side notes.

2.1 STREAM-LINING

MH had been considered an all-in-one system for mail handling. The community around nmh has a similar understanding. In fundamental difference, mmh shall be a MUA only. I believe that the development of all-in-one mail systems is obsolete. Today, email is too complex to be fully covered by single projects. Such a project won't be able to excel in all aspects. Instead, the aspects of email should be covered my multiple projects, which then can be combined to form a complete system. Excellent implementations for the various aspects of email exist already. Just to name three examples: Postfix is a specialized MTA, Procmail is a specialized MDA, and Fetchmail is a specialized MRA. I believe that it is best to use such specialized tools instead of providing the same function again as a side-component in the project.

Doing something well, requires to focus on a small set of specific aspects. Under the assumption that focused development produces better results in the particular area, specialized projects will likely be superior in their field of focus. Hence, all-in-one mail system projects – no matter if monolithic or modular – will never be the best choice in any of the fields. Even in providing the best consistent all-in-one system they are likely to be beaten by projects that focus only on integrating existing mail components to a homogeneous system.

The limiting resource in Free Software community development is usually man power. If the development power is spread over a large development area, it becomes even more difficult to compete with the specialists in the various fields. The concrete situation for MH-based mail systems is even tougher, given the small and aged community, including both developers and users, it has.

In consequence, I believe that the available development resources should be focused on the point where MH is most unique. This is clearly the user interface – the MUA. Peripheral parts should be removed to stream-line mmh for the MUA task.

2.1.1 Removal of the Mail Transfer Facilities

In contrast to nmh, which also provides mail submission and mail retrieval agents, mmh is a MUA only. This general difference in the view on the character of nmh

initiated the development of mmh. Removing the mail transfer facilities had been the first work task in the mmh project.

The MSA is called *Message Transfer Service* (MTS) in nmh. The facility established network connections and spoke SMTP to submit messages for relay to the outside world. This part was implemented by the `post` command. The changes in email in the last years demanded changes in this part of nmh too. Encryption and authentication for network connections needed to be supported, hence TLS and SASL were introduced into nmh. This added complexity to nmh without improving it in its core functions. Also, keeping up with recent developments in the field of mail transfer requires development power and specialists. In mmh this whole facility was simply cut off. [☞ f6aa95b] [☞ fecd5d3] [☞ 156d35f] Instead, mmh depends on an external MSA. The only outgoing interface available to mmh is the `sendmail` command, which almost any MSA provides. If not, a wrapper program can be written. It must read the message from the standard input, extract the recipient addresses from the message header, and hand the message over to the MSA. For example, a wrapper script for qmail would be:

```
#!/bin/sh
# ignore command line arguments
exec qmail-inject
```

The requirement to parse the recipient addresses out of the message header is likely to be removed in the future. Then mmh would give the recipient addresses as command line arguments. This is clearly the better interface, but mmh does not provide it yet.

To retrieve mail, the `inc` command established network connections and spoke POP3 to retrieve mail from remote servers. As with mail submission, the network connections required encryption and authentication, thus TLS and SASL were added. Support for message retrieval through IMAP will become necessary to be added soon, too, and so on for any changes in mail transfer. Mmh has dropped the support for retrieving mail from remote locations. [☞ ab7b484] Instead, it depends on an external tool to cover this task. In mmh there exist two paths for messages to enter mmh's mail storage: (1) Mail can be incorporate with `inc` from the system maildrop, or (2) with `rcvstore` by reading them, one at a time, from the standard input.

With the removal of the MSA and MRA, mmh converted from an all-in-one mail system to being a MUA only. Following the Unix philosophy, it now focuses on one job and tries to do that one well. Not only the programs follow that tenet but also the project itself does so. Now, of course, mmh depends on third-party software. An external MSA is required to transfer mail to the outside world; an external MRA is required to retrieve mail from remote machines. There exist excellent implementations of such software, which do this specific task likely better than the internal versions had done it. Also, the best suiting programs can be freely chosen.

As it had already been possible to use an external MSA or MRA, why not keep the internal version for convenience? The question whether there is sense in having a fall-back pager in all the command line tools, for the cases when `more` or `less` aren't available, appears to be ridiculous. Now, an MSA or MRA is more complex than a text pager and not necessarily available but still the concept of orthogonal design holds: ''Write programs that do one thing and do it well.'' [Salus94, McIlroy78] Here, this part of the Unix philosophy was applied not only to the programs but to the project itself.

In other words: ‘‘Develop projects that focus on one thing and do it well.’’ Projects grown complex should be split for the same reasons programs grown complex should be split. If it is conceptionally more elegant to have the MSA and MRA separate projects then they should be separated. This is the case here, in my opinion. The RFCs propose this separation by clearly distinguishing the different mail handling tasks [RFC 821]. The small interfaces between the mail agents support the separation.

