# Mutt: 阅读邮件列表

2015-08-19
09:42:04

Mutt是一款优秀的邮件客户端工具, 但是在配置使用上对初学者并不友好. 本文通过讲述为什么使用Mutt, Mutt是什么, 如何配置Mutt来尽可能全面的描述Mutt, 而非仅仅简单的介绍如何配置Mutt.

## 前言

网络上关于介绍Mutt的文章很多, 相信最有名的应该算是王垠的Mutt使用指南了吧, 这是我看过诸多讲述Mutt文章中讲的最好的, 为什么? 因为这篇文章真的只讲Mutt啊! 当然, 最后我还是无法根据这篇文章把我的Mutt配置用起来就是了. 除此之外, 网上遍布了各式各样的形如`(从零配置)?mutt+(getmail|fetchmail)+(msmtp|esmtp)(+procmail)?收发邮件`之类的文章, 那为什么我还要再造一个轮子, 哦不, 是再写这一篇文章来给互联网添加冗余的信息呢?

- 首先, 介绍下我的使用场景, 我需要使用阅读相关社区的Mailing List, 而邮件列表这种东西大家或多或少了解过, 主要的特点是邮件主题(Thread)多, 回复(Re)多, 引用(Quoted)多, 代码(Patch)多...如果你订阅(Subscribe)了一个邮件列表, 那么一天收个一百来封邮件很正常. 那么问题就来了, 这么多这么乱的邮件, 如何阅读? 如何整理? 我们常用的邮箱诸如Gmail, 163, QQmail etc. 这类的邮箱在日常使用中确实表现不错, 但是碰到邮件列表, 似乎就不怎么好用了. 此外, 还有一些邮件客户端, 诸如Outlook, Thunderbird etc. Thunderbird我没用过就不置评了, Outlook看邮件列表我没试过, 但是就是在Outlook中多几层引用回复我看起来就很费劲了, 可能是比较low吧, 并没有学习如何高效使用神器Outlook = =.

- 其次, 除了邮件又杂又多的问题, 在社区的开发者列表(devel)中经常会包含很多代码, 其中有讨论代码的, 也有是使用`git sendmail`发送的Patch的, 因此邮件内容中会包含很多的代码, 其中的代码还是`git diff`格式的, 增减的代码行首有`+`有`-`, 如何能高亮修改的代码行以及其它信息是高效阅读邮件的关键, 毕竟, 时间就是金钱, 我的朋友 ;-P

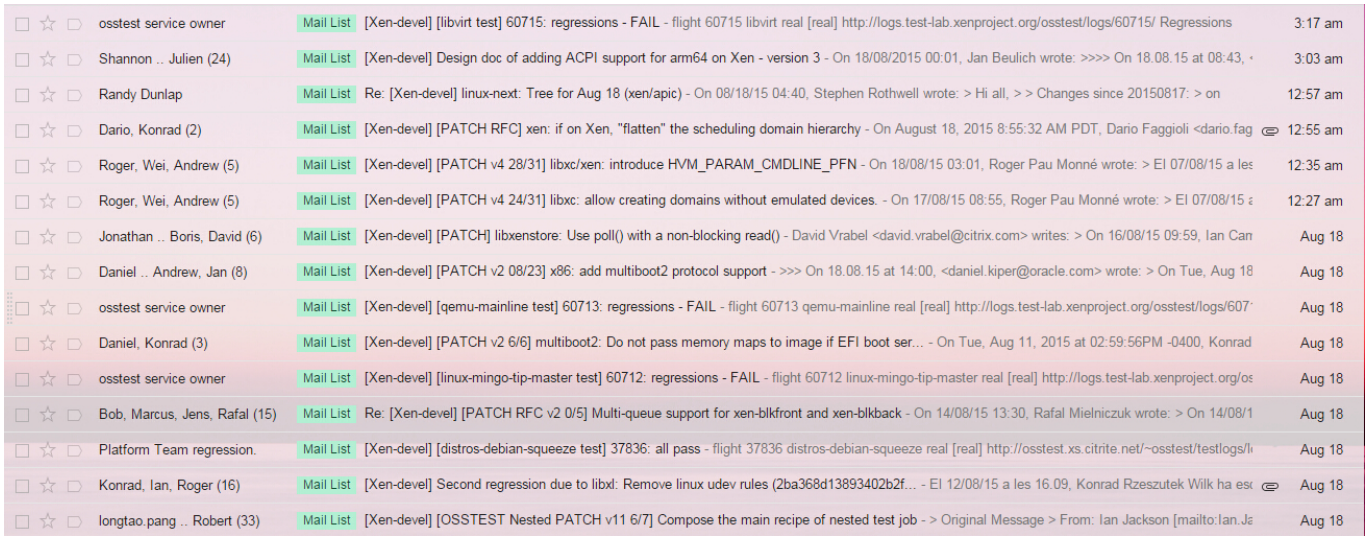
- 最后, 当我在网上搜索Mutt的相关介绍时候, 其实我是想找到如何配置Mutt的, 而搜索引擎返回给我的结果也确实是如何配置Mutt的, 但是很遗憾的, Mutt这个工具对于刚接触的人来说实在是太不友好了. 我只想收发邮件, 为什么还要整那么多没用的, 又是getmail, 又是msmtp的, 还有个看起来巨复杂的procmail? 然后当我困惑于这些个东西是什么关系的时候, 文章只是甩了我几个配置文件就完了? 当我半知不解的照猫画虎按照文章内容小心翼翼的编辑完配置文件后, 发现不能用亦或是根本就不对的时候, 我的内心是崩溃的(摔! 谁能告诉我配置文件的参数是什么意思, 为什么要配这个参数, 这个参数还有什么其它值, 行为是怎么样的, 还有没有其它的参数来完成XXX功能? 而当我苦苦在一篇篇文章中苦苦上下求索一无所获满头雾水的时候, 所幸最后Stackoverflow, 官网manual以及man page救了我.

综上, 可以看到, 我需要用Mutt来阅读邮件列表, 之所以选Mutt而不使用传统的邮件客户端或是常用的邮箱是因为其在阅读邮件列表时表现不佳, 缺乏效率. 而当我要着手配置Mutt的时候, 网络上的诸多文章仅仅只能起到参考的作用, 我希望能做到自己来定制Mutt并且是真的了解为什么这么配才去配, 而不是随手拿来一个配置文件就用. 再者, 就是本人有些许强迫症, 像Mutt和其它的一些在Unix-like环境下运行的Tools具有极强定制性的工具, 总是希望能将其尽可能的配置到顺手为止. 因此, 我才花时间来写下这篇文章, 一方面记录自己的配置过程, 方便以后回顾用, 另一方面, 若能顺便帮助下有需要的人, 那也算是意外收获了吧.

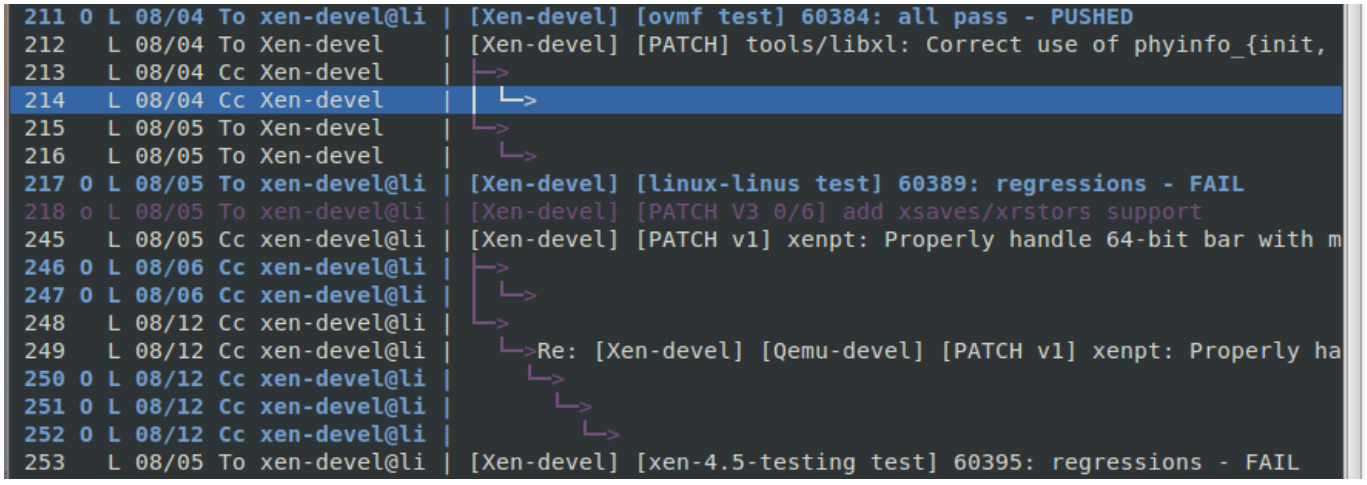
# Why: 为什么使用Mutt

如上文所述, 从阅读邮件的角度来说, 有很多很好用的邮箱, Gmail就是本人很喜欢的邮箱, 并且一直在用, 除了一些不可控的原因影响其用户体验外, 其它方面确实表现的很好. 那为什么我还要花这么大的力气去配置Mutt呢? Mutt跟其它的邮箱相比, 到底有哪些好处呢? 接下来我们来对比下集中应用场景(主要针对邮件列表阅读).

