Contents

1 The Pre-Installation Guide
   1.1 Architecture of Mailman Suite .................................................. 3
   1.2 Pre-Installation Guide ................................................................. 5

2 The Installation Guide
   2.1 Django for Mailman Admins ....................................................... 9
   2.2 Installation Instructions ............................................................ 11
   2.3 Upgrading Mailman 3 ................................................................. 25
   2.4 Frequenty Asked Questions (Admin) ............................................ 28

3 Configuring Mailman 3
   3.1 Configuring Mailman Core ....................................................... 31
   3.2 Configure Web Frontend ............................................................ 36

4 Migrating to Mailman 3 from Mailman 2.1
   4.1 Migrating from Mailman 2.1 to Mailman 3 ..................................... 41
   4.2 FAQ (2.1 to 3 Migration) ........................................................... 43

5 The User Guide
   5.1 GNU Mailman 3.1 - List Member Manual ...................................... 45

6 The Contributor Guide
   6.1 Internationalization(i18n) .......................................................... 51
   6.2 The Contributor Guide ............................................................... 52
   6.3 Contributing to Mailman Documentation ...................................... 58

Index ......................................................................................................... 65
The Mailman home page is http://www.list.org, and there is a community driven wiki at http://wiki.list.org.

The Mailman Suite consists of 5 individual projects. Below are links to documentation for each of the projects, their issue trackers and latest releases.

Those packages are copyrighted by the Free Software Foundation and distributed under the terms of the GNU General Public License (GPL) version 3 or later.

Note: There are inconsistencies throughout this documentation in examples which invoke the Django management commands. The actual invocation depends on how Django and Mailman 3 are installed. It may be

- mailman-web ...
- python manage.py ...
- django-admin ...

or something else. When reading these docs, you may need to translate one of the above into what works in your installation.
What do I need to know before trying to install Mailman3?

1.1 Architecture of Mailman Suite

Mailman Suite is a set of two functional components, Mailman Core and Web Frontend. Web frontend is based on Django Web Framework and is often just referred to as Django. The actual projects that use the Django web framework are called Postorius (Web Frontend) and Hyperkitty (Archiver). From a runtime perspective, Postorius and Hyperkitty can run in a single component.
- **MTA**: Mail-Transfer-Agent is typically responsible for receiving emails from internet and sending out emails. It is configured to forward emails to Mailman owned mailing lists to Mailman Core’s LMTP server via LMTP protocol.

  MTA also receives incoming requests to send out emails from both Mailman Core and Web Interface. Web interface will send out emails related to account management like verifying email addresses for example.

  Hyperkitty also allows replying to posts from the web interface, which are sent out as emails through the MTA, which then in-turn are forwarded to Mailman Core like any other incoming email.

- **Web Server**: Web server is typically just a frontend to Web Interface due to security and performance. It is configured as a reverse-proxy for the Web Interface.

  Web Frontend will support any web server that can act as a reverse proxy.

- **Mailman Core**: Mailman Core is a complex application and the above diagram only represents some parts of it. It listens primarily on two ports:

  - **REST API**: This is where Mailman’s administrative REST API listens. This is used by the frontend to retrieve and show information to the user.

  - **LMTP Server**: This is how Mailman receives incoming emails from MTA.

- **Web Interface**: This includes both Hyperkitty and Postorius.
1.2 Pre-Installation Guide

1.2.1 What is the current state of Mailman 3?

Mailman 3 is a complete re-write of Mailman 2.1 and has been split into several modular components which collectively we call Mailman Suite. We also often just call it Mailman 3. Let us look very briefly into these components:

- **Mailman Core** or just Core is the main engine that is responsible for handling emails. It does all the task of managing users, subscriptions, mailing lists, email addresses, talking to the MTA etc. However, the things that it doesn’t manage include user authorization & user authentication.

  However, users can manage their settings using email based commands which includes subscribing, unsubscribing, changing a few basic settings for their accounts etc.

  Most functionalities in Core can be accessed using an administrative REST API which provides full control over the Core. Core expects the clients that consume this API would take up the responsibility of authenticating and authorizing users and provide interfaces for them to manage their settings and subscriptions.

- **Mailman Client** is just sometimes called as Client and is the official Python bindings to the Core’s REST API.