In the beginning, email had been small and simple. (`/bin/mail` had once covered anything there was to email and still had been small and simple.) Then the essential complexity of email increased. (Essential complexity is the complexity defined by the problem itself. [Brooks86]) Email systems reacted to this change: They grew. RFCs started to introduce mail agents and separated the various tasks because the existing tasks became more extensive and new tasks appeared. Again, email systems grew, or they split parts off. In nmh, for instance, the POP server, which the original MH had included, was removed. Now is the time to go one step further and remove the MSA and MRA, too. Not only does this decrease the code size of the project, but, more important, it unburdens mmh of the whole field of message transfer with all its implications for the project. There's no more need to concern with changes in network transfer. This independence is received by depending on an external program that covers the field. Today, this is a reasonable exchange.

Function can be added in three different ways:

- Implementing the function originally in the project.
- Depending on a library that provides the function.
- Depending on a program that provides the function.

Whereas adding the function originally to the project increases the code size most and requires most maintenance and development work, it makes the project most independent of other software. Using libraries or external programs require less maintenance work but introduces dependencies on external software. Programs have the smallest interfaces and provide the best separation but possibly limit the information exchange. External libraries are stronger connected than external programs, thus information can be exchanged more flexible. Adding code to a project increases maintenance work. Implementing complex functions originally in the project will add a lot of code. This should be avoided if possible. Hence, the dependencies only change in kind, not in their existence. In mmh, library dependencies on `libsasl2` and `libcrypto`/`libssl` were treated against program dependencies on an MSA and an MRA. This also meant treating build-time dependencies against run-time dependencies. Besides program dependencies providing the stronger separation and being more flexible, they also allowed over 6 000 lines of code to be removed from mmh. This made mmh's code base about 12 % smaller. Reducing the project's code size by such an amount without actually losing functionality is a convincing argument. Actually, as external MSAs and MRAs are likely superior to the project's internal versions, the common user even gains functionality.

Users of MH should not have problems to set up an external MSA and MRA. Also, the popular MSAs and MRAs have large communities and a lot of documentation available. Choices for MSAs range from full-featured MTAs like *Postfix* over mid-size MTAs like *masqmail* and *dma* to small forwarders like *ssmtp* and *nullmailer*. Choices

for MRAs include *fetchmail*, *getmail*, *mpop* and *fdm*.

2.1.2 Removal of non-MUA Tools

One goal of mmh is to remove the tools that are not part of the MUA's task. Further more, any tools that don't improve the MUA's job significantly should be removed. Loosely related and rarely used tools distract from the lean appearance. They require maintenance work without adding much to the core task. On removing these tools, the project shall become more stream-lined and focused. In mmh the following tools are not available anymore:

- `conflict` was removed [☞ 8b23509] because it is a mail system maintenance tool that is not MUA-related. It even checked `/etc/passwd` and `/etc/group` for consistency, which is completely unrelated to email. A tool like `conflict` is surely useful, but it should not be shipped with mmh.
- `rcvtty` was removed [☞ 14767c9] because its use case of writing to the user's terminal on receiving of mail is obsolete. If users like to be informed of new mail, the shell's `MAILPATH` variable or graphical notifications are technically more appealing. Writing directly to a terminals is hardly ever wanted today. If though one wants to have it this way, the standard tool `write` can be used in a way similar to:

  ```
  scan -file - | write `id -un`
  ```

- `viamail` was removed [☞ eda72d6] when the new attachment system was activated, because `forw` could then cover the task itself. The program `sendfiles` was rewritten as a shell script wrapper around `forw`. [☞ 0e82199]
- `msgchk` was removed [☞ bb9360e], because it lost its use case when POP support was removed. A call to `msgchk` provided hardly more information than:

  ```
  ls -l /var/mail/meillo
  ```

  It did distinguished between old and new mail, but this detail information and can be retrieved with `stat`(1), too. A very small shell script could be written to output the information in a similar way, if truly necessary. As mmh's `inc` only incorporates mail from the user's local maildrop, and thus no data transfers over slow networks are involved, there's hardly any need to check for new mail before incorporating it.
- `msh` was removed [☞ 9166901] because the tool was in conflict with the philosophy of MH. It provided an interactive shell to access the features of MH, but it wasn't just a shell, tailored to the needs of mail handling. Instead it was one large program that had several MH tools built in. This conflicts with the major feature of MH of being a tool chest. `msh`'s main use case had been accessing Bulletin Boards, which have seized to be popular.

Removing `msh`, together with the truly archaic code relicts `vmh` and `wmh`, saved more than 7 000 lines of C code – about 15 % of the project's original source code amount.

Having less code (with equal readability, of course) for the same functionality is an advantage. Less code means less bugs and less maintenance work. As `rcvtty` and `msgchk` are assumed to be rarely used and can be implemented in different ways, why should one keep them? Removing them stream-lines mmh. `viamail`'s use case is now

partly obsolete and partly covered by `forw`, hence there's no reason to still maintain it. `conflict` is not related to the mail client, and `msh` conflicts with the basic concept of MH. Theses two tools might still be useful, but they should not be part of mmh.