1. **Thread视图**




Gmail中的邮件列表视图




Mutt中的邮件列表视图

从上图的对比中可以看到, Gmail中的邮件列表视图是基于发信时间的, 虽然这样可以查看到最新的邮件, 但是邮件列表的上下文逻辑就看不清楚了. Mutt则是基于邮件主题的模式进行排列的, 每封邮件都按照Thread的前后回复逻辑组织在一起, 一眼看去回复关系非常清楚, 而且Mutt还能设置高亮, 例如在上图的配置中, 将未读的邮件设置成高亮的蓝色, 已读的邮件设置成白色, 未读的Thread设置成高亮蓝色, 部分已读部分未读的Thread设置成紫色, 这些高亮都是可以自定义的, 因此打开Mutt的邮件列表视图, 一眼看去就知道哪些邮件是已经看过的, 哪些邮件是完全没看过的以及哪些邮件是只看一部分未看全的.

2. **嵌套引用**

**Ian Campbell** ian.campbell@citrix.com via lists.xen.org
to Wei, George, xen-devel, Luis, Luis

On Thu, 2015-08-06 at 11:56 +0100, Wei Liu wrote:
> On Thu, Aug 06, 2015 at 11:48:15AM +0100, Ian Campbell wrote:
> > On Thu, 2015-08-06 at 11:17 +0100, Wei Liu wrote:
> > >
> > > I don't think so. Though the doc is not clear on how we should use
> > > those
> > > APIs, I got my idea from
> > >
> > > http://0pointer.de/public/cups-patch-core.txt
> > >
> > > which doesn't call sd_booted.
> >
> > OK, FM then: Acked-by: Ian Campbell <ian.campbell@citrix.com>
> >
> > As a future cleanup I think some of the ifdeferry could be removed by
> > having systemd = false ifndef SYSTEMD_ENABLED, what do you think?
>
> Maybe. In the end there has to be some ifdeferry some where to gate the
> implementation. As a future cleanup I think I can provide a stub
> systemd_checkin() which always returns false and remove all ifdef in
> main().

Right, that's what I meant, i.e. just reduce the amount of ifdef.

...

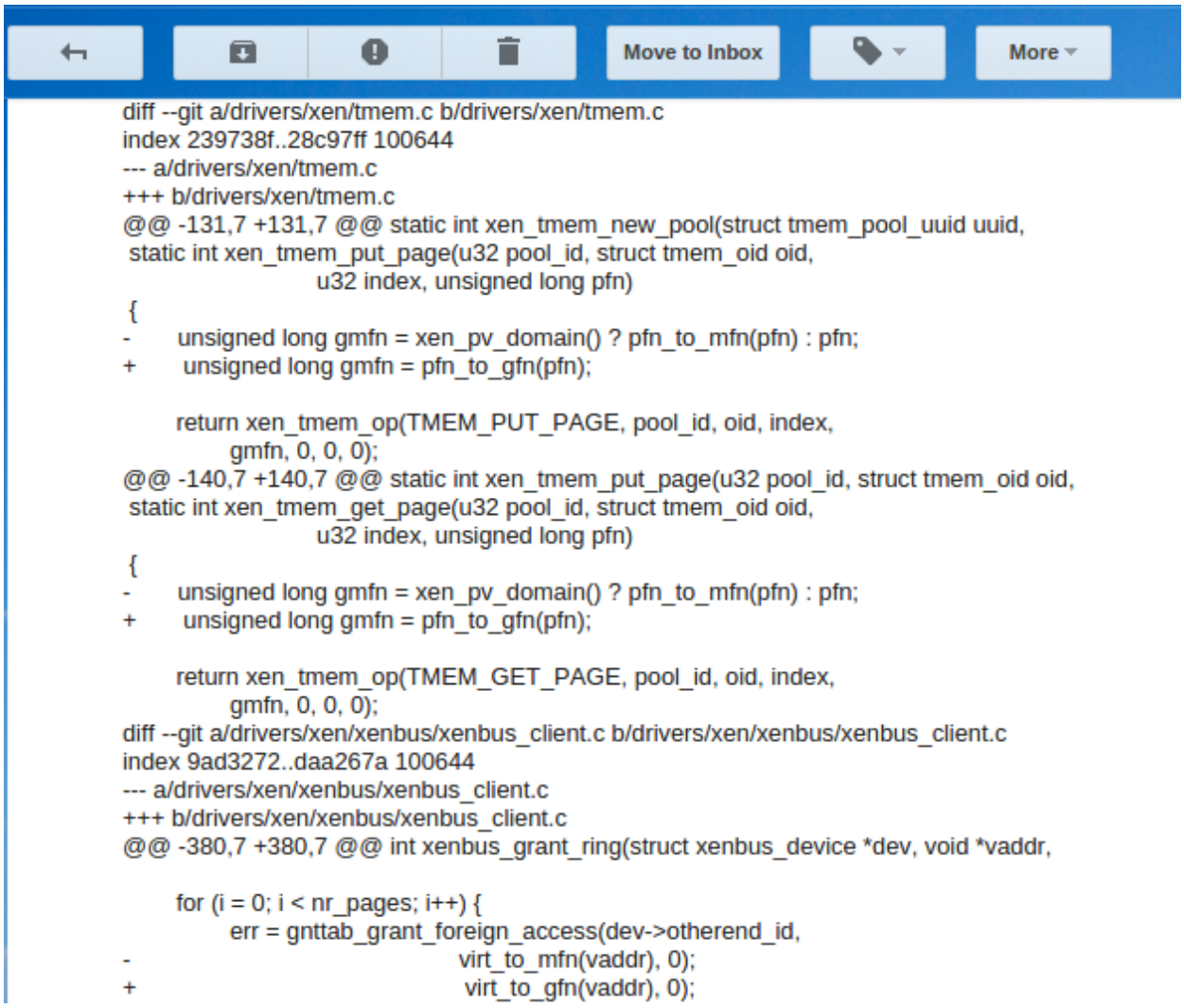
Gmail中的嵌套引用

```
Date: Thu, 6 Aug 2015 12:03:00 +0100
From: Ian Campbell <ian.campbell@citrix.com>
To: Wei Liu <wei.liu2@citrix.com>
Cc: George Dunlap <George.Dunlap@eu.citrix.com>, xen-devel <xen-devel@lists.xenproject.org>,
        "Luis R. Rodriguez" <mcgrof@suse.com>, "Luis R. Rodriguez" <mcgrof@do-not-panic.com>
Subject: Re: [Xen-devel] [PATCH v7 2/8] cxenstored: add support for systemd active sockets
X-Mailer: Evolution 3.16.3-1

On Thu, 2015-08-06 at 11:56 +0100, Wei Liu wrote:
> On Thu, Aug 06, 2015 at 11:48:15AM +0100, Ian Campbell wrote:
> > On Thu, 2015-08-06 at 11:17 +0100, Wei Liu wrote:
> > >
> > > I don't think so. Though the doc is not clear on how we should use
> > > those
> > > APIs, I got my idea from
> > >
> > > http://0pointer.de/public/cups-patch-core.txt
> > >
> > > which doesn't call sd_booted.
> >
> > OK, FM then: Acked-by: Ian Campbell <ian.campbell@citrix.com>
> >
> > As a future cleanup I think some of the ifdeferry could be removed by
> > having systemd = false ifndef SYSTEMD_ENABLED, what do you think?
>
> Maybe. In the end there has to be some ifdeferry some where to gate the
> implementation. As a future cleanup I think I can provide a stub
> systemd_checkin() which always returns false and remove all ifdef in
> main().

Right, that's what I meant, i.e. just reduce the amount of ifdef.
```

Mutt中的嵌套引用

在邮件列表的阅读中, 经常会出现多层的嵌套引用, 从上图的对比中可以看出, Gmail并未对嵌套的引文做任何处理, 而Mutt则可以使用不同的颜色区分不同层的引文, 引用次序非常清晰明了.

