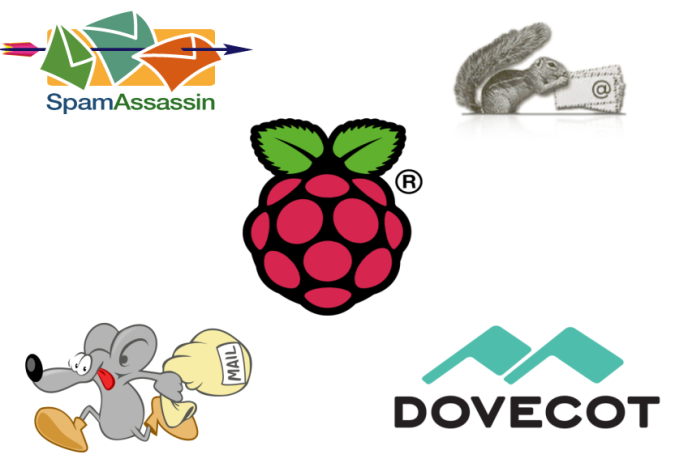
# Part O introduction



The RasPi’s small size and low power consumption make it an ideal choice for use as a home email server. After trying a couple of different pieces of software, I finally found an excellent combination: **Postfix** with **Dovecot** and **Squirrelmail**, plus **Spamasssassin** and **Sieve** for spam filtering.

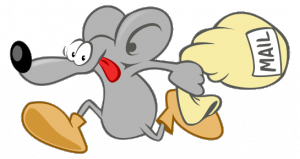
There are many, many tutorials out there for the first trilogy of programs, but since the configuration is slightly different for each distribution I kept coming unstuck when setting mine up on the Pi. Having finally got mine configured properly, I’ve put together a set of 5 tutorials, which will take you from a vanilla Raspbian image to a fully functioning email server in no time.

When writing the tutorial I made an effort to explain what each setting does instead of just dumping commands. With a bit of luck at the end of the process you’ll not only have a working server, you’ll understand how it works… without having to wade through reams of documentation like I did!

If you follow the tutorials from start to finish, here’s what you’ll end up with:

1. An email server that you can run 24/7/365 for under £5 of electricity per year
2. Personalised email address like you@yourdomain.com (requires you to have registered a domain name with a registrar like[namecheap.com](https://www.namecheap.com/) - see my [DNS basics tutorial](https://samhobbs.co.uk/2015/02/dns-basics-websites-and-email-servers))
3. The ability to connect from anywhere, and read & send email, using a secure IMAP connection on your phone, tablet or computer
4. Log in to webmail using any web browser on a secure HTTPS connection, read & send email
5. Complete control over your personal communication. Your emails are stored on YOUR server, and nobody is scanning them to sell you adverts.
6. Smart spam filtering with Spamassassin
7. Customisable mail sorting with Sieve rules

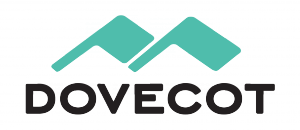
## Postfix, the Mail Transfer Agent

  
Postfix is the program that lets you send and receive email using Simple Mail Transfer Protocol (SMTP). Whilst you, the user, may connect to your email server using IMAP (on port 143 or 993), or POP (on port 110 or 995), email servers talk to each other using SMTP on port 25.

So, this is the basic core of the server. Without it, you wouldn’t be able to send or receive any emails!

I’ve covered the setup here:  
[Raspberry Pi Email Server Part 1: Postfix](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-1-postfix)

## Dovecot, the POP/IMAP Server

  
Dovecot is used for two things:

1. It provides you with IMAP functionality
2. It checks that you are who you say you are using Simple Authentication and Security Layer (SASL) before you send or fetch mail

If you’re not interested in connecting with IMAP on your devices, you still need Dovecot. Not only is it doing SASL for you, but Squirrelmail connects using IMAP in order to provide you with webmail.

I’ve covered Dovecot installation and configuration here:  
[Raspberry Pi Email Server Part 2: Dovecot](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-2-dovecot)

## Squirrelmail, for Webmail

  
Squirrelmail is handy because it allows you to check your email in any browser, from anywhere.

Of the first three, it’s probably the easiest to configure. I’ve covered it here:  
[Raspberry Pi Email Server Part 3: Squirrelmail](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-3-squirrelmail)

## Spamassassin, for Marking Spam

  
Spamassassin is the program that we will use to audit incoming mail and decide whether or not it’s spam. Spamassassin doesn’t actually sort the mail into the spam folder, it only changes information in the headers based on the results of the scan. I’ve covered it here: [Raspberry Pi Email Server Part 4: Spam Detection with Spamassassin](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-4-spam-detection-with-spamassassin).

## LMTP & Sieve for Spam Sorting & Mailbox Organisation

After Spamassassin has checked incoming mail to see if it’s spam or not, we need another program to sort it into the right mail folder. This final step will be done with Dovecot’s Local Mail Transfer Protocol (LMTP) daemon and a Sieve plugin.

Sieve is a simple programming language that allows users to define what to do with incoming email based on a predefined set of rules – think “if the header contains this flag, put it in the spam folder” kind of thing and you’ll get the gist. Aside from spam filtering, Sieve can be used to automatically sort & de-clutter your inbox. These steps are covered in the final tutorial: [Raspberry Pi Email Server Part 5: Spam Sorting with LMTP & Sieve](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-5-spam-sorting-with-lmtp-sieve)

Enjoy! I’d love to hear how you get on, so leave a comment below :)

**Category:**

[Email Server](https://samhobbs.co.uk/taxonomy/term/3)

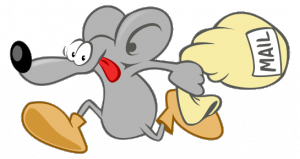
[Raspberry Pi](https://samhobbs.co.uk/taxonomy/term/1)

**Type:**

[Tutorial](https://samhobbs.co.uk/taxonomy/term/9)

# Part I postfix

Sam Hobbs · 22nd December, 2013 · 12:00am



This is the **first part** of a five part tutorial that will show you how to install a full featured email server on your Raspberry Pi. This tutorial covers Postfix, the Mail Transfer Agent.

The parts are:  
[The Introduction & Contents Page (read first)](https://samhobbs.co.uk/raspberry-pi-email-server)  
[Raspberry Pi Email Server Part 1: Postfix](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-1-postfix)  
[Raspberry Pi Email Server Part 2: Dovecot](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-2-dovecot)  
[Raspberry Pi Email Server Part 3: Squirrelmail](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-3-squirrelmail)  
[Raspberry Pi Email Server Part 4: Spam Detection with Spamassassin](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-4-spam-detection-with-spamassassin)  
[Raspberry Pi Email Server Part 5: Spam Sorting with LMTP & Sieve](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-5-spam-sorting-with-lmtp-sieve)

## Installing Postfix

Note: While you are setting up the mail server on the Pi, it’s a good idea to turn off port forwarding rules for email to the Pi in your router’s firewall. If you don’t have any port forwarding rules now, that’s great, don’t worry – I’ll prompt you to set them up later.

First, log into your Pi with a SSH session and install postfix:

sudo apt-get update

sudo apt-get install postfix

You will see a menu with some choices. Select **“Internet Site”** and then set the mail name to your domain name, not including www. (e.g. samhobbs.co.uk).

The setup script will then do some automatic configuration for you. The output will look something like this:

Selecting previously unselected package postfix.

(Reading database ... 67653 files and directories currently installed.)

Unpacking postfix (from .../postfix\_2.9.6-2\_armhf.deb) ...

Processing triggers for man-db ...

Setting up postfix (2.9.6-2) ...

Adding group `postfix' (GID XXX) ...

Done.

Adding system user `postfix' (UID XXX) ...

Adding new user `postfix' (UID XXX) with group `postfix' ...

Not creating home directory `/var/spool/postfix'.

Creating /etc/postfix/dynamicmaps.cf

Adding tcp map entry to /etc/postfix/dynamicmaps.cf

Adding sqlite map entry to /etc/postfix/dynamicmaps.cf

Adding group `postdrop' (GID XXX) ...

Done.

setting myhostname: samhobbs

setting alias maps

setting alias database

changing /etc/mailname to samhobbs.co.uk

setting myorigin

setting destinations: samhobbs.co.uk, samhobbs, localhost.localdomain, localhost

setting relayhost:

setting mynetworks: 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128

setting mailbox\_size\_limit: 0

setting recipient\_delimiter: +

setting inet\_interfaces: all

/etc/aliases does not exist, creating it.

WARNING: /etc/aliases exists, but does not have a root alias.

You can edit all of this later.

You may also get some warnings like this:

postmulti: warning: inet\_protocols: disabling IPv6 name/address support: Address family not supported by protocol

IPv6 is a new type of IP address that was introduced because we’re running out of the “old” IPv4 addresses. Not many ISPs support IPv6 yet, so you probably don’t need it. Unless you fix the warning, you’ll see it every time.

Change directory into the postfix configuration folder:

cd /etc/postfix/

Edit /etc/postfix/main.cf with your favourite command line text editor (e.g. sudo nano main.cf) and add inet\_protocols = ipv4to the end of the file.

Now is also a good time to check that your hostname is specified properly in /etc/postfix/main.cf. The setup script takes the hostname of the server and uses that, but it may not be in the right format, i.e. “samhobbs” instead of “samhobbs.co.uk”. Find the line that beginsmyhostname = and make sure it is your fully qualified domain name. This is important because your server will use this to talk to other mail servers, and some will reject your emails if you don’t use a fully qualified domain name to say hi! This is covered in more detail in the helo access restrictions later.

Restart postfix and you shouldn’t see the warnings any more:

sudo service postfix restart

## Testing and Configuration

Before you start, it’s probably worth backing up the configuration files in their current state. This way, you’ll have something to compare to if you’re ever trying to work out which bits were defaults and which bits you changed yourself:

cd /etc/postfix

sudo cp main.cf main.cf.BAK

sudo cp master.cf master.cf.BAK

## Mailbox Setup

There are a couple of different types of mailbox you can use, I’ve chosen to use a “Maildir” rather than “mbox” configuration. For users with “real” UNIX accounts on the system (like the one you’re using to log in), Maildir creates a folder in the user’s home directory and places emails inside it, one file for each email.

I prefer this to the alternatives, because it’s easier to see and understand: you can rummage around in your home folder and see all your emails as individual files.

To tell Postfix to use the Maildir format, add the following lines to /etc/postfix/main.cf:

home\_mailbox = Maildir/

mailbox\_command =

If there's already a line with mailbox\_command, comment it out by adding a # at the start of the line.

We also need to create the mail directory and its subfolders for existing users, and add some things to /etc/skel (the template for new users) so that if you create a new account this will be done automatically.

These commands are part of Dovecot, so first we need to install it:

sudo apt-get update

sudo apt-get install dovecot-common dovecot-imapd

You will get a lot of output: some other dovecot packages will automatically be installed and the config files will be created. You will also see some errors – don’t worry about those for now, I’ll explain how to deal with them in part 2, later.

Now we can create those mail folders. Run the following commands to create the template files:

sudo maildirmake.dovecot /etc/skel/Maildir

sudo maildirmake.dovecot /etc/skel/Maildir/.Drafts

sudo maildirmake.dovecot /etc/skel/Maildir/.Sent

sudo maildirmake.dovecot /etc/skel/Maildir/.Spam

sudo maildirmake.dovecot /etc/skel/Maildir/.Trash

sudo maildirmake.dovecot /etc/skel/Maildir/.Templates

Next, copy the files over to existing users’ home directories, and change the ownership and permissions for privacy (replace USER with the username you are doing this for, and repeat for all existing usernames):

sudo cp -r /etc/skel/Maildir /home/USER/

sudo chown -R USER:USER /home/USER/Maildir

sudo chmod -R 700 /home/USER/Maildir

## Initial Testing

Now, the best way to test Postfix during configuration is to use Telnet, because it is such a simple way of communicating between programs and there’s less to go wrong and get confused about.

First, install telnet:

sudo apt-get install telnet

Now, still inside the SSH session to your pi, type this command. It will connect you to port 25 on the Pi:

telnet localhost 25

You can now test sending an email using SMTP. Here are the steps:

1. send an ehlo command to tell the server who you are, and it will tell you its capabilities
2. use the mail from command to say who the email is from. If you are sending it from an address that exists on the server, you needn’t include the domain name (i.e. user instead of user@yourdomain.com)
3. use the rcpt to command to tell the server where to send the email
4. Use the data command to tell the server that you’re about to start giving it the message you want to send
5. Type Subject: YOUR SUBJECT then enter to set a subject
6. Type the body of your email. Once you’re done, press ENTER, then ., then ENTER again.
7. Type quit to exit

Here’s an example:

telnet localhost 25

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

220 samhobbs.co.uk ESMTP Postfix (Debian/GNU)

ehlo foobar

250-samhobbs.co.uk

250-PIPELINING

250-SIZE 10240000

250-VRFY

250-ETRN

250-STARTTLS

250-ENHANCEDSTATUSCODES

250-8BITMIME

250 DSN

mail from: me

250 2.1.0 Ok

rcpt to: me@outsideemail.com

250 2.1.5 Ok

data

354 End data with <CR><LF>.<CR><LF>

Subject: test

This is a test email

.