Finally, there's `slocal`. `slocal` is an MDA and thus not directly MUA-related. It should be removed, because including it is a violation of the idea that mmh is a MUA only. It should become a separate project. However, `slocal` provides rule-based processing of messages, like filing them into different folders, which is otherwise not available in mmh. Although `slocal` does neither pull in dependencies nor does it include a separate technical area (cf. Sec. XXX), still it accounts for about 1 000 lines of code that need to be maintained. As `slocal` is almost self-standing, it should be split off into a separate project. This would cut the strong connection between the MUA mmh and the MDA `slocal`. For anyone not using MH, `slocal` would become yet another independent MDA, like *procmail*. The need to install a complete MH system to have `slocal` would be gone. Likewise, mmh users could decide to use *procmail* without having a second, unused MDA, `slocal`, installed. That's conceptionally the best solution. Yet, `slocal` is not split off. I feel unsure with removing it from mmh. Hence, I defer the decision over `slocal`. In the meanwhile `slocal` does not hurt because it is unrelated to the rest of mmh.

2.1.3 `show` and `mhshow`

Since the very beginning – already in the first concept paper – `show` had been MH's message display program. `show` mapped message numbers and sequences to files and invoked `mhl` to have the files formatted. With MIME, this approach wasn't sufficient anymore. MIME messages can consist of multiple parts, some of which aren't directly displayable, further more text content might be encoded in foreign charsets. `show`'s understanding of messages and `mhl`'s display capabilities couldn't cope with the task any longer.

Instead of extending these tools, additional tools were written from scratch and added to the MH tool chest. Doing so is encouraged by the tool chest approach. Modular design is a great advantage for extending a system, as new tools can be added without interfering with existing ones. First, the new MIME features were added in form of the single program `mhn`. The command '`mhn -show 42`' would show the MIME message numbered 42. With the 1.0 release of nmh in February 1999, Richard Coleman finished the split of `mhn` into a set of specialized tools, which together covered the multiple aspects of MIME. One of them was `mhshow`, which replaced '`mhn -show`'. It was capable of displaying MIME messages appropriately.

From then on, two message display tools were part of nmh, `show` and `mhshow`. To ease the life of users, `show` was extended to automatically hand the job over to `mhshow` if displaying the message would be beyond `show`'s abilities. In consequence, the user would simply invoke `show` (possibly through `next` or `prev`) and get the message printed with either `show` or `mhshow`, whatever was more appropriate.

Having two similar tools for essentially the same task is redundant. Usually, users wouldn't distinguish between `show` and `mhshow` in their daily mail reading. Having two separate display programs was therefore mainly unnecessary from a user's point of view. Besides, the development of both programs needed to be in sync, to ensure that the programs behaved in a similar way, because they were used like a single tool.

Different behavior would have surprised the user.

Today, non-MIME messages are rather seen to be a special case of MIME messages, although it's the other way round. As `mhshow` had already be able to display non-MIME messages, it appeared natural to drop `show` in favor of using `mhshow` exclusively. [☞ `4c1efdd`] Removing `show` is no loss in function, because functionally `mhshow` covers it completely. The old behavior of `show` can still be emulated with the simple command line:

```
mhl `mhpath c`
```

For convenience, `mhshow` was renamed to `show` after `show` was gone. It is clear that such a rename may confuse future developers when trying to understand the history. Nevertheless, I consider the convenience on the user's side, to call `show` when they want a message to be displayed, to outweigh the inconvenience on the developer's side when understanding the project history.

To prepare for the transition, `mhshow` was reworked to behave more like `show` first. (cf. Sec. XXX) Once the tools behaved more alike, the replacing appeared to be even more natural. Today, mmh's new `show` became the one single message display program again, with the difference that today it handles MIME messages as well as non-MIME messages. The outcome of the transition is one program less to maintain, no second display program for users to deal with, and less system complexity.

Still, removing the old `show` hurts in one regard: It had been such a simple program. Its lean elegance is missing to the new `show`. But there is no chance; supporting MIME demands for higher essential complexity.

2.1.4 Removal of Configure Options

Customization is a double-edged sword. It allows better suiting setups, but not for free. There is the cost of code complexity to be able to customize. There is the cost of less tested setups, because there are more possible setups and especially corner-cases. And, there is the cost of choice itself. The code complexity directly affects the developers. Less tested code affects both, users and developers. The problem of choice affects the users, for once by having to choose, but also by complexer interfaces that require more documentation. Whenever options add little advantages, they should be considered for removal. I have reduced the number of project-specific configure options from fifteen to three.

Mail Transfer Facilities

With the removal of the mail transfer facilities five configure options vanished:

The switches `--with-tls` and `--with-cyrus-sasl` had activated the support for transfer encryption and authentication. This is not needed anymore. [☞ `fecd5d3`] [☞ `156d35f`]

The configure switch `--enable-pop` activated the message retrieval facility. The code area that would be conditionally compiled in for TLS and SASL support had been small. The conditionally compiled code area for POP support had been much larger. Whereas the code base changes would only slightly change on toggling TLS or SASL support, it changed much on toggling POP support. The changes in the code base

could hardly be overviewed. By having POP support togglable a second code base had been created, one that needed to be tested. This situation is basically similar for the conditional TLS and SASL code, but there the changes are minor and can yet be overviewed. Still, conditional compilation of a code base creates variations of the original program. More variations require more testing and maintenance work.