3. **Patch邮件**




Gmail中的Patch邮件

```
Signed-off-by: Julien Grall <julien.grall@citrix.com>
Cc: Konrad Rzeszutek Wilk <konrad.wilk@oracle.com>
Cc: Boris Ostrovsky <boris.ostrovsky@oracle.com>
Cc: David Vrabel <david.vrabel@citrix.com>
---
 drivers/xen/tmem.c | 21 +++++++--------------
 1 file changed, 7 insertions(+), 14 deletions(-)

diff --git a/drivers/xen/tmem.c b/drivers/xen/tmem.c
index 28c97ff..e0c8dc7 100644
--- a/drivers/xen/tmem.c
+++ b/drivers/xen/tmem.c
@@ -129,21 +129,17 @@ static int xen_tmem_new_pool(struct tmem_pool_uuid uuid,
 /* xen generic tmem ops */

 static int xen_tmem_put_page(u32 pool_id, struct tmem_oid oid,
-                            u32 index, unsigned long pfn)
+                            u32 index, struct page *page)
 {
-       unsigned long gmfn = pfn_to_gfn(pfn);
-
        return xen_tmem_op(TMEM_PUT_PAGE, pool_id, oid, index,
-                          gmfn, 0, 0, 0);
+                          page_to_gfn(page), 0, 0, 0);
 }

 static int xen_tmem_get_page(u32 pool_id, struct tmem_oid oid,
-                            u32 index, unsigned long pfn)
+                            u32 index, struct page *page)
 {
-       unsigned long gmfn = pfn_to_gfn(pfn);
-
        return xen_tmem_op(TMEM_GET_PAGE, pool_id, oid, index,
-                          gmfn, 0, 0, 0);
+                          page_to_gfn(page), 0, 0, 0);
 }
```

Mutt中的Patch邮件

在开源社区, 在发送Patch的时候常常使用`git sendmail`来发送Patch, 而不是以附件的形式. 我们可以看到, 在Gmail中将Patch中代码的部分直接当做正文来处理, 因此很难看清楚到底Patch中对哪里做了修改, 而在Mutt中, 可以通过正则表达式的方式来匹配正文中的任意字段实现高亮, 将其应用到Patch邮件的正文中, 将增减的行高亮出来, 这样就能很清楚的看明白Patch到底对哪里进行了修改.

以上例子仅仅只是针对邮件列表阅读这种特殊的应用场景而言, 之所以拿Gmail来比较是因为Gmail本身非常优秀, 也是本人最喜欢的邮箱, 并没有贬低Gmail的意思, 只是应用场景不同罢了. 事实上, 除了邮件列表和工作邮件, 本人的其它私人邮件使用的都是Gmail, 一方面UI设计简洁大方, 另一方面, 邮件搜索过滤对于日常应用来说确实非常好~

在弄清楚了应用场合之后, 我们来进入正文...

# What: Mutt是什么

## Mutt简介

Mutt是什么? 或者说Mutt是什么样子的? 根据Mutt官网上的介绍, Mutt是Unix系统环境下一个小巧但是强大的文本邮件客户端. 小是显而易见的, 功能强大对于本人而言主要体现在以下几个Mutt Features:

- 支持配色
- 对邮件列表支持很好
- 高度可定制, 支持键绑定和宏
- 支持正则表达式以及内部模式匹配等多种搜索
- 高效

此外, Mutt还支持其它很多特性, 比如MIME(这是什么我并不知道, 估计也不会用到), PGP(没有加密邮件的习惯), 支持POP3和IMAP协议(是个邮箱都支持), 完全控制邮件头部等等等等...但是这些特性其它的邮箱也有, 有的甚至做的更好.

因此, 对于Mutt, 我最看重的还是其它邮箱所没有的特性.

1. 高度可定制化, 键位绑定和宏对于习惯了Vim和CLI的人来说能提高不少效率;

2. 对邮件列表支持完善, 这也是我之所以选用Mutt最关键的原因, Mutt针对邮件列表添加了很多很方便的设置和操作, 极大的提高了阅读和回复邮件列表邮件的效率;

3. 邮件内容支持配色, 这对于阅读带有代码或是多层嵌套引用的邮件来说简直是神技.

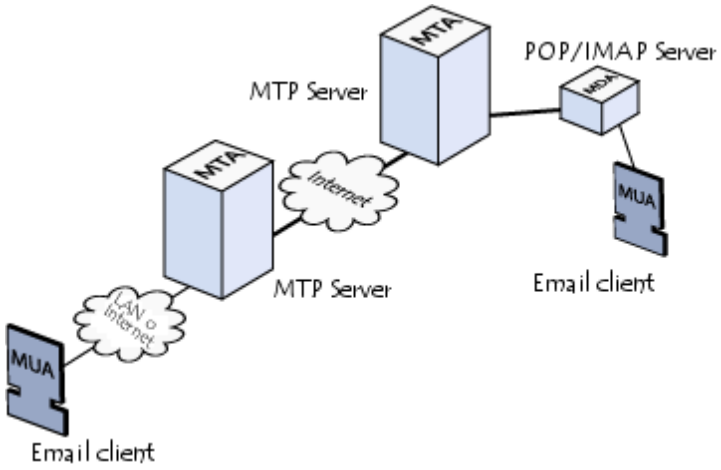
4. 高效, 主要反应在打开一个1000+邮件的信箱只需要1s不到时间, 很快.

诚然Mutt有很多优点, 但是Mutt仅仅只是一个邮件客户端, 那么问题来了, 什么是邮件客户端(Mail Client)? 通常我们认为一个邮件客户端应该像Outlook和Thunderbird那样, 能收能发能整理能查找. 但是很遗憾的, Mutt跟他们并不一样, Mutt是Unix环境下的small and powerful的邮件客户端, 按照Unix大多工具的尿性, 一般只会提供小而精的功能, 并不会做大而全的封装. Mutt也是一样, **Mutt只管理邮件, 而不负责邮件的收发**. 详细点说, Mutt做的事情只是从本机上的某个位置读

取邮件, 或是把邮件存放到本机上的某个位置, 此外, 还负责邮件的阅读, 查找, 标记等事务, 与其说Mutt是一个邮件客户端, 不如说**Mutt是一个邮件管理工具**.

# Email工作原理

说到这里, 我们就不得不简单的讲一下Email的工作原理了[?].




Email的工作原理

从上面的原理图中, 我们可以看到整个邮件发送接收的过程. 首先, 邮件在MUA(Mail User Agent)中编辑完成, 交由MTA(Mail Transfer Agent)进行发送, 在发送的过程中, 邮件会在多个路由和MTA服务器中中转, 邮件从一个MTA服务器被传输到另一个MTA服务器, 其中使用的是名为SMTP的邮件传输协议. 当邮件到达收件人所处的网络中的MTA时, MTA将邮件交给邮件服务器, 邮件服务器负责将邮件发送给指定用户的信箱中, 邮件服务器在分发邮件的时候, 具有MDA(Mail Delivery Agent)的功能, 其中可能会包含防火墙, 过滤垃圾邮件, 屏蔽黑名单等功能, 我们常见的网络邮箱的工作模式大致是这样的.

另外, 如果我们使用的是邮件客户端来接收邮件的话, 客户端还需要负责从远端的信箱中拖取邮件到本地的信箱中. 这里就需要使用MRA(Mail Retrieval Agent)来完成从远端信箱收取邮件至本地的操作, 在MRA取邮件的时候, 涉及到这么一个问题, 是单纯的将远端邮箱的邮件复制来呢, 还是让本机的信箱跟远端的信箱保持同步呢, 根据收取方式的不同, 区分了两种主流的收取邮件协议POP3和IMAP. 在邮件到达本地后, 有时在本机上, 会再次对邮件进行一部分分拣和过滤, 例如将来自邮件列表的邮件放到一个特定的文件夹中, 将包含特定文字的邮件放到垃圾邮件文件中等, 这些功能, 是通过本地的MDA来完成的. 最后, 邮件到达了特定的文件夹, MUA, 即邮件客户端从特定文件夹读取邮件, 并解码邮件格式, 展示给用户阅读. 而Mutt, 在整个邮件收发过程中, 做的也就是这个部分的事情!

# Mutt与msmtp, getmail和procmail的关系

解释了半天, 相信大家已经被各种M*A给整懵了吧. 以下对邮件发送过程中的各个部分(Agent)做下简要的解释:

- **MUA(Mail User Agent)**: 邮件客户端, 负责邮件的管理, 阅读等. 一些集成度较高的MUA会带有收发邮件乃至过滤邮件的功能如Outlook和Thunderbird, Mutt也带有简单的发送邮件功能, 不过一般不采用.

- **MTA(Mail Transfer Agent)**: 邮件传送代理, 负责邮件的发送, 处理发送过程中的主机识别, 路由跳转等问题. 对于用户而言MTA就是负责从本地发送邮件的部件. 常见的MTA有sendmail, emstp, msmtp.

- **MRA(Mail Retrieval Agent)**: 邮件收取代理, 负责从远端的邮件服务器收取邮件至本地文件夹. 常见的MRA有fetchmail, getmail.