- **Postorius** is the official web front-end over the Core’s REST API and is built on Django web framework. In Django’s terminology, Postorius is an “app” which can be be plugged into any Django installation or “project”. Postorius has been built to be deployed alongside other Django “apps” which makes it a little bit difficult to deploy since you need to have a “project” of your own if you just need it to work!

  **See also:**

  What is Django?

  If you don’t have any familiarity with Django, we do provide an example Django “project” with pre-populated settings which should run Postorius out of the box. However, because of the several different ways in which you can configure Django, it might not suite everybody’s needs, in which case you might have to dive into Django’s (and other helper library’s) settings related documentation to figure out the best settings for you.

- **Hyperkitty** is the official archiver and, similar to Postorius, is a Django “app”. It is also built along the same principles of re-usability and can be deployed alongside Postorius with the provided Django “project”.

1.2.2 How Can I upgrade from Mailman 2.1.x?

Mailman 2.1 series is the current most popular series of Mailman. The short version is that as of now, upgrading from Mailman 2.1 to Mailman 3.1 is fairly straightforward.

Now the long version. Because of the changes in Database Schema, migrating from Mailman 2.1 to Mailman 3.1 can have issues. There is a mailman import21 command that works well to import list settings and membership. The issues involve held moderator requests and pending digests and they can be handled with some scripting.

Archives can be imported into Hyperkitty easily, but URLs to attachments are going to break because the URL paths are different in Hyperkitty. Although, you might be able to retain your HTML archives and/or archived attachments from Mailman 2.1, at least if they are public, so the old URLs still work.
1.2.3 What do I need to know before deploying Mailman 3?

The installation guide presumes some knowledge of general Python based web applications and their ecosystem. This requirement however is only required if you follow the installation guide provided here. When the distro packages for Mailman 3 are out, you probably won’t need any of this information. If you are new to Python and have no idea about what pip, django and wsgi means, this section is for you!

Here are the three most basic terminologies that you will often encounter when trying to deploy Mailman 3:

- **pip** is the Python’s official package manager and can be used to download and install libraries and packages from PyPI a.k.a Python CheeseShop. On most Linux based distributions, you can install it from the distro’s package manager. Here are the instructions for apt & yum (or dnf) based systems:

  $ sudo apt install python3-pip

  and:

  $ sudo yum install python3-pip

  If you have any other Linux distro, please check its documentation on how to install. After that, you can use it to install python packages using the command like below:

  $ sudo pip3 install <packagename>

  The above command is equivalent to:

  $ sudo python3 -m pip install <packagename>

- **Django** is the Python web framework that Postorius & Hyperkitty are based on. It allows you to run several different web “apps” under a single “project”. People often write their web applications as reusable Django “apps” which can then be plugged into any running Django “project”. A Django “project” is also sometimes called a Django “installation”.

  A typical Django project has a structure of something like this:

  toplevel_project:
  
  |--- urls.py
  |--- settings.py
  |--- manage.py
  |--- wsgi.py

  It is important to understand the above structure even when using Django, because the configuration of a Django project requires you to edit these files.

  Here is a brief overview of these files:

  - urls.py: This is how Django routes requests. If you want to install a new Django “app”, you have to add its base_url to this file.

  - settings.py: This is Django’s configuration file. It is the home for all the different configuration options that are required.

  - manage.py: This is a generic helper script used to perform administrative tasks from the command line on a Django project. You should never edit this file after creating a project.

  - wsgi.py: This is the WSGI or Web Server Gateway Interface application for the Django project. You will need this file later to interface Django with an actual webserver like Nginx or Apache. Usually, you don’t need to change anything in this file.
• **wsgi** or the Web Server Gateway Interface is the protocol by which Python web applications talk to web servers. Django comes with a built-in web server, which you can invoke by using the following command:

```
$ python manage.py runserver
```

This will start a development server on http://localhost:8000/. You can also specify a different host:port to bind to. See `python manage.py runserver --help` for more instructions.

To deploy any Python based web application, you need an intermediate WSGI server which mediates the interaction between Python and a web server. There are several of them out there but we recommend using uwsgi. Uwsgi has several advantages over others including the ability to configure it entirely using environment variables, which helps in container based deployments. Also, Nginx and Apache web servers have plugins built-in for Uwsgi which makes it an even more compelling candidate.