Two other options only specified default configuration values: `--with-mts=[smtp|sendmail]` defined the default transport service. In mmh this fixed to *`sendmail`*. [☞ `f6aa95b`] With `--with-smtpservers=[server1...]` default SMTP servers for the *`smtp`* transport service could be specified. [☞ `128545e`] Both of them became irrelevant.

Backup Prefix

The backup prefix is the string that was prepended to message filenames to tag them as deleted. By default it had been the comma character '`,`'. In July 2000, Kimmo Suominen introduced the configure option `--with-hash-backup` to change the default to the hash symbol '`#`'. The choice was probably personal preference, because first, the option was named `--with-backup-prefix=` and had the prefix symbol as argument. Because giving the hash symbol as argument caused to many problems for configure, the option was limited to use the hash symbol as the default prefix. This makes me believe, that the choice for the hash was personal preference. Being it related or not, words that start with the hash symbol introduce a comment in the Unix shell. Thus, the command line '`rm #13 #15`' calls `rm` without arguments because the first hash symbol starts the comment that reaches until the end of the line. To delete the backup files, '`rm ./#13 ./#15`' needs to be used. Using the hash as backup prefix can be seen as a precaution agains data loss.

I removed the configure option but added the profile entry `backup-prefix`, which allows to specify an arbitrary string as backup prefix. [☞ `6c40d48`] Profile entries are the common method to change mmh's behavior. This change did not remove the choice but moved it to a location where it suited better.

Eventually, however, the new trash folder concept (→ Sec. XXX) obsoleted the concept of the backup prefix completely. [☞ `8edc5aa`] (Well, there still are corner-cases to remove until the backup prefix can be laid to rest, eventually.)

Editor and Pager

The two configure options `--with-editor=EDITOR --with-pager=PAGER` were used to specify the default editor and pager at configure time. Doing so at configure time made sense in the Eighties, when the set of available editors and pagers varied much across different systems. Today, the situation is more homogeneous. The programs `vi` and `more` can be expected to be available on every Unix system, as they are specified by POSIX since two decades. (The specifications for `vi` and `more` appeared in [XVS87] and [XCU92], respectively.) As a first step, these two tools were hard-coded as defaults. [☞ `5d43a99`] Not changed were the `editor` and `moreproc` profile entries, which allowed the user to override the system defaults. Later, the concept was reworked to respect the standard environment variables `VISUAL` and `PAGER` if they are set. Today, mmh determines the editor to use in the following order, taking the first available and non-empty item:

(1) Environment variable MMHEDITOR

(2) Profile entry Editor

(3) Environment variable VISUAL

(4) Environment variable EDITOR

(5) Command vi.

[☞ f85f4b7]

The pager to use is determined in a similar order, also taking the first available and non-empty item:

(1) Environment variable MMHPAGER

(2) Profile entry Pager (replaces moreproc)

(3) Environment variable PAGER

(4) Command more.

[☞ 0c4214e]

By respecting the VISUAL/EDITOR and PAGER environment variables, the new behavior confirms better to the common style on Unix systems. Additionally, the new approach is more uniform and clearer to users.

Locale

The configure option --disable-locale was removed because POSIX provides locale support and there's hardly any need to disable locale support. [☞ ccf4f17]

ndbm

slocal used to depend on *ndbm*, a database library. The database is used to store the 'Message-ID's of all messages delivered. This enables slocal to suppress delivering the same message to the same user twice. (This features was enabled by the -suppressdup switch.)

A variety of version of the database library exist [Wolter04]. Complicated autoconf code was needed to detect them correctly. Further more, the configure switches --with-ndbm=ARG and --with-ndbmheader=ARG were added to help with difficult setups that would not be detected automatically or correctly.

By removing the suppress duplicates feature of slocal, the dependency on *ndbm* vanished and 120 lines of complex autoconf code could be saved. [☞ ecd6d6a] The change removed functionality too, but that is minor to the improvement by dropping the dependency and the complex autoconf code.

mh-e Support

The configure option --disable-mhe was removed when the mh-e support was reworked. Mh-e is the Emacs front-end to MH. It requires MH to provide minor additional functions. The --disable-mhe configure option could switch these extensions off. After removing the support for old versions of mh-e, only the -build switches of forw and repl are left to be mh-e extensions. They are now always built in because they add little code and complexity. In consequence, the --disable-mhe configure

option was removed [☞ a7ce7b4] Removing the option removed a second code setup that would have needed to be tested. This change was first done in nmh and thereafter merged into mmh.

The interface changes in mmh require mh-e to be adjusted in order to be able to use mmh as back-end. This will require minor changes to mh-e, but removing the `-build` switches would require more rework.

Masquerading

The configure option `--enable-masquerade` could take up to three arguments: 'draft_from', 'mmailid', and 'username_extension'. They activated different types of address masquerading. All of them were implemented in the SMTP-speaking `post` command, which provided an MSA. Address masquerading is an MTA's task and mmh does not cover this field anymore. Hence, true masquerading needs to be implemented in the external MTA.