250 2.0.0 Ok: queued as A639C3EE6D

quit

221 2.0.0 Bye

## Some Access Restrictions

Add the following to /etc/postfix/main.cf to restrict who can send emails to external mail servers:

smtpd\_recipient\_restrictions =

permit\_sasl\_authenticated,

permit\_mynetworks,

reject\_unauth\_destination

Reload postfix:

sudo service postfix reload

* **Line 1** begins the list of restrictions.
* **Line 2** permits users who have authenticated with Simple Authentication and Security Layer (SASL) to send email to any destination (this is part of the Dovecot config in Part 2, later).
* **Line 3** will let users send emails to any destination if they have connected from an IP address defined in mynetworks.
* **Line 4** will reject the email if none of the above conditions have been met unless the “rcpt to” address is one of the addresses that your server is accepting email to (as defined in main.cf with the mydestination parameter).

In its present state, the email server will allow you to send external emails because the connection is originating from the Pi itself (you are logged in via SSH) and not an unknown computer. Addresses of “trusted” computers are listed under the mynetworks setting in main.cf, e.g.

mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128

Try sending an external email again, using telnet as before. You should be able to do so without any issues.

Now we want to see what kind of response someone would get if they were connecting from outside of the IP range defined in mynetworks, to make sure Pi won’t allow everyone to send outgoing emails from your server. To simulate this we can comment out permit\_mynetworksunder smtpd\_recipient\_restrictions:

smtpd\_recipient\_restrictions =

permit\_sasl\_authenticated,

# permit\_mynetworks,

reject\_unauth\_destination

Now reload the postfix configuration:

sudo service postfix reload

This will let you see what kind of response you would get if you weren’t sending the email from mynetworks. Try sending again, and you should receive an error “554: Relay access denied“:

admin@samhobbs /etc/postfix $ telnet localhost 25

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

220 samhobbs.co.uk ESMTP Postfix (Debian/GNU)

ehlo samhobbs.co.uk

250-samhobbs

250-PIPELINING

250-SIZE 10240000

250-VRFY

250-ETRN

250-STARTTLS

250-ENHANCEDSTATUSCODES

250-8BITMIME

250 DSN

mail from: USER

250 2.1.0 Ok

rcpt to: me@externalemail.com

554 5.7.1 <me@externalemail.com>: Relay access denied

quit

221 2.0.0 Bye

Connection closed by foreign host.

Perfect. Leave permit\_mynetworks commented out in your smtpd\_recipient\_restrictions (you'll see why in part 2).

## Helo access restrictions

Helo access restrictions can be a very useful way of blocking spam.

Note that we’re not talking about unauthorised people being able to send email outside your network any more (that’s taken care of with thesmtpd\_recipient\_restrictions); we’re now talking about stopping spammers from sending incoming mail to your email address.

Spammers try to conceal their identity so that they don’t end up on block lists, so they rarely use helo hostnames that could identify them – these hostnames are written to the mail log files. As a result, they often make up a random string or use an IP address instead of a domain name.

Luckily, these are easily taken care of.

Add the following to /etc/postfix/main.cf:

smtpd\_helo\_required = yes

smtpd\_helo\_restrictions =

permit\_mynetworks,

permit\_sasl\_authenticated,

reject\_invalid\_helo\_hostname,

reject\_non\_fqdn\_helo\_hostname,

reject\_unknown\_helo\_hostname

* **Line 1** requires people and programs to identify themselves when they send email, using the helo or ehlo commands I mentioned earlier.
* **Line 2** starts the list of restrictions.
* **Line 3** accepts any old rubbish in the ehlo if it comes from an IP address defined in mynetworks. If the connection isn’t connecting from an IP address in mynetworks, then the helo hostname is checked against the rest of the list.
* **Line 4** accepts any helo hostname if the client is authenticated with SASL (I added this to the tutorial recently after troubleshooting problems some people had in the comments – it allows you to connect from any network and still send messages through your Pi. Mobiles will usually work without this because most providers pass mail through their own proxies, so your Pi receives a connection from the proxy – which has a valid hostname – and not from the mobile, which may be called something like “android-b627cfe2efea7e67″).
* **Line 5** rejects connection attempts when the HELO hostname syntax is invalid.
* **Line 6** rejects non-fully qualified domain names (for example, foobar instead of foobar.com). This will also block those random strings, e.g. “kjhrsbvks”.
* **Line 7** rejects the helo hostname if it that domain doesn’t have a valid [DNS A](https://samhobbs.co.uk/2015/02/dns-basics-websites-and-email-servers#A) or [MX record](https://samhobbs.co.uk/2015/02/dns-basics-websites-and-email-servers#MX). For example, someone spamming you could make up a domain like theflyingspaghettimonster.com. If that domain doesn’t actually exist and have the right records, then your server won’t accept it as a hostname, and the email will be rejected.

If the helo hostname gets past line 7 and hasn’t been denied, it is accepted. You’d be surprised how much spam these helo access restrictions will block on their own (looking through my log files, I can see numerous spam scripts that have attempted to ehlo with my IP address), but there’s an extra step we can add in here to help:

## Blocking people claiming to be your domain name

Many spammers try to send email to you after helo’ing with your own domain name. Since postfix doesn’t check whether or not they’re lying about their helo hostname, this will usually work.

But, since we’ve put permit\_mynetworks at the top of the list, anyone **actually** sending an email from your domain will be accepted already. Anyone using your hostname who isn’t in mynetworks is an imposter.

So, add one more line to the end of the restrictions list:

smtpd\_helo\_restrictions =

permit\_mynetworks,

permit\_sasl\_authenticated,

reject\_invalid\_helo\_hostname,

reject\_non\_fqdn\_helo\_hostname,

reject\_unknown\_helo\_hostname,

check\_helo\_access hash:/etc/postfix/helo\_access

That last line checks a file for custom rules you’ve built in. Create the file:

sudo nano /etc/postfix/helo\_access

Add the following lines, edited for your domain:

samhobbs.co.uk REJECT Get lost - you're lying about who you are

mail.samhobbs.co.uk REJECT Get lost - you're lying about who you are

Now tell postfix to map the file, and restart postfix:

sudo postmap /etc/postfix/helo\_access

sudo service postfix restart

Now anyone who tries to ehlo with one of the hostnames you defined in that file gets rejected, and sees the “get lost” message. Your legitimate servers won’t have that problem, because they will already have been accepted higher up the list.

Neat, right? I found that little nugget of wisdom [at unixwiz.net](http://unixwiz.net/techtips/postfix-HELO.html).

## Moving on…

We’re almost done with Postfix now, athough there are a few bits of configuration that we’ll want to do once we’ve set up SASL with Dovecot, which I’ve chosen to lump in with the Dovecot tutorial.

In [Raspberry Pi Email Server Part 2: Dovecot](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-2-dovecot), we’ll set up Dovecot to provide SASL authentication and IMAP capability.

Please leave a comment if you’re having trouble with anything in this tutorial, and I’ll try and help you out!

**Tags:**

[Raspberry Pi](https://samhobbs.co.uk/tags/raspberry-pi)

[email](https://samhobbs.co.uk/tags/email)

[server](https://samhobbs.co.uk/tags/server)

[postfix](https://samhobbs.co.uk/tags/postfix)

**Category:**

[Email Server](https://samhobbs.co.uk/taxonomy/term/3)

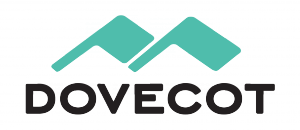
[Raspberry Pi](https://samhobbs.co.uk/taxonomy/term/1)

**Type:**

[Tutorial](https://samhobbs.co.uk/taxonomy/term/9)

# Part II dovecot

Sam Hobbs · 22nd December, 2013 · 12:00am



This is the **second part** of a five part tutorial that will show you how to install a full featured email server on your Raspberry Pi. This tutorial covers Dovecot, which provides SASL authentication and IMAP capabilities. The parts are:

[The Introduction & Contents Page (read first)](https://samhobbs.co.uk/raspberry-pi-email-server)  
[Raspberry Pi Email Server Part 1: Postfix](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-1-postfix)  
[Raspberry Pi Email Server Part 2: Dovecot](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-2-dovecot)  
[Raspberry Pi Email Server Part 3: Squirrelmail](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-3-squirrelmail)  
[Raspberry Pi Email Server Part 4: Spam Detection with Spamassassin](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-4-spam-detection-with-spamassassin)  
[Raspberry Pi Email Server Part 5: Spam Sorting with LMTP & Sieve](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-5-spam-sorting-with-lmtp-sieve)

## Fixing the errors that appeared during dovecot installation

In part 1, when you installed Dovecot I mentioned that you might see some errors like this:

Creating config file /etc/dovecot/conf.d/20-imap.conf with new version

[....] Restarting IMAP/POP3 mail server: dovecotError: socket() failed: Address family not supported by protocol

Error: service(imap-login): listen(::, 143) failed: Address family not supported by protocol

Error: socket() failed: Address family not supported by protocol

Error: service(imap-login): listen(::, 993) failed: Address family not supported by protocol

Fatal: Failed to start listeners

failed!

invoke-rc.d: initscript dovecot, action "restart" failed.

dpkg: error processing dovecot-imapd (--configure):

subprocess installed post-installation script returned error exit status 1

Setting up dovecot-ldap (1:2.1.7-7) ...

These errors are caused by the lack of IPv6 support, which I mentioned in the previous tutorial. To remove the errors, open the main dovecot configuration file (/etc/dovecot/dovecot.conf) and find this line:

listen = \*, ::

And change it to:

listen = \*

The \* means “all IPv4 addresses”, the :: means “all IPv6 addresses”. Now restart Dovecot, and you shouldn’t get any errors:

sudo service dovecot restart

Note: since I wrote this tutorial, there have been a few small changes to the default configuration file - you may find that the line is commented (with a # at the start of the line). If so, remember to uncomment it when you make your changes!

## Tell Dovecot where your Mailbox is

Open /etc/dovecot/conf.d/10-mail.conf and find this line:

mail\_location = mbox:~/mail:INBOX=/var/mail/%u

Change it to this:

mail\_location = maildir:~/Maildir

## Instruct Postfix to use Dovecot SASL

Now we need to tell Postfix that we would like to use Dovecot for SASL authentication. Open /etc/postfix/main.cf and add these lines:

smtpd\_sasl\_type = dovecot

smtpd\_sasl\_path = private/auth

smtpd\_sasl\_auth\_enable = yes

Now tell Dovecot to listen for SASL authentication requests from Postfix. Open /etc/dovecot/conf.d/10-master.conf and comment out the current block that begins with service auth (place a # at the start of each line). Replace it with this:

service auth {

unix\_listener /var/spool/postfix/private/auth {

mode = 0660

user = postfix

group = postfix

}

}

Now you want to enable plain text logins. Do it by adding these two lines to /etc/dovecot/conf.d/10-auth.conf. Make sure they are not already present in the file, or your settings may be overwritten with the default ones if the default is declared later in the file than the lines you add. If the parameters are already present, you can either modify the existing lines or comment them out and add these new ones:

disable\_plaintext\_auth = no

auth\_mechanisms = plain login

Note that although the logins are in plain text, we will be setting Postfix up later so that it only allows you to use plaintext logins from within SSL/TLS. This means that your login and password will sent in an encrypted session - you wouldn't see them in plain text if you used a packet sniffer, for example. For now, we’re allowing unencrypted plain text logins so that we can test logging in with Telnet. Since the connection is local (from the Pi to the Pi), your password isn’t being sent over any insecure networks so this is fine.

## Testing SASL

Creating a new user for testing purposes is a good idea. Let’s call this temporary user testmail and give it the password test1234 Use this command to add the user, and follow the prompts including setting a password.

sudo adduser testmail

Now restart Postfix and Dovecot:

sudo service postfix restart

sudo service dovecot restart

We’re now going to try and send an email after authenticating with SASL. The server is expecting to see a base64 encoded version of your username and password, so we have to convert it first. There are three ways of doing this, so I've given examples below using the testmailusername and test1234 password:

#Method No.1

echo -ne '\000testmail\000test1234' | openssl base64

#Method No.2

perl -MMIME::Base64 -e 'print encode\_base64("\0testmail\0test1234");'

#Method No.3

printf '\0%s\0%s' 'testmail' 'test1234' | openssl base64

I have discovered that if your password starts with a number, methods 1 and 2 don’t work. Assuming the username and password aretestmail and test1234, the commands produce this:

AHRlc3RtYWlsAHRlc3QxMjM0

**WARNING:** If you’re having problems with authentication and you paste examples to forums or mailing lists, be aware that it is really easy to convert this back into your username and password (hence the creation of a test user). If you're using your real username and password to test, redact it before posting!