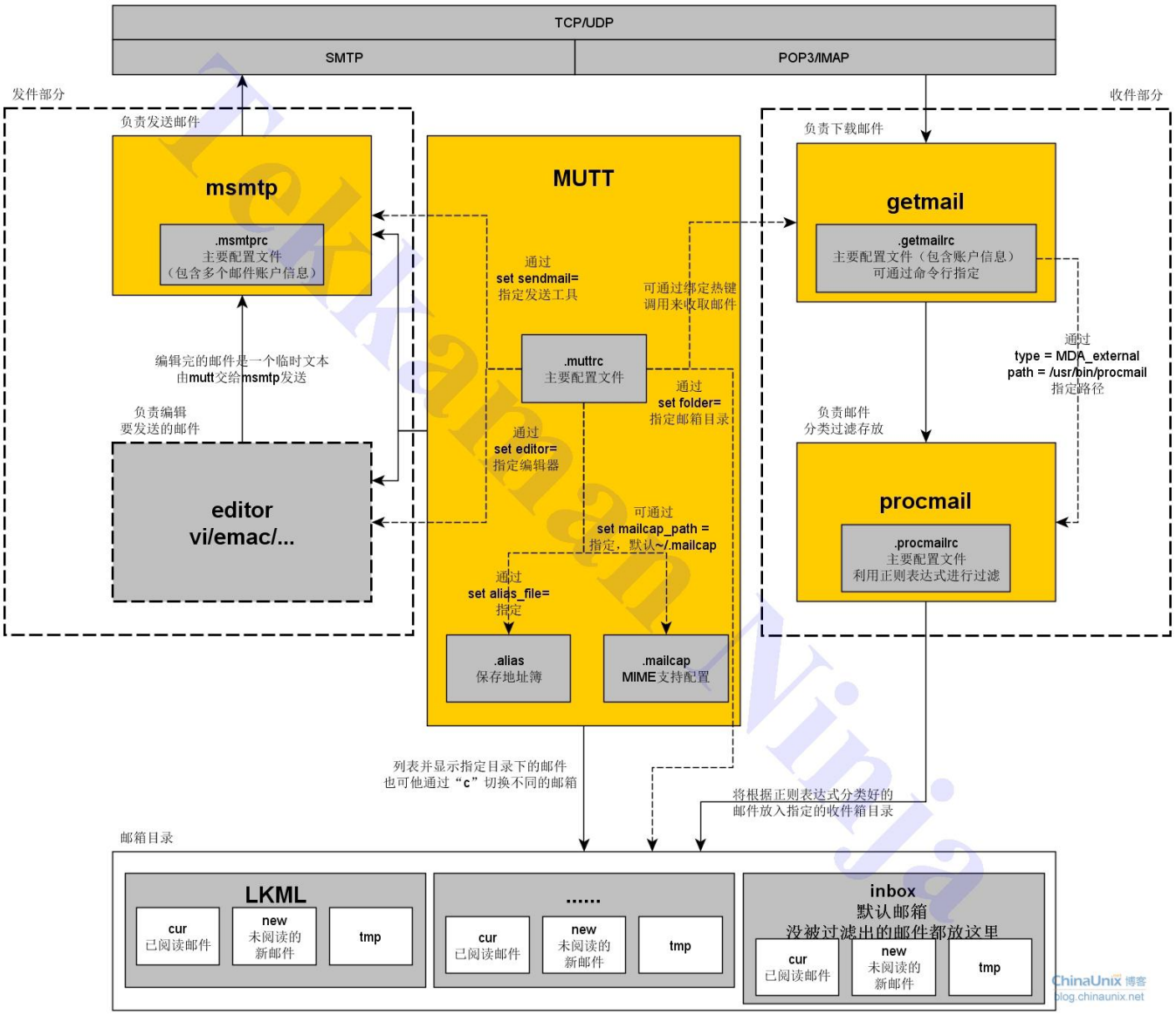
- **MDA(Mail Delivery Agent)**: 邮件分发代理, 负责根据规则过滤邮件并将邮件投放到不同的文件夹中. 常见的MDA有procmail.

之所以讲了这么多Email的工作原理, 主要是为了两件事情.

1. 明确Mutt在整个邮件收发过程中的位置.
2. 解释为什么在配置Mutt的时候需要额外安装配置msmtp, getmail乃至procmail.

当弄明白了Email的工作原理后, 其实也就不难理解为什么在网上搜索Mutt相关资料的时候, 跳出来的都是mutt+getmail+msmtp+procmail之类的文章了. 因为光一个Mutt, 根本无法完成最基本的邮件收发功能啊, 必须需要靠msmtp来发邮件, 靠getmail来收邮件, 如果还需要自定义一些邮件分类过滤行为的话, 还需要procmail来帮忙. 因此, 现在我们在谈论Mutt的时候, 我们其实实在谈论mutt+getmail+msmtp这一串东西.

然后, 我想再来看看以下这张图[?], 就明白Mutt与getmail, msmtp以及procmail之间的关系了.




Mutt与相关工具的关系(图片版权为tekkamanninja所有)

# How: 搭建Mutt环境

在明白了为什么用Mutt, 什么是Mutt, 已经Mutt与msmtp, getmail和procmail的关系之后, 接下来, 我们来了解下如何配置Mutt以及msmtp和getmail, 在这节中我们先不介绍procmail, 因为很多的邮箱反垃圾邮件和防火墙已经做的很好了, 用不着用户自定义规则来反垃圾邮件.
Mutt+msmtp+getmail已经可以完成基础的收发管理邮件的功能.

## 安装Mutt, msmtp和getmail

这步没什么好说的, 直接apt-get

```
1  $ sudo apt-get install mutt msmtp getmail4
```

# 配置Mutt

Mutt的配置主要分为2个部分: 一个是muttrc文件, 另一个是信箱目录的组织.

## muttrc文件

我们先来看一下基本的muttrc配置文件. 在用户的`$HOME`目录下创建并编辑`.muttrc`文件.

```
$ vim ~/.muttrc
```

一个本的muttrc文件如下所示:

```
# 设置发信人名称
set realname = "Your Name"

# 设置邮件发送程序
set sendmail = "/usr/bin/msmtp"

# 默认的发信地址
set from = "yourmail@mail.com"

# 在发送邮件时使用from域作为msmtp发送邮件的sender, 否则使用的是user@localhost
set envelope_from = yes

# mutt自动生成from地址
set use_from = yes

# 设置编辑时使用的编辑器
set editor = "vim"

# 建立信箱, 默认信箱是inbox, 所有新邮件都会被默认发送到inbox文件夹中
# 告知mutt邮件的存放格式(format), maildir格式要求相关文件夹下必须有tmp, new,
set mbox_type = maildir
# 存放邮件的根目录, 在muttrc文件中用"+"或者"="表示
set folder = "~/.mail"
# 接收到的邮件默认存放的位置, 系统会自动创建该目录, 而其它的目录需要手动创建
set spoolfile = "+inbox"
# 从收件箱(inbox)保存的邮件存放位置
set mbox = "+mbox"
# 延迟发送(postpone)邮件的存放位置
set postponed = "+postponed"
# 已发送的邮件存放的位置
```

```
set record = "+sent"

# 设置终端显示时采用的字符集
set charset = "utf-8"

# 设置发送字符集，根据linux内核邮件列表推荐设置
set send_charset = "us-ascii:utf-8"

# 回信时引用原文是否加入原文的邮件头
set header = no

# 将inbox中已读邮件自动移动到mbox文件夹中(ask-yes表示会询问，yes则直接移动)
set move = ask-yes

# 在信件内容窗口(pager)滚动到底部时，不自动跳到下一封邮件
set pager_stop

# 告知mutt订阅的邮件列表,以便开启相关特性,便于阅读邮件列表
subscribe xen-devel@lists.xen.org xen-devel@lists.xensource.com xen-devel
```

## 指定msmtp作为邮件发送工具

由于我们采用msmtp来发送邮件, 需要告知Mutt, 以便在发送邮件的时候, Mutt可以将邮件交给msmtp进行发送. 因此, 需要指定邮件发送工具路径.

```
set sendmail = "/usr/bin/msmtp"
```

**通过上述配置, 我们将Mutt和msmtp给联系起来了.**

## 设置信箱目录

另外, 我们看到在muttrc配置文件中设置了很多信箱文件夹, 如下:

```
set mbox_type = maildir
set folder = "~/.mail"
set spoolfile = "+inbox"
set mbox = "+mbox"
set postponed = "+postponed"
set record = "+sent"
```

这里就有必要解释下其意思了. 首先, 我们看到的是`mbox_type`这个变量, 这是用来设置邮件储存格式的, 通常将其设置为`maildir`格式[?]. `maildir`格式要求每个信箱文件下要包含3个子文件夹, 分别为`cur`, `new`, `tmp`.