The *mmailid* masquerading type is the oldest one of the three and the only one available in the original MH. It provided a *username* to *fakeusername* mapping, based on the password file's GECOS field. The man page *mh-tailor(5)* described the use case as being the following:

> This is useful if you want the messages you send to always appear to come from the name of an MTA alias rather than your actual account name. For instance, many organizations set up 'First.Last' sendmail aliases for all users. If this is the case, the GECOS field for each user should look like: ''First [Middle] Last <First.Last>''

As mmh sends outgoing mail via the local MTA only, the best location to do such global rewrites is there. Besides, the MTA is conceptionally the right location because it does the reverse mapping for incoming mail (aliasing), too. Further more, masquerading set up there is readily available for all mail software on the system. Hence, mmailid masquerading was removed. [☞ 0836c80]

The *username_extension* masquerading type did not replace the username but would append a suffix, specified by the `USERNAME_EXTENSION` environment variable, to it. This provided support for the *user-extension* feature of qmail and the similar *plussed user* processing of sendmail. The decision to remove this username_extension masquerading was motivated by the fact that `spost` hadn't supported it already. [☞ 2abae0b] Username extensions are possible in mmh, but less convenient to use.

The *draft_from* masquerading type instructed `post` to use the value of the `From:` header field as SMTP envelope sender. Sender addresses could be replaced completely. [☞ b14ea60] Mmh offers a kind of masquerading similar in effect, but with technical differences. As mmh does not transfer messages itself, the local MTA has final control over the sender's address. Any masquerading mmh introduces may be reverted by the MTA. In times of pedantic spam checking, an MTA will take care to use sensible envelope sender addresses to keep its own reputation up. Nonetheless, the MUA can set the `From:` header field and thereby propose a sender address to the MTA. The MTA may then decide to take that one or generate the canonical sender address for use as envelope sender address.

In mmh, the MTA will always extract the recipient and sender from the message header (`sendmail`'s `-t` switch). The `From:` header field of the draft may be set arbitrary by the user. If it is missing, the canonical sender address will be generated by the MTA.

Remaining Options

Two configure options remain in mmh. One is the locking method to use: `--with-locking=[dot|fcntl|flock|lockf]`. The idea of removing all methods except the portable dot locking and having that one as the default is appealing, but this change requires deeper technical investigation into the topic. The other option, `--enable-debug`, compiles the programs with debugging symbols and does not strip them. This option is likely to stay.

2.1.5 Removal of Switches

The command line switches of MH tools follow the X Window style. They are words, introduced by a single dash. For example: '`-truncate`'. Every program in mmh has two generic switches: `-help`, to print a short message on how to use the program, and `-Version`, to tell what version of mmh the program belongs to.

Switches change the behavior of programs. Programs that do one thing in one way require no switches. In most cases, doing something in exactly one way is too limiting. If there is basically one task to accomplish, but it should be done in various ways, switches are a good approach to alter the behavior of a program. Changing the behavior of programs provides flexibility and customization to users, but at the same time it complicates the code, documentation and usage of the program. Therefore, the number of switches should be kept small. A small set of well-chosen switches does no harm. But usually, the number of switches increases over time. Already in 1985, Rose and Romine have identified this as a major problem of MH [Rose85, p. 12]:

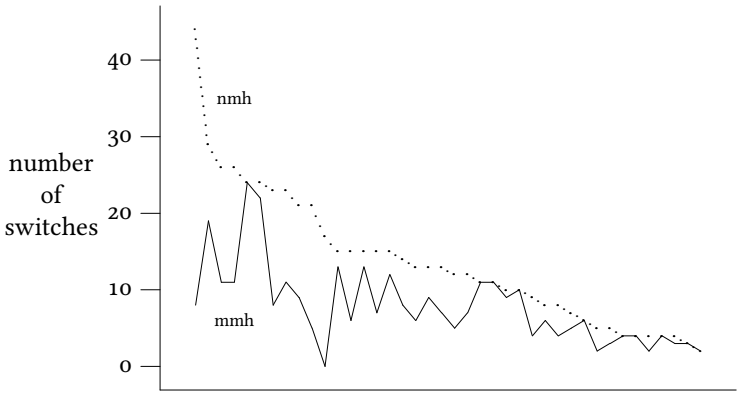
> A complaint often heard about systems which undergo substantial development by many people over a number of years, is that more and more options are introduced which add little to the functionality but greatly increase the amount of information a user needs to know in order to get useful work done. This is usually referred to as creeping featurism.
>
> Unfortunately MH, having undergone six years of off-and-on development by ten or so well-meaning programmers (the present authors included), suffers mightily from this.

Being reluctant to adding new switches – or 'options', as Rose and Romine call them – is one part of a counter-action, the other part is removing hardly used switches. Nmh's tools had lots of switches already implemented, hence, cleaning up by removing some of them was the more important part of the counter-action. Removing existing functionality is always difficult because it breaks programs that use these functions. Also, for every obsolete feature, there'll always be someone who still uses it and thus opposes its removal. This puts the developer into the position, where sensible improvements to style are regarded as destructive acts. Yet, living with the featurism is far worse, in my eyes, because future needs will demand adding further features, worsening the situation more and more. Rose and Romine added in a footnote, ''[...]