Now, still logged into the Pi via SSH, you can telnet port 25 to test whether or not SASL is working. There’s only one extra step, which is theAUTH PLAIN command that comes after ehlo but before mail from. For testing, the permit\_mynetworks parameter should be commented out under your postfix smtpd\_recipient\_restrictions block in /etc/postfix/main.cf. If you’re following on from [Raspberry Pi Email Server Part 1: Postfix](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-1-postfix) then this should already be the case. If you have to change it, remember to reload postfix (sudo service postfix reload) after you change the value. Here’s an example:

telnet localhost 25

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

220 samhobbs ESMTP Postfix (Debian/GNU)

ehlo facebook.com

250-samhobbs

250-PIPELINING

250-SIZE 10240000

250-VRFY

250-ETRN

250-STARTTLS

250-AUTH PLAIN LOGIN

250-ENHANCEDSTATUSCODES

250-8BITMIME

250 DSN

AUTH PLAIN AHRlc3RtYWlsAHRlc3QxMjM0

235 2.7.0 Authentication successful

mail from:testmail

250 2.1.0 Ok

rcpt to:me@externalemail.com

250 2.1.5 Ok

data

354 End data with .

Subject: This is my first email that has been authenticated with Dovecot SASL

Woop woop

.

250 2.0.0 Ok: queued as B87133F768

quit

221 2.0.0 Bye

Connection closed by foreign host.

Now try again but enter the username/password incorrectly (base64 encode something random) – you should get an error message and the email won’t send. If everything went to plan, then SASL is working properly!

**You can now uncomment permit\_mynetworks again**.

## Separating Incoming email (unauthenticated) from Outgoing Email (SASL authenticated)

It’s probably a good idea to have a dedicated port for sending outgoing email…here’s why: Port 25 doesn’t require (but does offer) SSL/TLS encryption. If you mess up configuring your mail client you could end up letting it authenticate with SASL over insecure connections. Using a different port that **only** accepts SSL/TLS connections removes the risk that a poorly configured email client could be sending your password unencrypted over dodgy networks. There are two ports you can use for this:

1. 465: SMTP over SSL
2. 587: Email submission

587 is the “official” port for email clients (like K9 mail, Thunderbird and Outlook) to use when submitting messages to the Mail Submission Agent (your email server) – the submission may be encrypted or unencrypted depending on the server configuration. 465 was a port that was assigned for SMTP with SSL/TLS before the STARTTLS protocol was introduced, back in the days when you chose your port and that decided on the type of connection you were going to get (encrypted or unencrypted).

STARTTLS changed things because it allows you to connect with an unencrypted connection (like the one you get with Telnet), and then upgrade to an encrypted connection without changing port… so when STARTTLS was introduced, SMTPS on port 465 was removed from the standard because you could do the same thing with a single port (25).

However, I think there is some value in specifying a port for submission that only accepts SSL/TLS encrypted connections, and won’t work if the connection isn’t encrypted. This means that if you misconfigure your email client it just won’t work, instead of working and sending your password in an unencrypted format. So, anyway… Here’s how to set up Postfix to listen on port 465 for encrypted connections. The first step is telling Postfix to listen on port 465, so open /etc/postfix/master.cf and uncomment the line:

smtps inet n - - - - smtpd

Now restart Postfix:

sudo service postfix restart

Test whether Postfix is listening on port 465:

telnet localhost 465

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

220 samhobbs.co.uk ESMTP Postfix (Debian/GNU)

ehlo samhobbs.co.uk

250-samhobbs

250-PIPELINING

250-SIZE 10240000

250-VRFY

250-ETRN

250-STARTTLS

250-AUTH PLAIN LOGIN

250-ENHANCEDSTATUSCODES

250-8BITMIME

250 DSN

quit

221 2.0.0 Bye

Connection closed by foreign host.

OK, so now it’s listening on the right port, but it’s allowing unencrypted connections. Here’s how you force TLS on port 465: open/etc/postfix/master.cf and find the line you uncommented earlier. Below it are some options, you want to edit them so that they look like this (i.e. uncomment lines 2 and 3):

smtps inet n - - - - smtpd

-o syslog\_name=postfix/smtps

-o smtpd\_tls\_wrappermode=yes

Line 3 is forcing TLS on port 465, and line 2 means that connections to port 465 have a different label in the logs, which can be useful for debugging.

sudo service postfix restart

Now try connecting with Telnet again… you should be able to establish a connection, but not receive any prompts from the server:

telnet localhost 465

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

exit

exit

Connection closed by foreign host.

Now try openssl:

openssl s\_client -connect localhost:465 -quiet

depth=0 CN = samhobbs

verify error:num=18:self signed certificate

verify return:1

depth=0 CN = samhobbs

verify return:1

220 samhobbs.co.uk ESMTP Postfix (Debian/GNU)

quit

221 2.0.0 Bye

Good. There are a couple more changes we want to make here: first, tell Postfix to only advertise SASL authentication over encrypted connections (so that you don’t accidentally send your password in the clear). Open /etc/postfix/main.cf and add this line:

smtpd\_tls\_auth\_only = yes

sudo service postfix reload

Now connect to port 25 and you shouldn’t see AUTH advertised:

telnet localhost 25

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

220 samhobbs.co.uk ESMTP Postfix (Debian/GNU)

ehlo samhobbs.co.uk

250-samhobbs.co.uk

250-PIPELINING

250-SIZE 10240000

250-VRFY

250-ETRN

250-STARTTLS

250-ENHANCEDSTATUSCODES

250-8BITMIME

250 DSN

Lastly, we want to override the smtp\_recipient\_restrictions for port 465 so that it doesn't accept incoming messages from unauthenticated users.

At first, I didn't make this change and I noticed that some spam emails were coming in on port 465 and bypassing my spam filter, which I configured to scan all incoming email on port 25, but not 465 because I only expected it to be used for outgoing email. We can do this by overriding the smtp\_recipient\_restrictions list for port 465 in /etc/postfix/master.cf. Open master.cf and find the smtps line. Add a new recipient restrictions list option like this:

smtps inet n - - - - smtpd

-o syslog\_name=postfix/smtps

-o smtpd\_tls\_wrappermode=yes

-o smtpd\_recipient\_restrictions=permit\_sasl\_authenticated,reject

Now reload postfix:

sudo service postfix reload

Perfect! Postfix configuration is now complete.

## Testing IMAP

There are two main protocols for fetching mail: POP and IMAP. The main difference between them is what they do with emails when they collect them: a POP client will fetch email from your server and remove it from the server when it’s done. This is inconvenient if you want to connect with two or more devices (like a phone and a computer) and have complete copies of all your emails on both. IMAP, on the other hand, makes a copy of the emails on the server and leaves the originals there. For this reason, I think IMAP is much more useful than POP and I didn’t even bother to set up POP on my server. We can now test the IMAP server with Telnet in a similar way to SMTP & SASL testing earlier. This time, we’ll be using port 143, the standard port for IMAP. The stages are:

1. establish a connection with telnet localhost 143
2. log in with a login "USERNAME" "PASSWORD"" (not base64 encoded this time)
3. select inbox to see messages inside b select inbox
4. logout with c logout

In case you're wondering, the "a b c" thing is done because a client can send multiple commands to the server at once, and they might not come back in the same order depending on what they are. So, the responses have the same letter as the commands they are responding to so that the client doesn't get muddled.

Here’s an example, using the testmail user we created earlier:

telnet localhost 143

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

\* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE STARTTLS AUTH=PLAIN AUTH=LOGIN] Dovecot ready.

a login "testmail" "test1234"

a OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE SORT SORT=DISPLAY THREAD=REFERENCES THREAD=REFS MULTIAPPEND UNSELECT CHILDREN NAMESPACE UIDPLUS LIST-EXTENDED I18NLEVEL=1 CONDSTORE QRESYNC ESEARCH ESORT SEARCHRES WITHIN CONTEXT=SEARCH LIST-STATUS SPECIAL-USE] Logged in

b select inbox

\* FLAGS (\Answered \Flagged \Deleted \Seen \Draft)

\* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft \\*)] Flags permitted.

\* 1 EXISTS

\* 0 RECENT

\* OK [UNSEEN 1] First unseen.

\* OK [UIDVALIDITY 1385217480] UIDs valid

\* OK [UIDNEXT 2] Predicted next UID

\* OK [NOMODSEQ] No permanent modsequences

b OK [READ-WRITE] Select completed.

c logout

\* BYE Logging out

c OK Logout completed.

Connection closed by foreign host.

## Adding TLS support

Now that we know IMAP is working, we need to enable IMAPS (imap with SSL/TLS). The standard port for this is 993.

Many other tutorials that were written for older versions of dovecot will tell you to do this in different ways that won’t work, I tried 3 different methods before I ended up with a working one.

First, edit /etc/dovecot/conf.d/10-master.conf, find the “service imap-login” block and uncomment the port and SSL lines so that it looks like this:

service imap-login {

inet\_listener imap {

port = 143

}

inet\_listener imaps {

port = 993

ssl = yes

}

}

Edit 14/10/2015: the default dovecot configuration files changed recently after Jessie became the new stable distribution of Debian, which caused some users problems; TLS on port 993 used to be enabled by default but now it isn't. We need to re-enable it.

In /etc/dovecot/conf.d/10-ssl.conf, find ssl = no and change it to:

ssl = yes

There have been some security vulnerabilities discovered in older versions of the SSL protocol in recent times. SSLv2 is disabled by default, but it doesn't harm to explicitly disable it again. SSLv3 is vulnerable to an attack called [POODLE](http://arstechnica.com/security/2014/10/ssl-broken-again-in-poodle-attack/), so we will disable it too. In the same file, find thessl\_protocols parameter line, uncomment it and add !SSLv3 to the end, like this:

ssl\_protocols = !SSLv2 !SSLv3

For some bizarre reason, the Dovecot package for Raspberry Pi (and possibly newer versions of Ubuntu) does not create a self-signed certificate during installation like it used to. So, we have to create one manually. If you look in /usr/share/dovecot/ you will find the script that used to be used to generate the certificate; we can use it ourselves to simplify the process.

The script is located at /usr/share/dovecot/mkcert.sh and looks like this:

[?](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-2-dovecot)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | #!/bin/sh    # Generates a self-signed certificate.  # Edit dovecot-openssl.cnf before running this.    OPENSSL=${OPENSSL-openssl}  SSLDIR=${SSLDIR-/etc/ssl}  OPENSSLCONFIG=${OPENSSLCONFIG-dovecot-openssl.cnf}    CERTDIR=/etc/dovecot  KEYDIR=/etc/dovecot/private    CERTFILE=$CERTDIR/dovecot.pem  KEYFILE=$KEYDIR/dovecot.pem    if [ ! -d $CERTDIR ]; then    echo "$SSLDIR/certs directory doesn't exist"    exit 1  fi    if [ ! -d $KEYDIR ]; then    echo "$SSLDIR/private directory doesn't exist"    exit 1  fi    if [ -f $CERTFILE ]; then    echo "$CERTFILE already exists, won't overwrite"    exit 1  fi    if [ -f $KEYFILE ]; then    echo "$KEYFILE already exists, won't overwrite"    exit 1  fi    $OPENSSL req -new -x509 -nodes -config $OPENSSLCONFIG -out $CERTFILE -keyout $KEYFILE -days 365 || exit 2  chmod 0600 $KEYFILE  echo  $OPENSSL x509 -subject -fingerprint -noout -in $CERTFILE || exit 2 |

If you were going to use this certificate for any significant length of time, it would be worth editing the parameters in the config file it uses (/usr/share/dovecot/dovecot-openssl.cnf) to set the proper common name and contact details on the certificate. However, I suggest you leave the defaults as they are, use this certificate just for testing, and then come back later and generate a new cert when everything is working (more on that later).

You must be in the same folder as the configuration file when you run the script, or it will not find the config and the certificate generation will fail. The following two commands will change to the right folder and then execute the script:

cd /usr/share/dovecot

sudo ./mkcert.sh

You should see a message "writing new private key to '/etc/dovecot/private/dovecot.pem'" and then some details about the certificate.