说到邮件储存格式, 我们就来简单了解下Mutt管理邮件的方式. 事实上, 我们通过Mutt看到的邮件, 都是以文件的形式保存在特定的目录中的, 一封邮件就是一个文件. Mutt做的事情就是从特定的信箱目录中读取邮件, 解码文件, 然后展示内容给用户. 在把邮件从一个信箱移动到另一个信箱的时候, Mutt实际上是将邮件文件从一个目录移动或是复制到另一个目录中, 就是这么简单!

例如如果inbox信箱中有邮件, 那么可以看到:

```
$ ls ~/.mail/inbox/cur
1439889842.29259_1.USER:2,S  1439890047.29915_0.USER:2,S  1439890260.30560
1439889842.29260_1.USER:2,S  1439890047.29916_0.USER:2,   1439890260.30561
1439889843.29261_0.USER:2,S  1439890047.29917_0.USER:2,   1439890261.30562
```

也可以尝试着将一个文件`mv`到另一个文件夹的`cur`子目录下, 再打开Mutt看看邮件是不是就从原先的邮箱移动到新的邮箱中了 0.0

接着我们看一下变量`$folder`, 该变量即指定了Mutt读取操作邮件的"根目录", 例如, 示例配置文件中, 将其设置为`~/.mail`目录, 那么在打开Mutt的时候, Mutt会默认从该目录下去读取操作邮件文件.

**在信箱组织方面, 一般建议设置`$spoolfile`, `$mbox`, `$postponed` 和 `$record`这四个变量.** 当然, 也可以都不设置, 但是Mutt会默认在`$folder`变量指定的目录下建立`inbox`目录, 并将全部的邮件都一股脑放到这个目录下. 这里, 我们都对其进行了设置, 当然, 使用前要确保这些指定目录已经建立好了, 并且每个目录下已经建立了`cur`, `new`, `tmp`这三个子目录.

```
$ mkdir ~/.mail
$ cd ~/.mail
$ mkdir inbox mbox postponed sent
$ cd ~/.mail/inbox
$ mkdir cur new tmp
...
```

这里我们简要的介绍下各个目录的功能[?]

**muttrc中变量　设置的目录　功能**

| muttrc中变量 | 设置的目录 | 功能 |
| --- | --- | --- |
| `$spoolfile` | inbox | 最基本的一个目录, 默认所有的新邮件都会存放在这个目录下 |
| `$mbox` | mbox | 已读邮件会被移动至该目录下(需要设置`$move`变量) |
| `$postponed` | postponed | 推迟发送的邮件会被保存在这里 |
| `$record` | sent | 已发送的邮件会被保存在这里 |

P.S. 在指定信箱目录的时候我们使用了`+`符号, 在muttrc文件中, `+`号代表`$folder`变量, 例如示例中设置了`$folder`变量为`~/.mail`目录, 那么`+inbox`表示的就是`~/.mail/inbox`目录[?].

## Mutt其它配置项

```
1  set move = ask-yes
```

`$move`变量有四个值, 分别为`yes`, `no`, `ask-yes`和`ask-no`. `$move`变量的意义在于指定是否将inbox信箱中已读的邮件自动移动到mbox目录中. 默认是不移动, 此处将其设置为`ask-yes`, 即表示自动将已读邮件移动至`mbox`目录中, 但是在移动前会提示用户是否执行. 这样可以保持`inbox`目录的整洁, 避免每次打开mutt, 上来就看到一堆已经读过的邮件.

```
1  set charset = "utf-8"
2  set send_charset = "us-ascii:utf-8"
```

这里主要是字符集的设置, 因为本人使用Mutt来阅读以及回复邮件列表, 因此需要设置通用的字符集来避免收信人接受到乱码邮件的问题, 此处的设置是参考Linux内核邮件列表对邮件客户端的建议设置

关于其它项的配置, 注释已经写的比较详细了, 就不再赘述了. 最后, 实际上Mutt是带有简单的收发邮件模块的, 因此有些用户可能需要在`.muttrc`文件中写入邮箱的账户和明文密码, 因此, 最好还是需要将`.muttrc`的权限设置为600.

```
1  $ chmod 600 ~/.muttrc
```

# 配置msmtp

msmtp是一个SMTP客户端[?]. 在整个Mutt收发系统中负责发送邮件的功能.

**msmtp的配置文件为`~/.msmtprc`.**

```
$ vim ~/.msmtprc
```

一个基本的msmtprc文件如下:

```
defaults
tls off
tls_certcheck off
logfile ~/.mail/msmtp.log

account your_account
host $smtp.your_smtp_server.com
from $your_mailaddress@mail.com
port 25
user $username
password $password

# Set a default account
account default : $username
```

注: 文件中带`$`符号的都是需要根据具体情况修改的信息.

由于msmtp是负责发送邮件的, 而之前我们说过, 邮件发送涉及到的协议主要是SMTP, 因此, 需要告知msmtp SMTP服务器的host地址, 一般的邮箱STMP地址都比较简单, 例如新浪邮箱的host为`smtp.sina.com`, Gmail的host为`smtp.gmail.com`. 而大部分邮箱的SMTP服务端口号为默认的25, Gmail的话使用的是465和587, 具体的还请查看官方的介绍.

**设置权限, 保护配置文件**

在知晓了SMTP服务器地址后, msmtp会访问SMTP服务器, 此时需要登录账户密码, 因此, 这些信息也要写入msmtprc以告知msmtp, 需要注意的是, 密码是明文保存的, 因此, 安全起见, 需要设置msmtprc文件的权限.

```
$ chmod 600 ~/.msmtprc
```

有关于msmtp更详细的配置, 请参见官方文档

P.S. 本人使用的是公司的邮箱, 因此需要关闭TLS. 但是如果使用的Gmail则需要相关的设置, 具体方法网上随便一搜都是, 不再赘述.

在配置msmtp后, 不妨测试下在Mutt中能否正常发送邮件:

```
$ echo "测试邮件内容" | mutt -s "测试邮件标题" test_mail@address.com
```

如果在测试邮箱中收到测试邮件, 即表明msmtp配置成功, 并且Mutt能顺利调用msmtp来发送邮件.

# 配置getmail

getmail是一个邮件收取工具, 用于从不同的邮件服务器上收取用户不同账户的邮件. 同时getmail也被设计来取代其它的一些邮件收取工具例如fetchmail[?].

在配置getmail前, 需要先建立getmail相应的目录`~/.getmail`, 之后在目录下新建并编辑配置文件getmailrc. 由于配置文件中包含邮箱账户以及明文密码, 因此限制`.getmail`目录的权限.

```
$ mkdir -m 700 ~/.getmail
$ vim ~/.getmail/getmailrc
```

P.S. 如果你有多个邮箱, 需要在`~/.getmail`目录下为每个邮箱都建立一个getmailrc文件, 可以通过后缀来区分不同的getmailrc文件, 例如`getmailrc.gmail`或是`getmailrc.163`, 但需要注意的是, 文件名的前半部分必须是`getmailrc`, 及配置文件的命名规范为`getmailrc.xxx`.

`getmailrc`配置的文件是以节(section)为基础的, `getmailrc`文件至少应该有2个节:

1. `[retriever]`, 用于告知getmail邮箱的账户信息以便getmail能从邮件服务器上收取邮件.
2. `[destination]`, 用于告知getmail如何处理已经收取到的邮件.
   此外, 还可以添加一个额外的`[optional]`节, 包含一些笼统的设置诸如日志文件路径设置等[?]

```
[retriever]
# type = SimplePOP3SSLRetriever
# port = 995
type = SimpleIMAPSSLRetriever
server = $mailserver.com
username = $username
password = $password

[destination]
# 以Maildir格式储存
type = Maildir
```

```
path = ~/.mail/inbox/

[option]
# 默认为True，每次执行getmail收取全部邮件，False表示只收取未收取过的邮件
read_all = False
# 本地删除服务器是否也删除邮件
delete = False
message_log = ~/.getmail/getmail.log
```

注: 上述配置文件中带`$`部分是需要根据实际邮箱信息更改的.

首先让我们先来看一下`[retriever]`节, 关键字段为`type`, 用于告知getmail你打算以哪种协议收取邮件, 这里指的收取协议主要是POP3和IMAP.

- POP3: 从远端邮件服务器上收取邮件, 同时服务器上保留邮件, 本地的对邮件的删除移动不影响到远端邮件服务器. 例如通过POP3协议收取完邮件后, 将本地的邮件删除, 服务器上依旧保存有该封邮件.

- IMAP: 本地邮件与远端邮件服务器上的邮件同步, 也就是说如果本地对邮件执行了删除或者移动也会同步到邮件服务器上.

如果采用POP3协议, 那么就将`type`设置成`SimplePOP3SSLRetrieve`, 如果用IMAP, 就设置成`SimpleIMAPSSLRetriever`. 事实上, 还可以设置成`SimplePOP3Retriever`和`SimpleIMAPRetriever`, 不过带SSL的表示使用SSL加密, 因此会更安全.