`send` will no doubt acquire an endless number of switches in the years to come.'' Although clearly humorous, the comment points to the nature of the problem. Refusing to add any new switches would encounter the problem at its root, but this is not practical. New needs will require new switches and it would be unwise to block them strictly. Nevertheless, removing obsolete switches still is an effective approach to deal with the problem. Working on an experimental branch without an established user base, eased my work because I did not offend users when I removed existing funtions.

Rose and Romine counted 24 visible and 9 more hidden switches for `send`. In nmh, they increased up to 32 visible and 12 hidden ones. At the time of writing, no more than 7 visible switches and 1 hidden switch have remained in mmh's `send`. (These numbers include two generic switches, help and version.)

Fig. XXX displays the number of switches for each of the tools that is available in both, nmh and mmh. Visible as well as hidden switches were counted, but not the generic help and version switches. Whereas in the beginning of the project, the average tool had 11 switches, now it has no more than 5 – only half as many. If the 'no' switches and similar inverse variant are folded onto their counter-parts, the average tool has 8 switches in pre-mmh to 4 now. The total number of functional switches in mmh dropped from 465 to 234.



A part of the switches vanished after functions were removed. This was the case for network mail transfer, for instance. Sometimes, however, the work flow was the other way: I looked through the *mh-chart*(7) man page to identify the tools with apparently too many switches. Then considering the value of each of the switches by examining the tool's man page and source code, aided by recherche and testing. This way, the removal of functions was suggested by the aim to reduce the number of switches per command.

Draft Folder Facility

A change early in the project was the completely transition from the single draft message to the draft folder facility. [☞ 337338b] The draft folder facility was introduced in the mid-Eighties. (Rose and Romine called it a ''relatively new feature'' [Rose85] in 1985.) Since then, the facility had existed but was deactivated by default. The default

activation and the related rework of the tools made it possible to remove the `-[no]draftfolder`, and `-draftmessage` switches from `comp`, `repl`, `forw`, `dist`, `whatnow`, and `send`. [☞ 337338b] The only flexibility removed with this change is having multiple draft folders within one profile. I consider this a theoretical problem only. In the same go, the `-draft` switch of `anno`, `refile`, and `send` was removed. The special-casing of 'the' draft message became irrelevant after the rework of the draft system. (See Sec. XXX.) Equally, `comp` lost its `-file` switch. The draft folder facility, together with the `-form` switch, are sufficient.

Inplace Editing

`anno` had the switches `-[no]inplace` to either annotate the message inplace and thus preserve hard links, or annotate a copy to replace the original message, breaking hard links. Following the assumption that linked messages should truly be the same message, and annotating it should not break the link, the `-[no]inplace` switches were removed and the previous default `-inplace` was made the only behavior. [☞ c819584] The `-[no]inplace` switches of `repl`, `forw`, and `dist` could be removed, too, as they were simply passed through to `anno`.

`burst` also had `-[no]inplace` switches, but with different meaning. With `-inplace`, the digest had been replaced by the table of contents (i.e. the introduction text) and the bursted messages were placed right after this message, renumbering all following messages. Also, any trailing text of the digest was lost, though, in practice, it usually consists of an end-of-digest marker only. Nontheless, this behavior appeared less elegant than the `-noinplace` behavior, which already had been the default. Nmh's *burst*(1) man page reads:

> If -noinplace is given, each digest is preserved, no table of contents is produced, and the messages contained within the digest are placed at the end of the folder. Other messages are not tampered with in any way.

The decision to drop the `-inplace` behavior was supported by the code complexity and the possible data loss it caused. `-noinplace` was chosen to be the definitive behavior. [☞ 68a686a]

Forms and Format Strings

Historically, the tools that had `-form` switches to supply a form file had `-format` switches as well to supply the contents of a form file as a string on the command line directly. In consequence, the following two lines equaled:

```
scan -form scan.mailx
scan -format "`cat .../scan.mailx`"
```

The `-format` switches were dropped in favor for extending the `-form` switches. [☞ f51956b] If their argument starts with an equal sign ('='), then the rest of the argument is taken as a format string, otherwise the arguments is treated as the name of a format file. Thus, now the following two lines equal:

```
scan -form scan.mailx
scan -form "=`cat .../scan.mailx`"
```

This rework removed the prefix collision between `-form` and `-format`. Now, typing `-fo` suffices to specify form or format string.

The different meaning of `-format` for `repl` and `forw` was removed in mmh. `forw` was completely switched to MIME-type forwarding, thus removing the `-[no]format`. [☞ 6e27160] For `repl`, the `-[no]format` switches were reworked to `-[no]filter` switches. [☞ 67411b1] The `-format` switches of `send` and `post`, which had a third meaning, were removed likewise. [☞ f3cb7cd] Eventually, the ambiguity of the `-format` switches was resolved by not anymore having any such switch in mmh.

MIME Tools

The MIME tools, which were once part of `mhn`, had several switches that added little practical value to the programs. The `-[no]realsize` switches of `mhbuild` and `mhlist` were removed, doing real size calculations always now [☞ 8d8f1c3], as ‘‘This provides an accurate count at the expense of a small delay.’’ This small delay is not noticable on modern systems.