Next, find the following two lines in /etc/dovecot/conf.d/10-ssl.conf and **uncomment them**:

#ssl\_cert = </etc/dovecot/dovecot.pem

#ssl\_key = </etc/dovecot/private/dovecot.pem

Now reload dovecot to apply the changes:

sudo service dovecot reload

Since IMAPS is a connection over SSL/TLS, we can’t use Telnet to test it. Instead, we use openssl to create a secure connection. There are two versions of the command, one will show you LOADS of information about the certificate used to encrypt the connection, and the other will suppress this info. I recommend trying the long version out of interest, but both will work the same for the test:

For full information:

openssl s\_client -connect localhost:993

For minimal information:

openssl s\_client -connect localhost:993 -quiet

I won’t print the output of the first command, because it’s ridiculously long. Here’s an example of the second, including a login test:

admin@samhobbs /etc/dovecot/conf.d $ openssl s\_client -connect localhost:993 -quiet

depth=0 O = Dovecot mail server, OU = samhobbs, CN = samhobbs, emailAddress = root@samhobbs.co.uk

verify error:num=18:self signed certificate

verify return:1

depth=0 O = Dovecot mail server, OU = samhobbs, CN = samhobbs, emailAddress = root@samhobbs.co.uk

verify return:1

\* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE AUTH=PLAIN AUTH=LOGIN] Dovecot ready.

a login "testmail" "test1234"

a OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID ENABLE IDLE SORT SORT=DISPLAY THREAD=REFERENCES THREAD=REFS MULTIAPPEND UNSELECT CHILDREN NAMESPACE UIDPLUS LIST-EXTENDED I18NLEVEL=1 CONDSTORE QRESYNC ESEARCH ESORT SEARCHRES WITHIN CONTEXT=SEARCH LIST-STATUS SPECIAL-USE] Logged in

b logout

\* BYE Logging out

b OK Logout completed.

Connection closed by foreign host.

Good stuff: SSL/TLS is working on port 993, and you can log in successfully.

Note that by default Dovecot uses a “snakeoil” self-signed certificate. SSL/TLS certificates are used for two purposes: encryption and verification. The “snakeoil” certificate will encrypt your content but it won’t verify that you’re talking to your server – you could be talking to someone imitating your server (anyone can create a self-signed certificate claiming to be any website).

If you’d like to get your certificate signed without forking out loads of money to a cert signing authority, I’d recommend [CAcert](http://www.cacert.org/). I've written a tutorial explaining how to generate your own cert and get it signed [here](https://samhobbs.co.uk/2014/04/ssl-certificate-signing-cacert-raspberry-pi-ubuntu-debian). If you opt for a commercial certificate, you can use the CAcert tutorial to generate the certificate and then [this tutorial will explain the differences in the installation/configuration of commercial certificates](https://samhobbs.co.uk/2014/08/site-now-ssl-default) once you have it signed.

## Tidying up and enabling WAN access

Before opening the ports on your router to the world, it’s a good idea to delete that test user because the password is so easy to guess.

sudo userdel testmail

Also, if you still use the "pi" login, for goodness' sake change the password from "raspberry"! You can do this using the passwd command when logged in as pi:

passwd

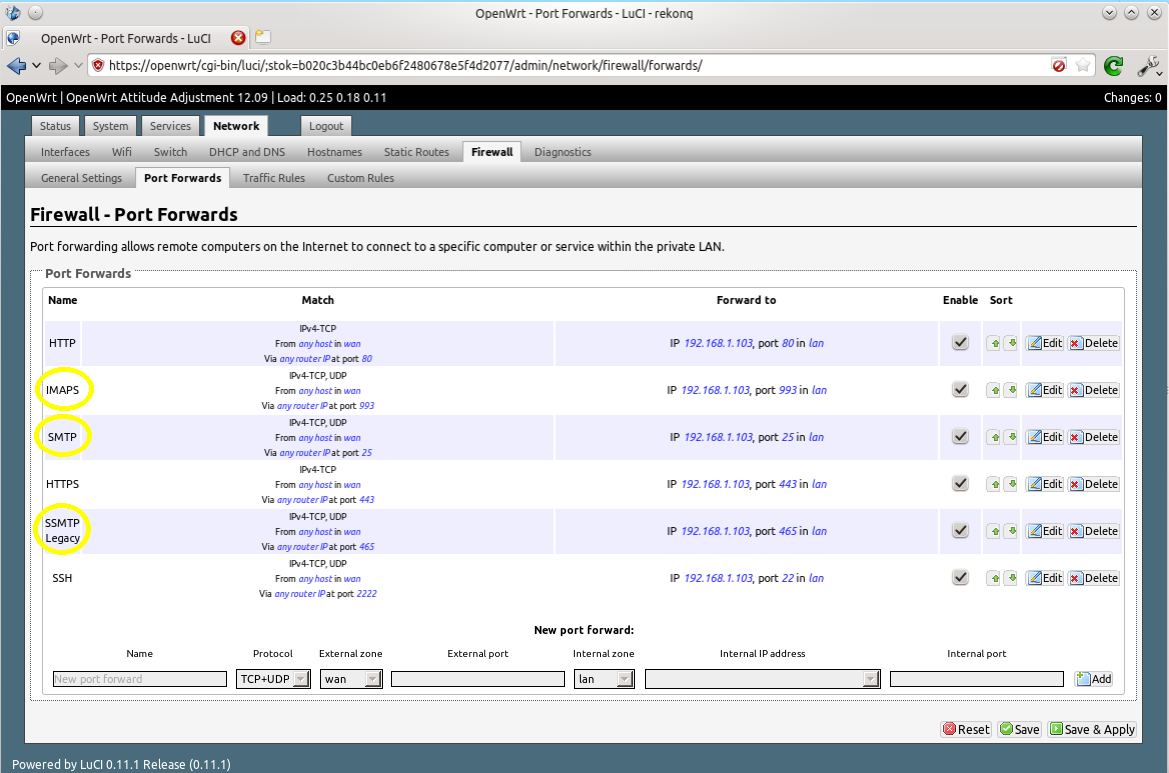
Or you can achieve the same thing when logged in as another user by using sudo to gain root privileges:

sudo passwd pi

Now you can open a few ports on your router’s firewall. Make sure your Pi has a static LAN IP address and then forward these ports from WAN to its LAN IP address:

* Port 25 for SMTP (used for receiving emails)
* Port 465 for secure SMTP (used for sending emails after SASL authentication)
* Port 993 for IMAPS (used to receive emails on your phone/tablet/computer)

Here’s an example on my router, running OpenWrt:



## Setting up IMAP Email Clients

I’m now going to run through setting up IMAP email clients quickly, using K9 Mail on Android and Thunderbird on GNU/Linux as examples. The setup for Thunderbird on Windows and Mac OSX should be very similar.

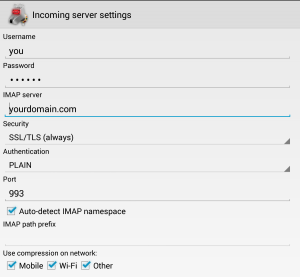
The basics are this:

* Select an IMAP connection
* Your login is your username only (omit @yourdomain.com), and you password is…your password!
* For incoming emails: select use SSL/TLS always and the program should automatically select port 993
* For outgoing emails: select SSL/TLS always. The program may suggest port 587, but you want port 465

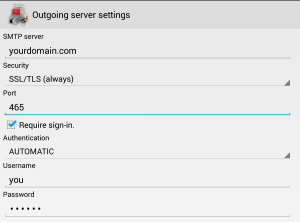
## K9 Mail

Open K9 Mail and select add new account. Type in your account information (you@yourdomain.com and password) and then select manual setup. Select IMAP and then enter your information as follows…

Incoming email:



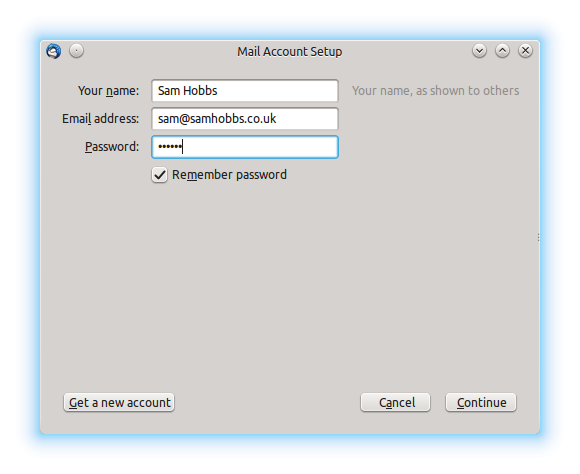
Outgoing email:



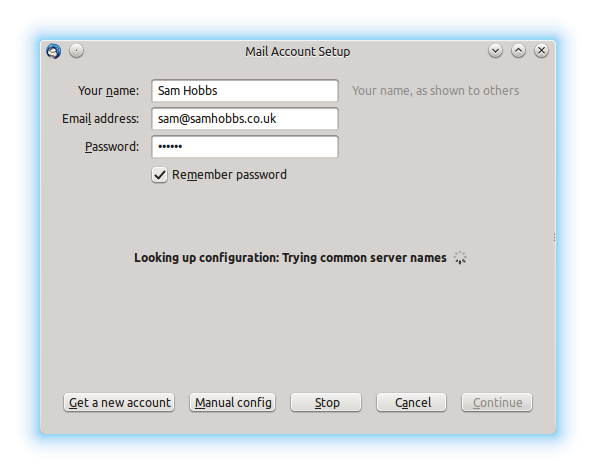
## Thunderbird

Open Thunderbird, and then click **Account Actions –> Add Mail Account.**

Fill in your password and email address, which is your username followed by your fully qualified domain name (FQDN), i.e. username@yourdomain.com:



Thunderbird will try to auto-detect settings and fail. Don’t worry, this is normal. Select “manual config”:



Now edit the settings as appropriate. I had to remove a period (.) from in front of my “server hostname”, and edit the SSL and Authentication settings. If you select “SSL/TLS” for both incoming and outgoing, ports 993 and 465 are automatically selected:



Now try emailing yourself from your external email address, and see if your email gets through. If you are having problems, be sure to check you’ve set up an [MX record](https://samhobbs.co.uk/2015/02/dns-basics-websites-and-email-servers#MX) as well as a [DNS A record](https://samhobbs.co.uk/2015/02/dns-basics-websites-and-email-servers#A).

## Stuck in spam filters?

A few people have contacted me recently to say that their email server is working fine but their emails are getting sent to Gmail's spam folder.

If you are experiencing problems like this (or even if you're not), try setting up an SPF and/or PTR record as explained in my [DNS basics](https://samhobbs.co.uk/2015/02/dns-basics-websites-and-email-servers) tutorial.

You might also want to check if your domain name or IP address are on any blacklists. There's a handy website called [MX toolbox](http://mxtoolbox.com/SuperTool.aspx) that lets you do this (choose blacklist check from the dropdown menu).

## Almost done…

Good news! If you’ve reached this far and everything is working, then you’re almost done. The next step ([Webmail with Squirrelmail](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-3-squirrelmail)) is optional but by far the easiest of the three steps.

If you’ve hit a rut, please post a comment and I’ll try and help you out.

If not… continue to [Raspberry Pi Email Server Part 3: Squirrelmail](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-3-squirrelmail)

**Tags:**

[Raspberry Pi](https://samhobbs.co.uk/tags/raspberry-pi)

[email](https://samhobbs.co.uk/tags/email)

[server](https://samhobbs.co.uk/tags/server)

[dovecot](https://samhobbs.co.uk/tags/dovecot)

[postfix](https://samhobbs.co.uk/tags/postfix)

**Category:**

[Email Server](https://samhobbs.co.uk/taxonomy/term/3)

[Raspberry Pi](https://samhobbs.co.uk/taxonomy/term/1)

**Type:**

[Tutorial](https://samhobbs.co.uk/taxonomy/term/9)

# Part III [Squirrelmail](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-3-squirrelmail)

Sam Hobbs · 22nd December, 2013 · 12:00am



This is the third part of a five part tutorial that will show you how to install a full featured email server on your Raspberry Pi. This tutorial covers how to set up Webmail with Squirrelmail.