关于POP3和IMAP协议收取邮件, 在本人的具体实践过程中, 发现并不是严格的遵守其协议行为, 即便是采用IMAP协议, 在本地删除了邮件的情况下, 邮件服务器上仍旧会保留邮件, 这跟getmail的配置有关, 同时可能还和邮件服务器的配置有关.

注意到在上述的rc文件的`[option]`节中, 包含字段`delete`的设置, 用于指明在本地删除邮件后服务器是否也删除. 如果设置为`True`, 则在getmail会在成功收取邮件后删除邮件服务器上的对应邮件, 若设置为`False`, 则保留, 默认值是`False`. 该字段可能会影响POP3和IMAP收取协议, 因为本人通过反复设置POP3和IMAP, 以及`detele`字段, 发现不管怎么样邮件服务器上的邮件都会被保留, 但这可能是邮件服务器本身设置的原因, 因此无法验证. 为了保险起见, 建议初次设置getmail的时候采用POP3协议收取, 同时`delete`设置为`False`, 便于熟悉与验证. 不然手一抖把本地的邮件删了, 发现服务器上的邮件也没了就不好玩了. 当然, 也可以自己给自己发几封验证邮件, 然后在本地删除或是移动看看邮件服务器上的变化来验证getmail的配置.

本人在设置协议的时候采用IMAP来收取协议, 主要的原因在于在使用过程中一些邮件服务器的POP3功能不稳定, 会出现一些邮件收不到, 收不及时或是重复收取的情况, 而IMAP则每次都能正确收取邮件, 相对而言, IMAP比POP3稳定[?].

如果你在使用getmail收取邮件时, 出现了重复收取邮件的情况, 不妨在`[option]`节中将`read_all`字段设置为`False`. 表示每次getmail收取邮件的时候只收取未收取过的邮件, 避免重复收取邮件. 那么, 从实现角度, getmail是如何完成识别邮件是否已经收取过了, 从而决定是否要收取邮件呢. 可以发现, 在使用getmail收取邮件后, 在`~/.getmail`目录下会多出一个`oldmail-mailserver.com-port-username-INBOX`的文件, 该文件的作用就是用来记录哪些邮件是已经收取过了的, 从而在下次收取的时候, 避免收取文件中已包含的邮件. 可以做个实验, 将该文件删除, 那么再使用getmail收取邮件的时候, 就又会收取全部的邮件了.

**设置权限, 保护配置文件**

和其它配置文件一样的, 由于getmailrc文件中包含了明文的密码, 因此最好设定权限:

```
1  chmod 600 ~/.getmail/getmailrc
```

关于getmailrc文件的更多信息, 请参见官方文档, 官网同时提供了多种getmailrc文件模板以供参考.

在配置完getmail后, 需要测试下getmail是否正常工作了, 执行getmail收取邮件.

```
1  $ getmail -n
```

`-n`代表只收取新邮件. 执行命令后, getmail将会按照配置文件中的设置, 将邮件投递到`path`所指定的目录`~/.mail/inbox`下. 而该目录在Mutt的配置文件中被指定为`spoolfile`所使用的目录, 即`inbox`收件箱的目录, 因此, 打开mutt, mutt就会自动从该目录下获取邮件, 查看是否收取了邮件来验证getmail是否正常工作. getmail就是通过这个目录将邮件交给Mutt的, 二者的联系, 仅此而已.

**设置crontab定时收取邮件**

getmail的收取邮件是通过执行getmail命令来实现的, 因此, 要想及时获取新的邮件, 就需要定时的执行getmail来收取邮件, 这个任务交给cron守护进程最好不多了.

```
1  $ crontab -e
```

执行crontab命令, `-e`代表对crontab进行编辑, 在crontab的最后一行添加:

```
1  */10 * * * * /usr/bin/getmail -n
```

表示每隔10分钟执行一次`getmail -n`命令来收取邮件.

**至此, 基础的Mutt环境已经搭建完毕, 可以使用Mutt来正常的收发, 查阅, 发送, 回复邮件了.**

---

# Misc: 额外部分

## Vim风格键位绑定

Mutt默认的键位设置以及相关操作, 请参阅: Mutt Manual: Chapter 2

对于习惯了Vim操作的用户而言, 再重新去适应Mutt的一些基本翻页按键着实显得有些浪费时间, 所以不妨按照Vim的习惯对Mutt的键位进行相应的绑定.

绑定设置同样是在`~/.muttrc`中完成的, 在`~/.muttrc`文件中添加:

```
# Vim式键位映射, \c表示ctrl键
bind pager G bottom
bind pager j next-line
bind pager k previous-line
bind pager \cf next-page
bind pager \cb previous-page
bind pager \cj next-entry
bind pager \ck previous-entry
bind pager gg top
bind pager G bottom
bind index gg first-entry
bind index G last-entry
bind index R group-reply
bind index \cf next-page
bind index \cb previous-page
bind index } bottom-page
bind index f change-folder
```

以上是按照个人习惯所进行的键位绑定, 当然, 每个人的习惯是不一样的, 可以自行修改.

注意到绑定命令中出现了`index`和`pager`, 这里稍作下解释, 其中`index`代表信箱的列表页面, 而`pager`为阅读邮件的界面. 指定`index`或是`pager`来使得绑定的键位在对应的页面下生效. 有关键位绑定的更多参考, 请看: Mutt Manual: Changing the Default Key Bindings.

关于Mutt的键位操作, 这里建议在`~/.muttrc`文件中设置`pager_stop`, 来避免当阅读至邮件末尾的时候, 自动跳到下一封邮件.

```
# 在信件内容窗口(pager)滚动到底部时，不自动跳到下一封邮件
set pager_stop
```

# Mutt配色高亮设置

Mutt支持对各个界面进行高亮设置, 包括邮箱的列表部分`index`, 邮件内容部分`body`, 邮件头部信息部分`header`, 甚至连列表模式中的回复箭头颜色都可以设置. 在这部分, 主要分为3个部分, 分别介绍Mutt常规界面的配色, 对Patch补丁添加高亮支持以及嵌套引用中不同层的高亮区分.

## Mutt常规界面配色

在介绍Mutt常规界面的配色之前, 我们先来了解下Mutt中各个界面的名称, 如前文所述, Mutt的界面主要包含有`index`, `body`和`header`, `index`就是打开Mutt时看见的界面, 通过列表的形式——列出邮件主题和收发信息. 而查看某封邮件的内容的界面则属于`pager`. `pager`含有2个部分, 一个是邮件头部, 包含有邮件的头部信息, 这部分属于`header`. 另一部分是邮件内容的主体`body`, 包含邮件的正文. 基本上需要配置的也就这3个地方. 当然, 此外还有其它的界面, 如`File Brower`, `Alias Menu`等, 一般很少见到, 暂时就不配色了. 具体的界面和菜单介绍, 参阅: Mutt Manual: Screens and Menus.

在着手自定义Mutt配色之前, 可以先看一下Github上关于Mutt配色的项目: Solarized Colorscheme for Mutt, 如果觉得合适, 就不需要自己折腾配色了.

事实上, solarized的配色已经相当不错, 但是如果你希望按自己的意愿来更改Mutt配色的话, 接下来我们就来介绍自定义Mutt界面配色方案.

```
# basic colors ---------------------------------------------------
color normal        white           black
color error         red             black
color tilde         white           black
color message       cyan            black
color markers       red             black
color attachment    brightred       black
color search        brightmagenta   black
color status        brightwhite     black
color indicator     white           blue
color tree          magenta         black   # arrows in threads

```

```
# 列表部分 ---------------------------------------------------------
# 高亮不同状态的邮件，具体的pattern(例如~N)参见mutt manual的4.2节

color index         red             black         "~A"
color index         brightred       black         "~E"
color index         brightcyan      black         "~N"
color index         brightcyan      black         "~O"
color index         brightmagenta   black         "~Q"
color index         white           black         "~R"
color index         brightblue      black         "~U"
color index         brightblue      black         "~U~$"
color index         brightblue      black         "~v"
color index         brightblue      black         "~P"
color index         cyan            black         "~p!~F"
color index         brightgreen     black         "~N~p!~F"
color index         brightgreen     black         "~U~p!~F"
color index         green           black         "~R~p!~F"
color index         red             black         "~F"
color index         red             black         "~F~p"
color index         red             black         "~N~F"
color index         red             black         "~N~F~p"
color index         red             black         "~U~F~p"
color index         white           brightmagenta "~D"
color index         white           black         "~v~(!~N!~O)"
color index         magenta          black         "~v~(~N|~O)"
color index         magenta           black         "~N~v~(~N)"
color index         red             white          "~v~(~F)!~N"
color index         yellow          white          "~v~(~F~N)"
color index         green           white          "~N~v~(~F~N)"
color index         green           white          "~N~v~(~F)"
color index         yellow          red            "~v~(~D)"

# 邮件内容页面邮件头部分高亮 ---------------------------------------

# color header        brightcyan      default         "^"
color hdrdefault    brightblue      black
color header        brightgreen     black         "^(From)"
color header        brightyellow    black         "^(Subject)"
```

这里的配色方案是以Solarized为基础的, 稍作修改. 这里对以上的配色配置稍作解释.
首先, 语句的模式是

```
color <界面> <前景色> <背景色> [其它选项]
```

Mutt支持8种颜色, 分别为: Black, Red, Green, Yellow, Blue, Magenta, Cyan, White[?]. 在颜色前面添加`bright`可使得字体加粗, 并使得颜色更鲜艳, 例如red就是正常粗细的暗红色字体, 但是brightred就变为加粗的鲜红字体. 这样, 我们可以使用的颜色就有16种了, 再配合上前景色背景色的各种搭配, 真是变换无穷呢 = =b

对于自定义配色, 没什么好说的, 只要知道是针对`index`配色, 还是`pager`配色或者是`header`就行, 剩下就是改下颜色, 看哪里颜色变了, 就能确定语句控制的是那部分的颜色了.