The `-[no]check` switches were removed together with the support for `Content-MD5:` header fields [RFC 821]. [☞ 31dc797] (See Sec. XXX)

The `-[no]ebcdicsafe` and `-[no]rfc934mode` switches of `mhbuild` were removed because they are considered obsolete. [☞ 01a3480] [☞ 3363e26]

Content caching of external MIME parts, activated with the `-rcache` and `-wcache` switches was completely removed. [☞ d1fefd9] External MIME parts are rare today, having a caching facility for them is appears to be unnecessary.

In pre-MIME times, `mhl` had covered many tasks that are part of MIME handling today. Therefore, `mhl` could be simplified to a large extend, reducing the number of its switches from 21 to 6. [☞ 350ad6d] [☞ 0e46503]

Mail Transfer Switches

With the removal of the mail transfer facilities, a lot of switches vanished automatically. `inc` lost 9 switches, namely `-host`, `-port`, `-user`, `-proxy`, `-snoop`, `-[no]pack`, as well as `-sasl` and `-saslmech`. `send` and `post` lost 11 switches each, namely `-server`, `-port`, `-client`, `-user`, `-mail`, `-saml`, `-send`, `-soml`, `-snoop`, as well as `-sasl`, `-saslmech`, and `-tls`. `send` had the switches only to pass them further to `post`, because the user would invoke `post` not directly, but through `send`. All these switches, except `-snoop` were usually defined as default switches in the user's profile, but hardly given in interactive usage.

Of course, those switches did not really ‘‘vanish’’, but the configuration they did was handed over to external MSAs and MRAs. Instead of setting up the mail transfer in mmh, it is set up in external tools. Yet, this simplifies mmh. Specialized external tools will likely have simple configuration files. Hence, instead of having one complicated central configuration file, the configuration of each domain is separate. Although the user needs to learn to configure each of the tools, each configuration is likely much simpler.

Maildrop Formats

With the removal of MMDF maildrop format support, `packf` and `rcvpack` no longer needed their `-mbox` and `-mmdf` switches. `-mbox` is the sole behavior now. [☞ `3916ab6`] In the same go, `packf` and `rcvpack` were reworked (see Sec. XXX) and their `-file` switch became unnecessary. [☞ `ca10237`]

Terminal Magic

Mmh's tools will no longer clear the screen (`scan`'s and `mhl`'s `-[no]clear` switches [☞ `e57b173`] [☞ `943765e`]). Neither will `mhl` ring the bell (`-[no]bell` [☞ `e11983f`]) nor page the output itself (`-length` [☞ `5b9d883`]).

Generally, the pager to use is no longer specified with the `-[no]moreproc` command line switches for `mhl` and `show`/`mhshow`. [☞ `39e87a7`]

`prompter` lost its `-erase` and `-kill` switches because today the terminal cares for the line editing keys.

Header Printing

`folder`'s data output is self-explaining enough that displaying the header line makes few sense. Hence, the `-[no]header` switch was removed and headers are never printed. [☞ `601cc73`]

In `mhlist`, the `-[no]header` switches were removed, too. [☞ `b24f965`] But in this case headers are always printed, because the output is not self-explaining.

`scan` also had `-[no]header` switches. Printing the header had been sensible until the introduction of format strings made it impossible to display the column headings. Only the folder name and the current date remained to be printed. As this information can be perfectly retrieved by `folder` and `date`, consequently, the switches were removed. [☞ `c477dc5`]

By removing all `-header` switches, the collision with `-help` on the first two letters was resolved. Currently, `-h` evaluates to `-help` for all tools of mmh.

Suppressing Edits or the WhatNow Shell

The `-noedit` switches of `comp`, `repl`, `forw`, `dist`, and `whatnow` was removed, but it can now be replaced by specifying `-editor` with an empty argument. [☞ `75fca31`] (Specifying '`-editor true`' is nearly the same, only differing by the previous editor being set.)

The more important change is the removal of the `-nowhatnowproc` switch. [☞ `ee4f43c`] This switch had introduced an awkward behavior, as explained in nmh's man page for *comp*(1):

> The –editor editor switch indicates the editor to use for the initial edit. Upon exiting from the editor, comp will invoke the whatnow program. See whatnow(1) for a discussion of available options. The invocation of this program can be inhibited by using the –nowhatnowproc switch. (In truth of fact, it is the whatnow program which starts the initial edit. Hence, –nowhatnowproc will prevent any edit from occurring.)

Effectively, the `-nowhatnowproc` switch stored a copy of the form file into the draft folder. As '`-whatnowproc true`' causes the same behavior, the `-nowhatnowproc` switch was removed for being redundant. Likely, however, the intention for specifying `-nowhatnowproc` is sending a fully prepared form file at once. This can be done with '`-whatnowproc send`'.

Compatibility Switches

- The hidden `-[no]total` switches of `flist`. They were simply the inverse of the visible `-[no]fast` switches: `-total` was `-nofast` and `-nototal` was `-fast`. I removed the `-[no]total` legacy. [☞ `ea21fe2`]
- The `-subject` switch of `sortm` existed for compatibility only. It can be fully replaced by '`-textfield subject`' thus it was removed. [☞ `00140a3`]

Various

- In order to avoid prefix collisions among switch names, the `-version` switch was renamed to `-Version` (with capital 'V'). [☞ `32b2354`] Every program has the `-version` switch but its first three letters collided with the `-verbose` switch, present in many programs. The rename solved this problem once for all. Although this rename breaks a basic interface, having the `-V` abbreviation to display the version information, isn't all too bad.
- `-[no]preserve` of `refile` was removed because what use was it anyway?