The parts are:  
[The Introduction & Contents Page (read first)](https://samhobbs.co.uk/raspberry-pi-email-server)  
[Raspberry Pi Email Server Part 1: Postfix](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-1-postfix)  
[Raspberry Pi Email Server Part 2: Dovecot](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-2-dovecot)  
[Raspberry Pi Email Server Part 3: Squirrelmail](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-3-squirrelmail)  
[Raspberry Pi Email Server Part 4: Spam Detection with Spamassassin](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-4-spam-detection-with-spamassassin)  
[Raspberry Pi Email Server Part 5: Spam Sorting with LMTP & Sieve](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-5-spam-sorting-with-lmtp-sieve)

## Installing Apache

If you don't already have apache installed (you might if you've followed my [wordpress tutorial](https://samhobbs.co.uk/2014/02/how-to-install-wordpress-on-a-raspberry-pi)), then install it now.

sudo apt-get update

sudo apt-get install apache2

Enable the SSL apache module so that you can use HTTPS:

sudo a2enmod ssl

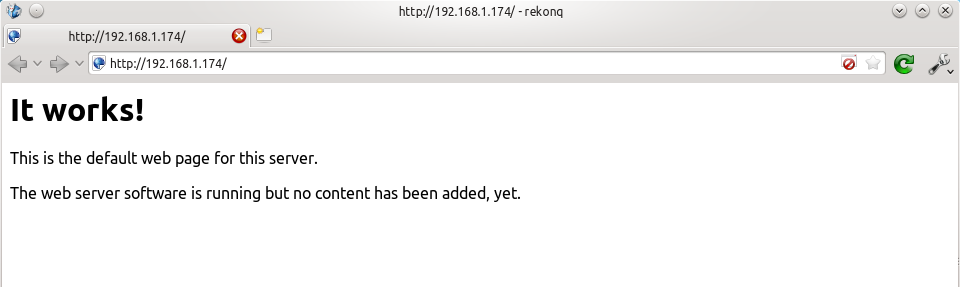
There are some "pre-made" virtualhost configurations that come with apache. This command will enable the "default-ssl" virtualhost, by creating a symbolic link from /etc/apache2/sites-available/default-ssl to /etc/apache2/sites-enabled/default-ssl:

sudo a2ensite default-ssl

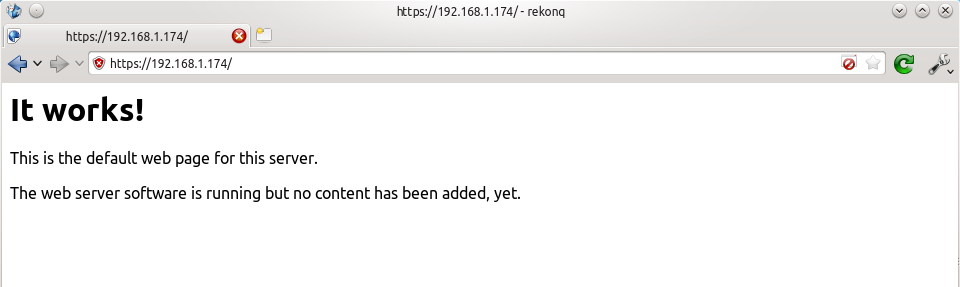
Now reload apache to make the changes take effect:

sudo service apache2 reload

If you type the IP address or hostname of the pi into a web browser now, you should see the default Apache test page:



If you try the https version, you'll get a certificate error because you are using a self-signed SSL certificate. If you like, you can follow my [CAcert tutorial](https://samhobbs.co.uk/2014/04/ssl-certificate-signing-cacert-raspberry-pi-ubuntu-debian) to get a free SSL certificate for your domain, or you can just store an exception for the certificate and generate a proper one later.



That's it for Apache. If you want to know more about setting up Apache for multiple websites, subdomains, and SSL configurations, I suggest you read [my tutorial explaining Apache's VirtualHost files](https://samhobbs.co.uk/2014/01/multiple-websites-and-subdomains-with-ssltls-in-apache2-virtualhosts).

## Installing Squirrelmail

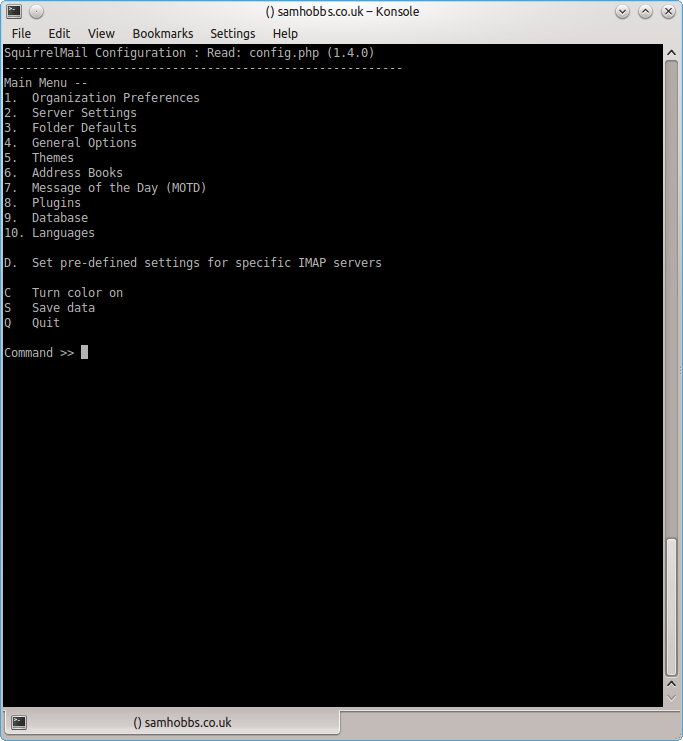
Now we need to install squirrelmail:

sudo apt-get update

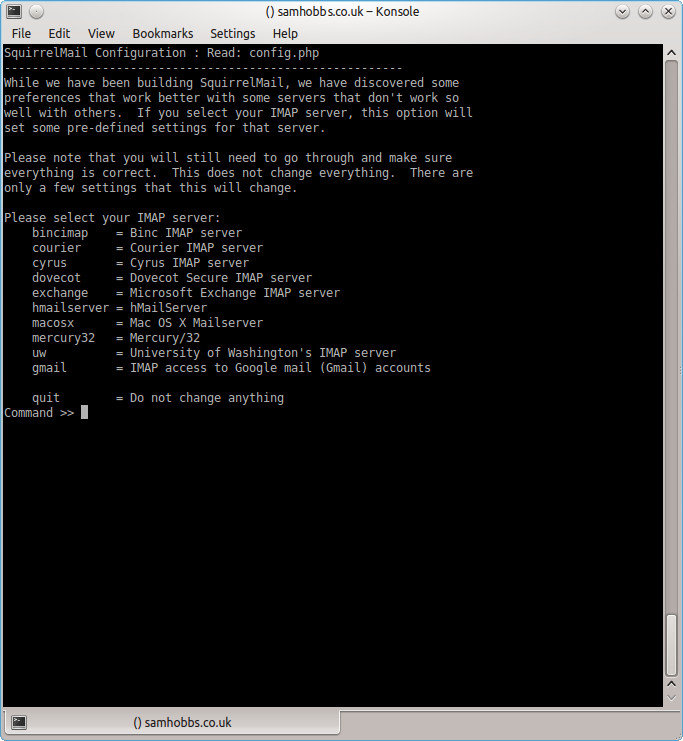
sudo apt-get install squirrelmail

The basic configuration for squirrelmail is really easy, and can be done with the setup script. To run the script, use this command:

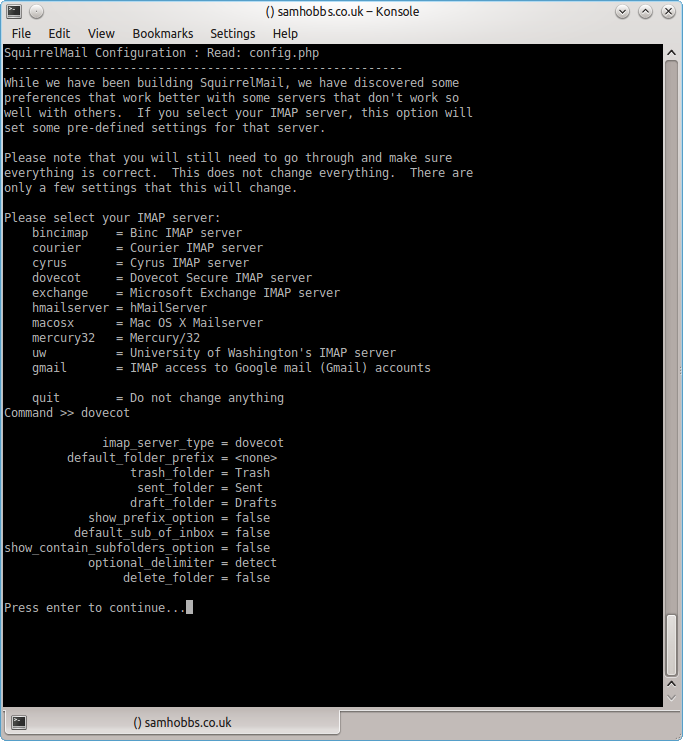
sudo squirrelmail-configure



Choose “D” for pre-defined settings



Now type “dovecot” and hit enter



Press enter to continue, then save and quit (press Q and save when prompted, or press S then Q).

The configuration script creates a configuration file for apache in /etc/squirrelmail/apache.conf. You need to create a symbolic link so that Apache2 will load your Squirrelmail apache configuration file when it starts up.

On Raspbian Wheezy, the command is:

sudo ln -s /etc/squirrelmail/apache.conf /etc/apache2/conf.d/squirrelmail.conf

On Raspbian Jessie, the apache configuration directory structure is more like ubuntu, with separate folders for configuration files that are available and files that are enabled. This command will create a symlink from the directory where enabled configuration is stored, to squirrelmail's apache configuration file:

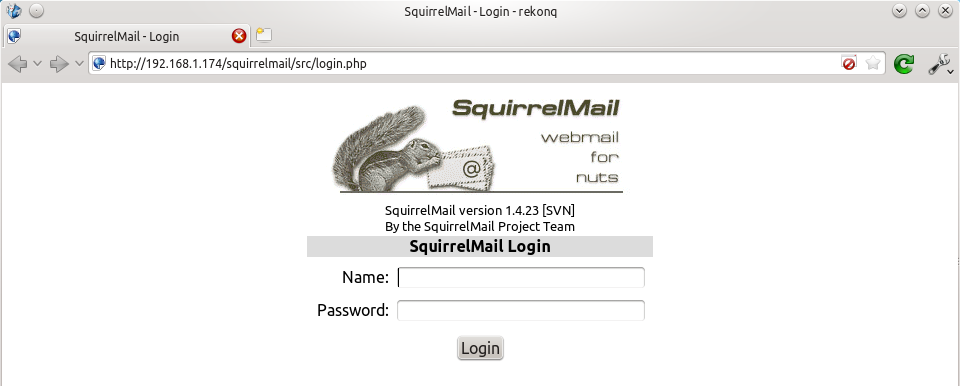
sudo ln -s /etc/squirrelmail/apache.conf /etc/apache2/conf-enabled/squirrelmail.conf

On a related note, there's a convenience command a2enconf that works similarly to a2ensite: it creates a symlink from the conf-available directory to the conf-enabled directory. You should use this in situations where config already exists in conf-available, e.g.sudo a2enconf squirrelmail would create a symlink for a file called squirrelmail.conf.

Now reload Apache one more time so that it reads the config file we just symlinked:

sudo service apache2 reload

Now visit the IP address or hostname of your Pi again, but add /squirrelmail to the path, e.g. **192.168.1.174/squirrelmail**, you should see the login page:



The squirrelmail configuration file just adds an alias that should affect every virtualhost, so if you install a wordpress site or something like that on your pi, you will be able to get to the squirrelmail login page by visiting **yourdomain.com/squirrelmail**.

## Redirect http to https for secure login

Since you don't want to send your login details and confidential information over the internet without SSL, it's best to redirect all http URLs to https.

The default squirrelmail apache configuration file at /etc/squirrelmail/apache.conf contains some rewrite rules we can use, we just need to uncomment them. Open the file and uncomment the lines by removing the # at the start of each so that it looks like this:

<IfModule mod\_rewrite.c>

<IfModule mod\_ssl.c>

<Location /squirrelmail>

RewriteEngine on

RewriteCond %{HTTPS} !^on$ [NC]

RewriteRule . https://%{HTTP\_HOST}%{REQUEST\_URI} [L]

</Location>

</IfModule>

</IfModule>

This configuration makes sure that the **rewrite** and **ssl** modules are enabled, and does nothing if they aren't. We already enabled the **ssl**module earlier, so all we need to do now is enable the **rewrite** module:

sudo a2enmod rewrite

And reload Apache:

sudo service apache2 reload

Now if you visit the HTTP page, you should be redirected to HTTPS.

## Optional: Configuring Apache to serve Squirrelmail on a subdomain

If you would like to move the login page to the root of your domain (i.e. so that yourdomain.com would serve the login page for squirrelmail), or if you would like to serve it on a subdomain like mail.yourdomain.com, you can edit the configuration file. If not, you can skip this section.

By default, this line in /etc/squirrelmail/apache.conf means that http://www.yourdomain.com/squirrelmail will load squirrelmail:

Alias /squirrelmail /usr/share/squirrelmail

If you would rather have webmail on a subdomain like mail.yourdomain.com then you could edit the /etc/squirrelmail/apache.conf file to look like this (comment out the rest):

<VirtualHost \*:80>

DocumentRoot /usr/share/squirrelmail

ServerName mail.yourdomain.com

<Directory /usr/share/squirrelmail>

Options FollowSymLinks

<IfModule mod\_php5.c>

php\_flag register\_globals off

</IfModule>

<IfModule mod\_dir.c>

DirectoryIndex index.php

</IfModule>

# access to configtest is limited by default to prevent information leak

<Files configtest.php>

order deny,allow

deny from all

allow from 127.0.0.1

</Files>

</Directory>

</VirtualHost>

Note that if you want to serve mail on a subdomain, then that subdomain needs a DNS record, so edit your records with your DNS provider accordingly.