以上配置中, 还有一些地方相对比较难懂, 例如

```
color header        brightyellow    black        "^(Subject)"
```

这条语句的作用是讲邮件头部的邮件主题(Subject)行显示为黑底高亮黄字. 这里使用了正则表达式来匹配需要高亮的行, `"^(Subject)"`匹配开头是"Subject"的行, 而该匹配模式又只作用与邮件头部`Header`, `Header`部分邮件主题行的行首就是"Subjuect"开头的. 当然, 上述配置中, 将发信人所在行用高亮绿字显示, 此外, 可以采用这种方法, 高亮任意的邮件头部行, 感兴趣的话, 可以在邮件内容页面按下`h`来显示详细的邮件头部信息.

接着我们来看下下面的语句:

```
color index         white           black        "~v~(!~N!~O)"
```

该语句首先针对的是`index`界面, 即列表界面, 然后前景色是白色, 背景色是黑色, 这没什么好说的, 关键在于`"~v~(!~N!~O)"`的解释, 这里的`~N`的表示是Mutt特有的通配符.

| 通配符 | 描述 |
| --- | --- |
| ~A | 所有邮件 |
| ~D | 已删除的邮件 |
| ~F | 已被标记的邮件 |
| ~N | 新的未读邮件 |
| ~O | 旧的未读邮件 |
| ~v | 邮件列表某主题(Thread)中的邮件 |

除了通配符, Mutt还支持有限的逻辑操作, 分别为: 逻辑非符号`!`, 逻辑或符号`|`和组合符号`()`, 逻辑与的话直接挨着就是.

这下回头看上述语句的意思就明白了, 对于符合条件(1. 是邮件列表某个主题包含的邮件 2. 不是新的未读邮件 3. 不是旧的未读邮件)的邮件, 实际上就是已读的列表邮件, 将其所在行的颜色设置为黑底白字.

以上是几个主要的通配符的解释, 以及更多的Mutt模式匹配信息, 请参阅: Mutt Manual: Patterns: Searching, Limiting and Tagging.

另外, 值得一提的是, 该通配符不仅适用与高亮设置, 同样是适用于搜索邮件, 例如, 想要查看当前邮箱下的全部未读的新邮件, 则只要在`index`界面输入`/`, 然后输入匹配模式`~N`回车即可. 当然, 还有更复杂的组合模式, 关于Mutt的查找模式匹配, 参见: Mutt Manual: Simple Searches.

我们在上面介绍了`index`界面和`header`界面的配色, 关于邮件内容主体`body`的配色也是一样道理. 例如, 可以将邮件正文中的表情符号显示成蓝色:

```
color body blue black "[;:][-o][)/(|]"
```

这样, 正文中像`:-)`, `;-o`, `:-(`这类的卖萌表情就统统被高亮出来了 0.0

## Patch高亮支持

对于混迹开发者列表的同学来说, 将Mutt设置成支持Patch高亮绝对是一劳永逸的一件事情之一, 但是对于普通邮件用户而言这并没有什么卵用...

```
# 自定义的patch补丁高亮，方便查看patch
color   body     red           black    "^-.*"
color   body     green         black    "^[+].*"
color   body     brightwhite   black    "^diff --git.*"
color   body     brightwhite   black    "^index [a-f0-9].*"
color   body     brightwhite   black    "^\-\-\- a.*"
color   body     brightwhite   black    "^[\+]{3} b.*"
color   body     brightyellow  black    "^@@.*"
color   body     brightmagenta black    "^(Signed-off-by).*"
color   body     brightmagenta black    "^(Reported-by).*"
color   body     brightmagenta black    "^(Suggested-by).*"
color   body     brightmagenta black    "^(Acked-by).*"
color   body     brightmagenta black    "^(Reviewed-by).*"
color   body     brightmagenta black    "^\-\-\-$"
```

```
# color   body     brightmagenta  black    "^(Cc).*"
# color   body     brightmagenta  black    "^(CC).*"
color   body     white          black    "^( \#define).*"
color   body     white          black    "^( \#include).*"
color   body     white          black    "^( \#if).*"
color   body     white          black    "^( \#el).*"
color   body     white          black    "^( \#endif).*"

# optional highlightling
color   body    green           black    "LGTM"
color   body    brightmagenta   black    "-- Commit Summary --"
color   body    brightmagenta   black    "-- File Changes --"
color   body    brightmagenta   black    "-- Patch Links --"
color   body    green           black    "^Merged #.*"
color   body    red             black    "^Closed #.*"
color   body    brightblue      black    "^Reply to this email.*"
```

这是本人设置的Patch高亮模式, 采用正则表达式在邮件正文(body)中进行匹配高亮对应行. 这里就不对正则表达式部分进行详细解释了.

## 嵌套引用高亮区分

Mutt很贴心的设置了引用高亮功能, 配置如下:

```
# 引文起始符号设置
# set quote_regexp = "^([ ]t]*[|>:}#])+"
set quote_regexp = "^([ \t]*[>])+"

# 嵌套引文不同层的颜色
color quoted        blue            black
color quoted1       magenta         black
color quoted2       cyan            black
color quoted3       yellow          black
color quoted4       red             black
```

这里, quoted4代表最里层的引用内容, quoted代表最外层的引用内容. 通过给不同层的引文设置不同颜色, 就可能很迅速的区分引文和本次邮件作者的回复内容了, 非常高效!

另外, 在设置引文高亮的时候, 需要设置引用符号, 常见的引用符号有:`>`, `#`, `+`乃至tab缩进等, 通过`quote_regexp`变量告知Mutt要将哪些行识别为引文, 这样才能使得之后的quoted设置生效. 由于本人订阅的邮件列表采用的符号`>`作为引文开头, 所以就之设置了一个引文符号, 避免不必要的混乱高亮.

# 配置procmail

正如前言中说的, 本人配置Mutt主要是为了阅读邮件列表, 经过上述一系列的配置, 基本已经能正常并相对高效的阅读邮件列表了. 但是邮件列表一天能有数十封乃至几百封邮件, 根据上述的配置, getmail收取的所有的邮件都直接投放到inbox目录下, 这导致了我一打开Mutt, 就是满屏的来自邮件列表的邮件, 而私人邮件和工作邮件则完全被淹没了, 虽然可以通过设置不同的高亮来区分, 但是将邮件列表的邮件直接放在inbox中实在不是一个明智的操作, 一个合理的办法是专门设置一个信箱, 例如mlist来存放来自邮件列表的邮件.

当通过getmail收取到邮件的时候, 如果是来自邮件列表的邮件, 直接跳过inbox信箱, 直接投放到mlist的信箱中, 这样的话既能保持inbox信箱的清洁, 又能在想查看邮件列表的时候切换到mlist邮箱中查看. 想要完成这个功能, 我们需要在getmail和信箱目录之前添加一层邮件分拣的操作, 而这个操作, 可以通过procmail工具来完成.

procmail是一个邮件分拣工具, 可以根据发信人, 主题, 信件长短, 关键词等信息对邮件进行过滤[?].