  > Normally when a message is refiled, for each destination folder it is assigned the number which is one above the current highest message number in that folder. Use of the –preserv [sic!] switch will override this message renaming, and try to preserve the number of the message. If a conflict for a particular folder occurs when using the –preserve switch, then refile will use the next available message number which is above the message number you wish to preserve.

- The removal of the `-[no]reverse` switches of `scan` [☞ `8edc5aa`] is a bug fix, supported by the comments ''–[no]reverse under #ifdef BERK (I really HATE this)'' by Rose and ''Lists messages in reverse order with the '–reverse' switch. This should be considered a bug.'' by Romine in the documentation. The question remains why neither Rose and Romine had fixed this bug in the Eighties when they wrote these comments nor has anyone thereafter.

2.2 MODERNIZING

2.2.1 Removal of Code Relicts

The code base of mmh originates from the late Seventies, had been extensively worked on in the mid Eighties, and had been partly reorganized and extended in the Nineties. Relicts of all those times had gathered in the code base. My goal was to remove any ancient code parts. One part of the task was converting obsolete code constructs to standard constructs, the other part was dropping obsolete functions.

REFERENCES

Bourne82.
Stephen R. Bourne, *The UNIX System,* International Computer Science Series, Addison-Wesley, 1982. ISBN: 0201137917

Brooks86.
Frederick P. Brooks, Jr., ''No Silver Bullet: Essence and Accidents of Software Engineering,'' in *Information Processing 1986, the Proceedings of the IFIP Tenth World Computing Conference*, p. 1069–1076, Elsevier Science B.V., Amsterdam, The Netherlands, 1986.

Curry96.
David A. Curry, *UNIX Systems Programming for SVR4,* Nutshell Series, O'Reilly, 1996. ISBN: 1-56592-163-1

Gancarz95.
Mike Gancarz, *The UNIX Philosophy,* Digital Press, 1995. ISBN: 1-55558-123-4

Kernighan84.
Brian W. Kernighan and Rob Pike, *The UNIX Programming Environment,* Prentice Hall, 1984. ISBN: 0-13-937681-X

Kernighan88.
Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language,* Second Edition, Prentice Hall, 1988. ISBN: 0-13-110362-8

Kernighan99.
Brian W. Kernighan and Rob Pike, *The Practice of Programming,* Addison-Wesley, 1999. ISBN: 0-201-61586-X

McIlroy78.
M. D. McIlroy, E. N. Pinson, and B. A. Tague, ''UNIX Time-Sharing System: Foreword,'' *The Bell System Technical Journal*, vol. 57, no. 6, p. 1902, Bell Laboratories, 1978.

ML:edginess.
Paul Vixie, ''edginess,'' *nmh-workers mailing list*, December 26 2011. http://www.mail-archive.com/nmh-workers@nongnu.org/msg02582.html

ML:mmh_ann.
Markus Schnalke, ''Experimental version: mmh,'' *nmh-workers mailing list*, December 8 2011. http://www.mail-archive.com/nmh-workers@nongnu.org/msg02503.html

ML:MTA-MUA.
Thread with the subjects: ''nmh @ gsoc'', ''external MTA'' and ''should nmh be an MTA or an MUA?,'' *nmh-workers mailing list*, January 2010. http://www.mail-archive.com/nmh-workers@nongnu.org/msg01875.html

Peek95.
Jerry Peek, *MH & xmh: Email for Users & Programmers,* O'Reilly, 1995. http://rand-mh.sourceforge.net/book/

RFC 821.
Jonathan B. Postel, *Simple Mail Transfer Protocol,* Request for Comments, 821, IETF, August 1982. http://www.ietf.org/rfc/rfc821.txt

Rochkind85.
Marc J. Rochkind, *Advanced UNIX Programming,* Software Series, Prentice-Hall, 1985. ISBN: 9780130118004

Rose85.
Marshall T. Rose and John L. Romine, ''How to process 200 messages a day and still get some real work done,'' in *Proceedings, Summer Usenix Conference and Exhibition*, Portland, Oregon, 1985.

Salus94.
Peter H. Salus, *A Quarter Century of UNIX,* Addison-Wesley, 1994. ISBN: 0-201-54777-5

Schnalke10.
Markus Schnalke, ''Why the Unix Philosophy still matters,'' Term paper, Ulm University, 2010. http://marmaro.de/docs/studium/unix-phil/

Wolter04.
Jan Wolter, ''DBM Hash Libraries,'' in *Unix Incompatibility Notes*, 2000–2004. http://www.unixpapa.com/incnote/dbm.html

XCU92.
''Commands and Utilities (XCU), Issue 4,'' in *CAE Specification*, The Open Group, July 1992. ISBN: 1-872630-48-0

XVS87.
''XVS Commands and Utilities,'' in *X/Open Portability Guide*, vol. 1, January 1987. ISBN: 0-444-70174-5