However, the virtualhost code above only does HTTP. You also want an HTTPS virtualhost for the subdomain on port 443:

<IfModule mod\_ssl.c>

<VirtualHost \*:443>

DocumentRoot /usr/share/squirrelmail

ServerName mail.yourdomain.com

<Directory /usr/share/squirrelmail>

Options FollowSymLinks

<IfModule mod\_php5.c>

php\_flag register\_globals off

</IfModule>

<IfModule mod\_dir.c>

DirectoryIndex index.php

</IfModule>

# access to configtest is limited by default to prevent information leak

<Files configtest.php>

order deny,allow

deny from all

allow from 127.0.0.1

</Files>

</Directory>

ErrorLog ${APACHE\_LOG\_DIR}/error.log

LogLevel warn

CustomLog ${APACHE\_LOG\_DIR}/ssl\_access.log combined

SSLEngine on

SSLCertificateFile /etc/ssl/certs/your-ssl-certificate.crt

SSLCertificateKeyFile /etc/ssl/private/your-ssl-certificate-keyfile.key

</VirtualHost>

</IfModule>

If you use this configuration, the rewrite rules from the "rewrite to HTTPS" section (the ones from the default config file) won't work. If you want an HTTPS-only solution, you can replace the virtualhost for port 80 with this:

<VirtualHost \*:80>

ServerName mail.yourdomain.com

<IfModule mod\_rewrite.c>

<IfModule mod\_ssl.c>

<Location />

RewriteEngine on

RewriteCond %{HTTPS} !^on$ [NC]

RewriteRule . https://%{HTTP\_HOST}%{REQUEST\_URI} [L]

</Location>

</IfModule>

</IfModule>

</VirtualHost>

As before, make sure you have the rewrite module enabled:

sudo a2enmod rewrite

Now reload apache:

sudo service apache2 restart

If you get an error like this:

[....] Restarting web server: apache2[Fri Dec 06 15:54:04 2013] [warn] \_default\_ VirtualHost overlap on port 443, the first has precedence

Then add NameVirtualHost \*:443 to the start of the SSL VirtualHost block, i.e.:

NameVirtualHost \*:443

<IfModule mod\_ssl.c>

<VirtualHost \*:443>

DocumentRoot /usr/share/squirrelmail

ServerName mail.yourdomain.com

...

For more info on HTTP and HTTPS VirtualHost configuration on Apache2, see [this tutorial of mine](https://samhobbs.co.uk/2014/01/multiple-websites-and-subdomains-with-ssltls-in-apache2-virtualhosts).

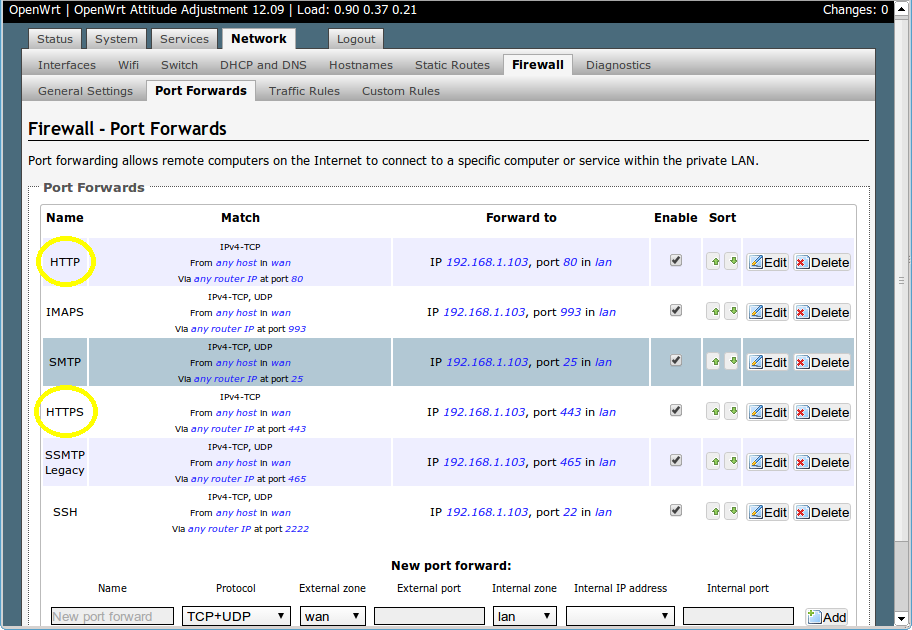
If all went to plan then you can navigate to mail.yourdomain.com and you should see the squirrelmail login page (you might need to forward some ports on your router if you haven't already - see the next section):



## Port Forwarding

Squirrelmail will log in to your IMAP server on port 143 to display your emails. It doesn’t need to authenticate because it’s in your network (remember the permit\_mynetworks parameter from the previous tutorials?). You don’t need to worry about it being an unencrypted connection, because the data isn't actually travelling over any insecure networks (the connection is internal). Similarly, you don’t need to open port 143 (“plain” imap without SSL/TLS) on your router because the connection happens within the Pi, and the content is actually served to you, the user, on port 443 (https).

So you **do** need to open ports 80 and 443 for http and https like below:



## Testing

Before you start testing your webmail, make sure that the permit\_mynetworks parameters are uncommented in your postfix confuguration file/etc/postfix/main.cf (in both your smtpd\_recipient\_restrictions and smtpd\_helo\_restrictions).

Now reload your postfix configuration:

sudo service postfix reload

Try sending and receiving emails from within Squirrelmail. You should have no problems, but if you do please post a comment and I’ll try to help you out.

## Customising the Squirrelmail Login

To customise the login page, run the configuration wizard:

sudo squirrelmail-configure

1. Select “1″ (organisation preferences)
2. Select “7″ and change to your domain (e.g. http://www.samhobbs.co.uk)
3. Select “8″ and change to you/your organisation’s name

## Squirrelmail Plugins

There are loads of plugins available for Squirrelmail, for all kinds of things. On most systems, these plugins are installed by downloading a .zip file to your server, unzipping the plugin to the right location and then tinkering with the settings manually.

Thankfully, some of the most commonly used plugins are available from the Raspbian repositories, so installation is much simpler. Here is a list of the plugins in the repo:

1. squirrelmail-compatibility
2. squirrelmail-decode
3. squirrelmail-locales
4. squirrelmail-lockout
5. squirrelmail-logger
6. squirrelmail-quicksave
7. squirrelmail-secure-login
8. squirrelmail-sent-confirmation
9. squirrelmail-spam-buttons
10. squirrelmail-viewashtml

The one I think is most useful and the one I’m going to use as an example is **lockout**.

To use the lockout package, we need to install the compatibility package, which basically makes sure that plugins built for different versions of squirrelmail can still work with the version you are running.

sudo apt-get update

sudo apt-get install squirrelmail-compatibility

Now we need to enable the plugin:

sudo squirrelmail-configure

1. select “8″
2. select “compatibility”
3. select “S” (to save)
4. select “Q” (to quit)

That’s all you need to do for the compatibility plugin. Now we can install the lockout plugin:

sudo apt-get install squirrelmail-lockout

Now for the configuration:

sudo squirrelmail-configure

Make sure that lockout is enabled

Now we can manually edit some settings. Before starting, I like to back up the default config files for reference:

cd /etc/squirrelmail

sudo cp lockout-table.php lockout-table.php.BAK

sudo cp lockout-config.php lockout-config.php.BAK

Now edit the lockout-table.php file. Read the comments in the file for an explanation of how the table works. I wanted to disable logins for the user “admin”, so I commented out the examples at the end of the file and replaced them with this:

user: admin locked\_out.php

Now edit lockout-config.php and set $use\_lockout\_rules = 1; to turn on lockouts.

Now try and log in as the user you locked out, and you should get this message: “Access Denied / Please contact your system administrator”.

We can also lock out IP addresses of users who enter incorrect username/password combinations repeatedly. To do this, open lockout-config.php and set $max\_login\_attempts\_per\_IP = '3:5:0'.

The first number in this parameter is the number of incorrect attempts that are allowed before a ban. The second number is the time frame for these incorrect attempts, and the last number is the amount of time the ban lasts for when activated (0 is forever).

So, the setting I gave you above means that anyone who makes 3 incorrect attempts to authenticate in 5 minutes is permanently banned.

When a successful login is made, the count is reset to 0.

Data on current bad login attempts and bans is stored here:/var/lib/squirrelmail/data/lockout\_plugin\_login\_failure\_information

The plugin will add entries like this to keep track of bad logins:

999.999.99.99\_login\_failure\_times=1386774015:1386774034:1386774053

999.999.99.99\_TOO\_MANY\_FAILED\_LOGIN\_ATTEMPTS=PERMANENT

…where 999.999.99.99 is the offending IP address

If you accidentally ban yourself, you’ll have to log in via SSH and edit this file to remove those lines.

That’s it, you’re done! Have fun exploring the other plugins!

The next two tutorials, [Part 4](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-4-spam-detection-with-spamassassin) and [Part 5](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-5-spam-sorting-with-lmtp-sieve) deal with spam detection and filtering.

**Tags:**

[Raspberry Pi](https://samhobbs.co.uk/tags/raspberry-pi)

[email](https://samhobbs.co.uk/tags/email)

[server](https://samhobbs.co.uk/tags/server)

[squirrelmail](https://samhobbs.co.uk/tags/squirrelmail)

[apache2](https://samhobbs.co.uk/tags/apache2)

**Category:**

[Email Server](https://samhobbs.co.uk/taxonomy/term/3)

[Raspberry Pi](https://samhobbs.co.uk/taxonomy/term/1)

**Type:**

[Tutorial](https://samhobbs.co.uk/taxonomy/term/9)

# Part IV spam detect

Sam Hobbs · 5th March, 2014 · 12:00am



This is the fourth part of a five part tutorial that will show you how to install a full featured email server on your Raspberry Pi. This tutorial covers how to mark emails as spam with Spamassassin.

The parts are:  
[The Introduction & Contents Page (read first)](https://samhobbs.co.uk/raspberry-pi-email-server)  
[Raspberry Pi Email Server Part 1: Postfix](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-1-postfix)  
[Raspberry Pi Email Server Part 2: Dovecot](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-2-dovecot)  
[Raspberry Pi Email Server Part 3: Squirrelmail](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-3-squirrelmail)  
[Raspberry Pi Email Server Part 4: Spam Detection with Spamassassin](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-4-spam-detection-with-spamassassin)  
[Raspberry Pi Email Server Part 5: Spam Sorting with LMTP & Sieve](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-5-spam-sorting-with-lmtp-sieve)

## Intro

I don’t actually get very many spam emails (famous last words, right?) but the occasional email gets past my helo access restrictions list (discussed in [Raspberry Pi Email Server Part 1: Postfix](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-1-postfix)).

So, I decided to set up Spamassassin, a program that will check incoming emails and mark them as spam if they look suspicious. Spamassassin is pretty clever, it uses [bayesian filtering](https://en.wikipedia.org/wiki/Bayesian_spam_filtering) to decide what’s spam and what’s not, and **it will learn based on previous results**, so it gets more accurate over time if you correct it when it gets things wrong.

Spamassassin will only mark emails as spam, it will not sort them into folders for you as well. We’ll be doing the sorting with Dovecot’s Local Mail Transfer Protocol (LMTP) and the Sieve plugin, in the next tutorial: [Raspberry Pi Email Server Part 5: Spam Sorting with LMTP & Sieve](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-5-spam-sorting-with-lmtp-sieve).

Let’s get started:

## Installing & Configuring Spamassassin

First, install Spamassassin:

sudo apt-get update

sudo apt-get install spamassassin

Now we need to edit values in the file /etc/spamassassin/local.cf. Some of these may already be set, in which case you can leave them as they are; add or amend the others as necessary:

This one will add the spam score to the subject line of emails that Spamassassin considers to be spam:

rewrite\_header Subject [\*\*\*\*\* SPAM \_SCORE\_ \*\*\*\*\*]

Spamassassin will also flag spam emails with **“X-Spam-Flag: YES”** in the headers. This flag is what we will eventually use to sort emails with; the rewritten subject line is purely to make the score easier to see.