我们要利用procmail做的就是, 从收取到的邮件中过滤出来自邮件列表的邮件, 然后将其投递到`mlist`目录中. 那么显然的, 我们需要先在Mutt的工作目录下建立相应的`mlist`目录, 为了符合maildir格式, 我们还需要在mlist目录下建立`cur`, `new`, `tmp`这3个子目录.

```
$ mkdir ~/.mail/mlist
$ cd ~/.mail/mlist
$ mkdir cur new tmp
```

## 修改getmail配置

我们提到过, procmail需要将getmail收取到的邮件进行过滤, 那么很明显的, procmail需要在getmail将邮件投递到Mutt的inbox目录前对邮件进行过滤. 因此, 我们需要改变getmail的投递行为, 让getmail收取邮件后, 不放到Mutt的inbox目录中, 而是将邮件交给procmail来处理, 这里我们需要对上文所述的getmailrc配置文件进行适当修改.

```
$ vim ~/.getmail/getmailrc
```

将`[destination]`节中的`type`和`path`字段稍作修改

```
[destination]
# 以目录形式储存
# type = Maildir
# path = ~/.mail/inbox/
```

```
# 由于采用了procmail, 收取的邮件交由procmail分拣而非直接投递到文件夹中
type = MDA_external
path = /usr/bin/procmail
```

## 编辑procmailrc文件

这样, getmail就会将收取的邮件交给procmail来做过滤了. 接下来, 我们来配置procmail. procmail的配置集中在`~/.procmailrc`文件中. 新建并对其进行编辑如下:

```
# procmail用于将收到的邮件进行分拣,并发放到不同的邮箱(文件夹)中
# 设置PATH变量以便procmail正常运行,根据文档,PATH应在第一行就设置
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin
# 指定shell
SHELL = /bin/bash
# 指定发信工具位置
SENDMAIL=/usr/bin/msmtp
# Mutt邮件存放主目录
MAILDIR=$HOME/.mail
# 默认存放的目录
DEFAULT=$MAILDIR/inbox
# 日志文件存放位置
LOGFILE=$MAILDIR/procmail.log
# 诊断用,关闭
VERBOSE=no

# 分拣规则, procmail采用名为recipe的规则来处理,具体查看man procmailrc
# recipe的一般格式:
#        :0 [flags] [ : [locallockfile] ]
#        <zero or more conditions (one per line)>
#        <exactly one action line>
#
# 其中, conditions行以'*'作为开头, '*'之后跟的正则表达式完全兼容egrep形式.
# 关于规则示例可以查看man procmailex
# action行中,可以以'!','|','{'开头,其中'|'开头之后可以跟一条bash命令,
# 此外的任意符号被认为是邮箱文件夹, 可以采用绝对路径, 若使用相对路径,
# 则是相对与$MAILDIR, 例如inbox/,表示$MAILDIR/inbox

:0
* ^To:.*my_mail@mail.com
inbox/

# 归档Xen的邮件列表
:0
```

```
* ^Return-Path:.*xen-devel-bounces@lists.xen.org
xen/

# 剩下的邮件存放至inbox目录中
:0
* .*
inbox/
```

在上述配置文件中, 各个字段的作用注释已经说的很清楚了, 有趣的是, 我们之前一直说procmail如何过滤投递邮件, 但事实上procmail也有转发邮件的功能, 因此大家会看到字段设置中有设置发信程序的部分.

## recipe示例讲解

procmail配置文件中唯一需要讲解也是重点需要讲解的就是规则部分了.
promail的规则称之为recipe. 可以通过man procmailrc来查看具体信息. recipe的格式如下:

```
:0 [flags] [ : [locallockfile] ]
<zero or more conditions (one per line)>
<exactly one action line>
```

第一行以`:0`开头, 后面可以跟一些标志, 以及使用锁文件避免潜在冲突.
第二行是conditions, 即匹配条件, 以`*`作为开头, 后跟正则表达式用于匹配, 此处的正则表达式兼容egrep的extended regular expression格式.
第三行是action, 即执行部分, 该行若以`|`开头, 表示之后跟一条bash命令; 如果以`!`开头, 后面则跟的是对转发(forward)地址; 如果以`{`开头, 则是嵌套recipe. **此外其它的字符开头都将视为邮箱目录**.

在讲解具体的规则之前, 我们先来理解下procmail的recipe处理流.

```
             -> recipe1                    recipe1                    recipe1
1000 mails/    recipe2 --- 800 mails --> recipe2 --- 300 mails\    recipe2
               recipe3                    recipe3                -> recipe3
```

如上图所示, 假设在procmailrc文件中包含3条recipie, 从上到下分别是recipe1, recipe2, recipe3. 现在有1000封邮件, procmail将会根据规则依次过滤它们. procmail采取的默认操作是移动邮件(move), 所以我们看到, 1000封邮件在经过recipe1的规则后, 假设匹配掉了200封, 那么在进行recipe2匹配的时候, 是对剩余的800封邮件进行处理了. 从这个过程可以看出, 排在越前面的recipe优先级越高, 而且已经被recipe处理过得邮件若没有特殊指明, 是不会再被接下来的recipe处理的.

以下通过几条recipe来简要讲解下:

```
:0 c
* ^To:.*my_mail@mail.com
inbox/
```

上述recipe表示, 对于发给my_mail@mail.com的邮件, 将其**复制**到`~/.mail/inbox`信箱中.
注意到这里在第一行的`:0`之后加了一个`c`作为标志, 代表复制邮件, 而非移动邮件. 复制的意义在于, 该邮件通过recipe进行过滤操作, 同时留下一个副本参与之后的recipe过滤操作. 这么做实际上产生了2封重复的邮件. 关于`c`标志以及其它`[flags]`标志, 请参见man procmailrc.
然后是第三行的`inbox/`, 由于不是以`|`, `!`或是`{`开头, 因此被解释为信箱目录, 这里采用的是相对路径的形式, 当然也完全可以写成`~/.mail/inbox/`, 这里的相对路径是相对于`~/.procmailrc`配置文件中的`$MAILDIR`变量而言的, 之前我们将其设置为了`$HOME/.mail`, 因此这里的`inbox/`代表的是`~/.mail/inbox`目录.

再来看一下来自邮件列表的邮件归档设置

```
# 归档Xen的邮件列表
:0
* ^Return-Path:.*xen-devel-bounces@lists.xen.org
xen/
```

上述recipe将来自Xen的开发者列表的邮件都投递到`~/.mail/xen`目录下. 没什么特殊的技巧, 主要起作用的是第二行的正则表达式, 这里匹配邮件头部中的`Return-Path`字段, 若该字段包含`xen-devel-bounces@lists.xen.org`, 则该邮件被认为是来自xen的邮件列表, 从而被投递到xen的信箱中.
值得一提的是, Xen的开发者邮件列表有多个地址, 例如`xen-devel@lists.xenproject.org`, `xen-devel@lists.xen.org`, `xen-devel@lists.xensource.com`, 而且不同的发信人, 可能会把邮件列表的地址写在收信人(To)一栏, 也有部分人习惯把邮件列表地址写在抄送(Cc)一栏, 如果考虑从To或是Cc字段来匹配邮件列表那就比较复杂了, 但是有意思的是不管发给哪个列表地址的邮件, 邮件头部`Return-Path`字段的地址都是`xen-devel-bounces@lists.xen.org`, 因此只需要对`Return-Path`字段进行匹配就能过滤出全部来自该邮件列表的邮件. 其它的邮件列表也是如此.

最后, 我们看下如下的规则

```
# 剩下的邮件存放至inbox目录中
:0
* .*
inbox/
```

这条规则是最后一条recipe, 按照我们上面说的recipe处理流, 已经被上层recipe过滤掉的邮件并不会被这条recipie过滤, 因此, 我们这里匹配任意的邮件, 则实现的是将上层全部recipe规则之外没被匹配掉的邮件全都放到inbox信箱中.

至此, procmail配置讲解结束, 关于procmail一些其它的高级功能例如转发等请查看promcail manual, 更多的recipe示例可以参考man procmailex.

## Hooks

Hooks是Mutt中非常强大的功能, 不过鉴于常规使用上不怎么会用到, 亦深入研究, 先不介绍了, 如果之后用到了在将相关内容补上, 没有的话就...XD

Hooks的相关介绍, 请参阅: Mutt Manual: Using Hooks

# 总结

本文针对邮件列表阅读这一应用场景出发, 对为什么使用Mutt, Mutt是什么, 怎么配置Mutt进行了简要的叙述, 其中参考了众多资料, 在文中相应的地方都给出了引用链接. 本文篇幅较长且行文冗余啰嗦, 无奈受限于作者文学素养不高却又想尽可能表述的清楚明白, 同时尽量将涉及到的相关内容都覆盖讲解到以更好的理解Mutt. 如前言所述, 此文主要是用来记录本人配置Mutt过程中的心得以便日后回顾, 同时也希望能帮助到一些刚接触Mutt被整的一头雾水的同学, 经过几天的使用, 越发发现Mutt真的是一个非常高效且好用的工具, 虽然前期的配置和理解较为复杂, 但是我认为完全值得. 最后, 本人水平有限, 文中若有存在不足或错误之处, 还请各位同学指出.

---

email[1] getmail[1] msmtp[1] mutt[1] procmail[1] tools[1]



**正式完成从Wordpress迁移至Hexo**

# 目录