This next setting will tell Spamassassin to modify headers only, without making any changes to the body of the email:

report\_safe 0

This one lowers the threshold for mail to be considered spam from 5 to 2. You can change this later if you get lots of false positives, but it’s nice to have some emails set off the rules to begin with, just so you know it’s working:

required\_score 2.0

This tells Spamassassin to use Bayesian filtering:

use\_bayes 1

This turns on automatic learning:

bayes\_auto\_learn 1

Now edit **/etc/default/spamassassin** and set:

ENABLED=1

You can now start the spamassassin daemon:

sudo service spamassassin start

## Instructing Postfix to use Spamassassin

At this stage, the Spamassassin daemon is running but none of your incoming emails are being passed through it. We need to edit this line in/etc/postfix/master.cf (just under the headers):

smtp inet n - - - - smtpd

-o content\_filter=spamassassin

And append this to the bottom of that same file, which will pipe the output back to Postfix using the Postfix’s [Sendmail compatibility interface](http://www.postfix.org/sendmail.1.html):

spamassassin unix - n n - - pipe user=debian-spamd argv=/usr/bin/spamc -f -e /usr/sbin/sendmail -oi -f ${sender} ${recipient}

Note: this is all one line, even if it appears wrapped in your browser.

Now restart postfix:

sudo service postfix restart

If you get an error like this:

[....] Stopping Postfix Mail Transport Agent: postfix/usr/sbin/postconf: fatal: file /etc/postfix/master.cf: line 22: bad field count

postfix/postfix-script: fatal: cannot execute /usr/sbin/postconf!

failed!

…then check the whitespace before the -o in content\_filter=spamassassin. I can’t quite remember what I did but I think I changed tabs to spaces or the other way round, and then restarted postfix.

Now watch the mail log with this command:

tail -f /var/log/mail.log

…and send a test email. You should see testing something like this:

Jan 8 22:21:18 samhobbs postfix/smtpd[952]: connect from blu0-omc2-s3.blu0.hotmail.com[65.55.111.78]

Jan 8 22:21:19 samhobbs postfix/smtpd[952]: 542E83F519: client=blu0-omc2-s3.blu0.hotmail.com[65.55.111.78]

Jan 8 22:21:19 samhobbs postfix/cleanup[957]: 542E83F519: message-id=

Jan 8 22:21:19 samhobbs postfix/qmgr[941]: 542E83F519: from=, size=1579, nrcpt=1 (queue active)

Jan 8 22:21:19 samhobbs spamd[445]: spamd: connection from localhost [127.0.0.1] at port 35680

Jan 8 22:21:19 samhobbs postfix/smtpd[952]: disconnect from blu0-omc2-s3.blu0.hotmail.com[65.55.111.78]

Jan 8 22:21:19 samhobbs spamd[445]: spamd: setuid to debian-spamd succeeded

Jan 8 22:21:19 samhobbs spamd[445]: spamd: creating default\_prefs: /var/lib/spamassassin/.spamassassin/user\_prefs

Jan 8 22:21:19 samhobbs spamd[445]: config: created user preferences file: /var/lib/spamassassin/.spamassassin/user\_prefs

Jan 8 22:21:19 samhobbs spamd[445]: spamd: processing message for debian-spamd:111

Jan 8 22:21:24 samhobbs spamd[445]: spamd: clean message (0.0/2.0) for debian-spamd:111 in 5.0 seconds, 1541 bytes.

Jan 8 22:21:24 samhobbs spamd[445]: spamd: result: . 0 - HTML\_MESSAGE,MSGID\_FROM\_MTA\_HEADER scantime=5.0,size=1541,user=debian-spamd,uid=111,required\_score=2.0,rhost=localhost,raddr=127.0.0.1,rport=35680,mid=,autolearn=ham

Jan 8 22:21:24 samhobbs postfix/pickup[940]: D83DE3F521: uid=111 from=

Jan 8 22:21:24 samhobbs postfix/pipe[958]: 542E83F519: to=, relay=spamassassin, delay=5.7, delays=0.44/0.05/0/5.2, dsn=2.0.0, status=sent (delivered via spamassassin service)

Jan 8 22:21:24 samhobbs postfix/qmgr[941]: 542E83F519: removed

Jan 8 22:21:24 samhobbs postfix/cleanup[957]: D83DE3F521: message-id=

Jan 8 22:21:24 samhobbs postfix/qmgr[941]: D83DE3F521: from=, size=1890, nrcpt=1 (queue active)

Jan 8 22:21:25 samhobbs postfix/local[964]: D83DE3F521: to=, relay=local, delay=0.2, delays=0.06/0.1/0/0.03, dsn=2.0.0, status=sent (delivered to maildir)

Jan 8 22:21:25 samhobbs postfix/qmgr[941]: D83DE3F521: removed

Jan 8 22:21:25 samhobbs spamd[439]: prefork: child states: II

So the steps you can see here are:

1. Outlook server connects to RasPi/Postfix on port 25
2. Postfix accepts the message and hands it to Spamassassin to process
3. Spamassassin decides the message is clean and marks it as HAM
4. The email is passed back from Spamassassin to Postfix and delivered to the inbox

## Training Spamassassin

We’ve deliberately set the score limit for spam to a low value. This inevitably means we’ll get some false positives, but we can use these to train Spamassassin and make it better.

First, some things to understand about the Maildir format we’re using. Here’s what my structure looks like:

admin@samhobbs ~ $ sudo ls -al /home/sam/Maildir/

total 604

drwx------ 12 sam sam 4096 Mar 6 14:55 .

drwxr-xr-x 3 sam sam 4096 Mar 5 23:07 ..

drwx------ 2 sam sam 36864 Mar 6 12:59 cur

-rw------- 1 sam sam 11920 Mar 6 04:14 dovecot.index

-rw------- 1 sam sam 415744 Mar 6 14:50 dovecot.index.cache

-rw------- 1 sam sam 10332 Mar 6 13:08 dovecot.index.log

-rw------- 1 sam sam 32784 Mar 5 16:22 dovecot.index.log.2

-rw------- 1 sam sam 30 Jan 13 22:30 dovecot-keywords

-rw------- 1 sam sam 144 Mar 3 17:49 dovecot.mailbox.log

-rw------- 1 sam sam 27138 Mar 6 09:27 dovecot-uidlist

-rw------- 1 sam sam 8 Mar 5 23:07 dovecot-uidvalidity

-r--r--r-- 1 sam sam 0 Nov 23 22:55 dovecot-uidvalidity.52913278

drwx------ 5 sam sam 4096 Mar 5 22:36 .Drafts

drwx------ 5 sam sam 4096 Mar 4 21:53 .foo

drwx------ 5 sam sam 4096 Mar 3 17:49 .INBOX.foo

drwx------ 2 sam sam 4096 Mar 6 09:37 new

drwx------ 5 sam sam 4096 Mar 5 22:36 .Sent

drwx------ 5 sam sam 4096 Mar 6 14:37 .Spam

-rw------- 1 sam sam 37 Mar 3 17:49 subscriptions

drwx------ 5 sam sam 4096 Nov 27 19:00 .Templates

drwx------ 2 sam sam 4096 Mar 6 09:27 tmp

drwx------ 5 sam sam 4096 Mar 6 04:08 .Trash

You can see I’ve created a couple of test folders here: one top level folder called “foo” and another subfolder in the inbox also called “foo” (.INBOX.foo). Each folder has three subdirectories: **new** for new (unread) emails, **cur** for emails that have been read, and **tmp** for temporary storage during delivery.

You can read more about this on the [Dovecot Wiki](http://wiki2.dovecot.org/MailboxFormat/Maildir) if you’d like to know more.

So, the important thing to take away from this is that HAM emails are stored here:  
/home/username/Maildir/cur

…and SPAM emails are stored here:  
/home/username/Maildir/.Spam/cur

Spamassassin has a commandline training tool that is invoked like this:

sa-learn --no-sync [--spam or --ham] [folder/{cur,new}]

Here’s the plan: move any false positives back into the inbox with your email client, and move any missed spam into the spam folder. Then run these three commands:

# Scan HAM

sudo sa-learn --no-sync --ham /home/username/Maildir/{cur,new}

# Scan SPAM

sudo sa-learn --no-sync --spam /home/username/Maildir/.Spam/{cur,new}

# sync the journal and databases

sa-learn --sync

On my Pi, running the HAM command took about 5mins to process ~500 messages, with WordPress running at the same time. If you’re sure you will always move emails into the correct folders, you could add these two commands to a cron job so that they run regularly and keep everything up to date.

Alternatively, you can just run the commands when you notice a few false positives or missed spam emails. Over time, your spam filter will get better and better.

## Automated learning using a script

If you don't want to run the commands manually all the time, you can use this simple script I wrote. Create the script like this:

sudo nano /etc/cron.daily/spamassassin-learn

Now copy and paste this into the file (ctrl + shift + v to paste in nano):

[?](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-4-spam-detection-with-spamassassin)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | #!/bin/bash    # Script by Sam Hobbs [https://samhobbs.co.uk](https://samhobbs.co.uk/)    # redirect errors and output to logfile  exec 2>&1 1>> /var/log/spamassassin.log    NOW=$(date +"%Y-%m-%d")    # Headers for log  echo ""  echo "#================================ $NOW ================================#"  echo ""    # learn HAM  echo "Learning HAM from Inbox"  sa-learn --no-sync --ham /home/sam/Maildir/{cur,new}    # learn SPAM  echo "Learning SPAM from Spam folder"  sa-learn --no-sync --spam /home/sam/Maildir/.Spam/{cur,new}    # Synchronize the journal and databases.  echo "Syncing"  sa-learn --sync |

**Important: edit the paths so that they match your username**!

The script will learn from ham/spam daily, and write a log file at /var/log/spamassassin.log. Make sure you move any spam you find into your spam folder, and any false positives back into your inbox. Don't worry if ham is accidentally marked as spam one day and gets "learned", if you move the messages to their correct locations then the next time the script runs spamassassin will correct itself.

## What’s next?

We’re now done with Spamassassin. The only thing left to do is find a way to sort spam emails directly into the spam folder, which is covered in the next tutorial: [Raspberry Pi Email Server Part 5: Spam Sorting with LMTP & Sieve](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-5-spam-sorting-with-lmtp-sieve).

Feel free to leave a comment to let me know how you get on!

**Tags:**

[Raspberry Pi](https://samhobbs.co.uk/tags/raspberry-pi)

[email](https://samhobbs.co.uk/tags/email)

[server](https://samhobbs.co.uk/tags/server)

[spamassassin](https://samhobbs.co.uk/tags/spamassassin)

[spam](https://samhobbs.co.uk/tags/spam)

[postfix](https://samhobbs.co.uk/tags/postfix)

**Category:**

[Email Server](https://samhobbs.co.uk/taxonomy/term/3)

[Raspberry Pi](https://samhobbs.co.uk/taxonomy/term/1)

**Type:**

[Tutorial](https://samhobbs.co.uk/taxonomy/term/9)

# Part V spam sorting

Sam Hobbs · 25th March, 2014 · 12:00am

This is the **fifth and final part** of a five part tutorial that will show you how to install a full featured email server on your Raspberry Pi.

This tutorial covers how to automatically sort spam emails into the spam folder using Dovecot’s Local Mail Transfer Protocol (LMTP) and Sieve rules.

The parts are:  
[The Introduction & Contents Page (read first)](https://samhobbs.co.uk/raspberry-pi-email-server)  
[Raspberry Pi Email Server Part 1: Postfix](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-1-postfix)  
[Raspberry Pi Email Server Part 2: Dovecot](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-2-dovecot)  
[Raspberry Pi Email Server Part 3: Squirrelmail](https://samhobbs.co.uk/2013/12/raspberry-pi-email-server-part-3-squirrelmail)  
[Raspberry Pi Email Server Part 4: Spam Detection with Spamassassin](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-4-spam-detection-with-spamassassin)  
[Raspberry Pi Email Server Part 5: Spam Sorting with LMTP & Sieve](https://samhobbs.co.uk/2014/03/raspberry-pi-email-server-part-5-spam-sorting-with-lmtp-sieve)

## Intro

If you followed the previous tutorial, you currently have an email server that automatically scans incoming emails using Spamassassin. However, in its current state, Spam and Ham alike are delivered to the inbox, which is annoying. Since Spamassassin only marks emails based on their spam score, we need to use an external program to handle sorting & delivery.

The best tool for the job is Sieve: a set of rules defined by each user to determine how incoming emails are filtered. Sieve rules can sort emails based on a myriad of things: headers, the body of the email, various tags added by external programs, sender address… the list goes on.

In this case, we are going to use Sieve rules to send emails that have been marked by Spamassassin with spam flags like this **(“X-Spam-Flag: YES“)** in the headers straight into the Spam folder.

At present, emails are delivered to the inbox by Postfix. However, since Sieve is a Dovecot plugin, the final step of the delivery must be handled by Dovecot. This gives us two choices:

1. Use Dovecot’s Local Delivery Agent (LDA)
2. Use Dovecot’s Local Mail Transfer Protocol (LMTP)

For our setup, LDA is not ideal because it doesn’t run as root and therefore can’t access the individual inboxes of each user due to permissions. There are some other differences too…from the [Dovecot Wiki](http://wiki2.dovecot.org/LMTP):

The main difference is that the LDA is a short-running process, started as a binary from command line, while LMTP is a long-running process started by Dovecot’s master process.

In other words, with LDA a new process is started each time an email needs to be delivered, whereas with LMTP a process is always running and it handles a queue of emails. From what I have read on mailing lists etc. it seems that LMTP is more efficient.

So… LMTP it is!

## Install & Configure Dovecot LMTP

First, install dovecot-lmtpd:

sudo apt-get update

sudo apt-get install dovecot-lmtpd

This will create a new config file at /etc/dovecot/conf.d/20-lmtp.conf.

Now to change the config in a few files:

## /etc/dovecot/dovecot.conf

Append this to enable lmtp:

protocols = imap lmtp

## /etc/dovecot/conf.d/20-lmtp.conf

Add this line to enable address extensions:

lmtp\_save\_to\_detail\_mailbox = yes

This means that if you send an email to you+folder@yourdomain.com it should be automatically placed in the “folder” folder. Cool, eh? You can use this for loads of things, but here’s a typical student example from a recent graduate ;) …

Change your email address for Pizza takeaway companies to you+pizza@yourdomain.com and create a folder called “pizza”. Now all your emails about pizza go into a separate folder, instead of cluttering your inbox. Awesome :)

Note: folder names are **case sensitive**, and the folder must be top level (not a folder within your inbox).

Now change the lmtp protocol block to look like this:

protocol lmtp {

mail\_plugins = $mail\_plugins sieve

postmaster\_address = postmaster@yourdomain.com

}

## /etc/dovecot/conf.d/10-master.conf

Now find the service lmtp {… block and then change the line unix\_listener lmtp {… to look like this. This will allow postfix to access Dovecot’s LMTP from within its chroot:

service lmtp {

unix\_listener /var/spool/postfix/private/dovecot-lmtp {

mode = 0666

}

}

## /etc/dovecot/conf.d/10-auth.conf

By default, Dovecot will try to look up “you@yourdomain.com” in your user database, when it should be looking up just the first bit (“you”). This setting instructs Dovecot to strip the domain name before doing the lookup, and convert the username to all lowercase letters:

auth\_username\_format = %Ln

(the L is the lowercase part and the n drops the domain name).

## /etc/dovecot/conf.d/10-director.conf

I’m not sure if this bit is necessary, but I commented out the “protocol lmtp {…” block completely.

## /etc/postfix/main.cf

We still need to instruct Postfix to hand over control to Dovecot’s LMTP for the final stage of delivery.

Comment out:

mailbox\_command=

…and add:

mailbox\_transport = lmtp:unix:private/dovecot-lmtp

## Sieve Rules

Dovecot's sieve is already installed, you can check by running:

sudo apt-get install dovecot-sieve

Now we need to change one more parameter in /etc/dovecot/conf.d/90-sieve.conf:

Uncomment this setting:

recipient\_delimiter = +

We still need to reload/restart Postfix and Dovecot to make that all the changes are loaded:

sudo service postfix reload

sudo service dovecot reload

The default place to put the sieve script is in the user's home folder: ~/.dovecot.sieve.

Create it like this:

sudo nano /home/user/.dovecot.sieve

and add this:

require ["fileinto"];

# Move spam to spam folder

if header :contains "X-Spam-Flag" "YES" {

fileinto "Spam";

# Stop here - if there are other rules, ignore them for spam messages

stop;

}

Or, if you want spam messages to be marked as read as well as moved:

require ["fileinto","imap4flags"];

if header :contains "X-Spam-Flag" "YES" {

addflag "\\Seen";

fileinto "Spam";

stop;

}

Now chown the file to the owner of the mailbox, e.g.:

sudo chown sam:sam /home/user/.dovecot.sieve

When Spamassassin marks emails as Spam it adds X-Spam-Flag: YES to the headers. This rule checks the headers and sends mail to the spam folder if that flag exists.

## Testing: GTUBE SPAM email

Here's how to send an email to your server using Telnet that will **definitely** be marked as spam. There's a neat trigger called [GTUBE](http://spamassassin.apache.org/gtube/) (Generic Trigger for Unsolicited Bulk Email) that is implemented in spamassassin.

All we need to do is send a message that contains this line in the body:

XJS\*C4JDBQADN1.NSBN3\*2IDNEN\*GTUBE-STANDARD-ANTI-UBE-TEST-EMAIL\*C.34X

We can use Telnet to send this email like we did when testing in parts 1 and 2:

feathers-mcgraw@Hobbs-T440s:~$ telnet yourdomain.com 25

Trying 192.168.1.174...

Connected to yourdomain.com.

Escape character is '^]'.

220 yourdomain.com ESMTP Postfix (Debian/GNU)

ehlo randomdomain.com

250-yourdomain.com

250-PIPELINING

250-SIZE 10240000

250-VRFY

250-ETRN

250-STARTTLS

250-ENHANCEDSTATUSCODES

250-8BITMIME

250 DSN

mail from: test@randomdomain.com

250 2.1.0 Ok

rcpt to: pi

250 2.1.5 Ok

data

354 End data with .

Subject: test spam email

This should set it off...

XJS\*C4JDBQADN1.NSBN3\*2IDNEN\*GTUBE-STANDARD-ANTI-UBE-TEST-EMAIL\*C.34X

.

250 2.0.0 Ok: queued as DDFDEDA77

quit

221 2.0.0 Bye

Connection closed by foreign host.

That email should land in your Spam folder. Note that if you're using Squirrelmail you won't be automatically subscribed to the Spam folder, you have to add it yourself.

Thanks to [Jens](https://samhobbs.co.uk/comment/2984#comment-2984) for finding this test!

## Optional: Managesieve

Managesieve is a service that will allow you to log in remotely with a compatible email client and manage your sieve scripts.

First, we need to install the necessary package:

sudo apt-get update

sudo apt-get install dovecot-managesieved

You will notice that this creates a new configuration file: /etc/dovecot/conf.d/20-managesieve.conf, which you can leave as it is.

One more configuration change to make: open /etc/dovecot/dovecot.conf and add sieve to the protocols line:

protocols = imap lmtp sieve

and restart Dovecot:

sudo service dovecot restart

The managesieve service uses port 4190, so log in to your router's admin page and forward this port to your Pi.

You can now test if the service is running with telnet:

sam@samhobbs:/etc/dovecot$ telnet localhost 4190

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

"IMPLEMENTATION" "Dovecot (Ubuntu) Pigeonhole"

"SIEVE" "fileinto reject envelope encoded-character vacation subaddress comparator-i;ascii-numeric relational regex imap4flags copy include variables body enotify environment mailbox date ihave"

"NOTIFY" "mailto"

"SASL" "PLAIN LOGIN"

"STARTTLS"

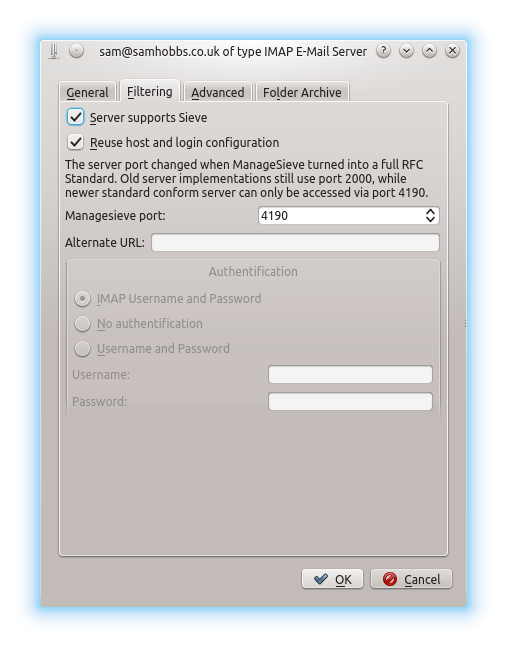
"VERSION" "1.0"

OK "Dovecot (Ubuntu) ready."

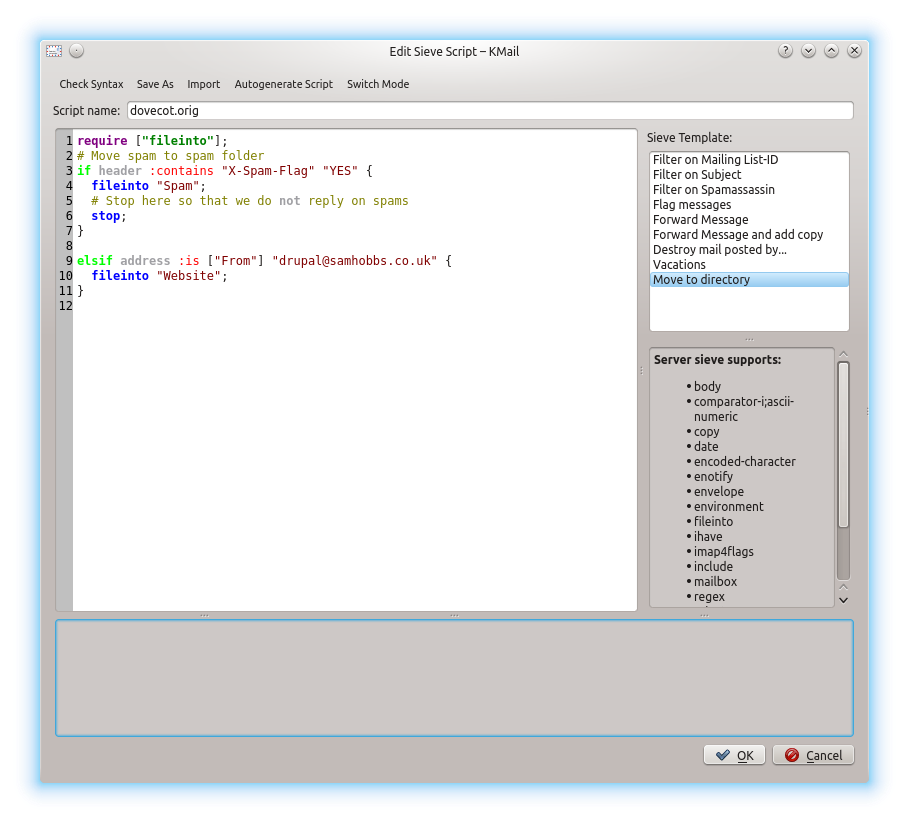
If it's up and running, then you can try connecting with a compatible email client, like **Kmail**, the KDE email client that ships with distributions like[Kubuntu](http://www.kubuntu.org/). There is also a [free software sieve plugin](https://addons.mozilla.org/en-US/thunderbird/addon/sieve/#id=2548) for Thunderbird that you might like to try if that's your client of choice.

To enable sieve in Kmail, go to:  
Settings --> configure Kmail --> accounts --> Modify --> Filtering

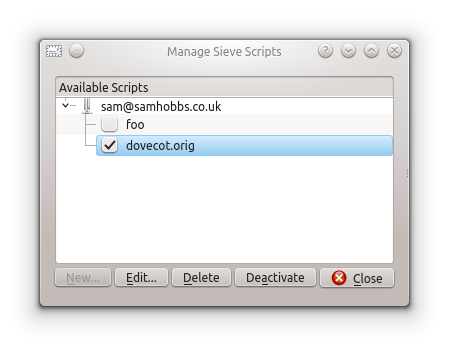
Check "Server supports Sieve" and "Reuse host and login configuration".



Now you can select Settings --> "Manage Sieve Scripts" and edit your scripts using Kmail's editor, complete with syntax highlighting.



If you write multiple scripts, they will be stored in ~/sieve and the active one will be symlinked from ~/.dovecot.sieve. If you had a script at ~/.dovecot.sieve before you set up managesieve, it will be saved as dovecot.orig when you create a new one.



Thanks to the members of [Kubuntuforums](https://kubuntuforums.net/), without whom I would never have even heard of sieve rules!

Please leave a comment, I'd love to hear how you're getting on!

**Tags:**

[Raspberry Pi](https://samhobbs.co.uk/tags/raspberry-pi)

[email](https://samhobbs.co.uk/tags/email)

[server](https://samhobbs.co.uk/tags/server)

[LMTP](https://samhobbs.co.uk/tags/lmtp)

[sieve](https://samhobbs.co.uk/tags/sieve)

[dovecot](https://samhobbs.co.uk/tags/dovecot)

[postfix](https://samhobbs.co.uk/tags/postfix)

**Category:**

[Email Server](https://samhobbs.co.uk/taxonomy/term/3)

[Raspberry Pi](https://samhobbs.co.uk/taxonomy/term/1)

**Type:**

[Tutorial](https://samhobbs.co.uk/taxonomy/term/9)