However, you are not tied to using Uwsgi and are free to choose any other WSGI server.

• When running Django using the built-in development web server, Django serves its static files which makes it easy for the developers. However, in production environment, it is advised to serve the static files separately.

To collect all the static files for all the Django projects in one single place:

```
$ python manage.py collectstatic
```

This will collect all the static files in the location mentioned in Django’s settings as STATIC_ROOT, which is usually under static directory in Django’s project’s root path.

You need to serve this directly with your web server using a proxy or alias rule, depending on your webserver. Here is the relevant portion of the configuration for Nginx:

```yaml
server {
    ...
    location /static/ {
        alias /path/to/django/STATIC_ROOT;
    }
}
```

You can do this in Apache using a configuration that looks something like this:

```xml
<VirtualHost *:443>
    ...
    Alias /static /opt/mailman/web/static
    Alias /favicon.ico /opt/mailman/web/static/hyperkitty/img/favicon.ico
    ProxyPassMatch ^/static/ !
    ...
</VirtualHost>
```
2.1 Django for Mailman Admins

This is a brief primer on Django to help with administration of Mailman as it is used as a web framework within the web interface of Mailman 3.

Ideally, Django should just be a python dependency and the details should be wrapped behind configurations and settings in Mailman. However, due to the amount of functionality provided to us by Django, it would be a monumental effort to hide it completely. We often resort to referring Django documentation directly in our documentation and also rely on several deployment related features that Django provides.

There are three main topics that is important for system administrators that we are going to cover in this doc, configuration, management commands and running (deployment).

2.1.1 Configuration

Django expects its configuration to be provided via a settings module. The name of the module itself isn’t a constant and can be passed in as --settings command line parameter in Django commands and also be set using DJANGO_SETTINGS_MODULE environment variable.

The un-usual thing to note here is that it expects a module not a file path, so whatever the name of the module is, it should be importable by Django(in Python), which can be achieved by adding the directory the file settings.py exists in the PYTHONPATH environment variable. It works similar to the PATH environment variable that unix shell use.

Since it is using Python, you can use more Pythonisms in how the configuration is defined or loaded. For example, a common pattern is to keep settings.py as close to the one provided by the upstream project, but make the changes required in a settings_local.py and then import the configuration in settings.py.

```python
# settings.py

<SNIP>
```

(continues on next page)
try:
    from settings_local import *
except ImportError:
    # If the import fails, don't worry.
    pass

# settings_local.py
DATABASES = ...
SECRET_KEY = ...

This way, you can keep the “local” configuration overrides in a separate file. You can also take this a step further to put secrets into a secrets.py and import from there in settings.py if you want to separate secrets from other configuration.

**Django upstream settings**

Django’s upstream settings documentation is probably the most useful place to lookup the variables you might find in the Mailman provided settings.py but don’t fully understand how to configure.

All the settings are applicable directly, including things like LOGGING (to setup logging), DATABASES(to setup backend database configuration), USE_I18N(Internationalization of the interface), EMAIL_BACKEND (how to talk to local or remote MTA for sending emails).

**Mailman-web configuration**

On top of these base settings provided by Django, the configurations for Postorius, Hyperkitty and django-mailman3 all belong to the same settings.py (or settings_local.py, if you choose to use that).

There are other third party Django applications, which we reuse for various functionality in Mailman-web use similar configuration process via settings.py.

The most important packages for Mailman are:

- **django-allauth**: Good place to find any configuration related to account management in the Web UI from email sending issues to contents of the email sent for account/email verification.

- **Django-haystack**: This is used for search indexing of the emails in Hyperkitty. This is a good place to find various settings for setting up different search backends.

- **Django-qcluster**: This will provide all the asynchronous jobs that are run via cron in Hyperkitty to update various internals in a routine fashion. Use this to see how different backends can be configured to store and run jobs and how to tweak the configuration of each backend.

**2.1.2 Management Commands**

This is probably the most useful (and confusing!) this about Django due to the sheer number of ways that this can be done. Django management commands provide a framework to write and run commands for Django web applications for various manual or other tasks.

Django’s [documentation on management commands](https://docs.djangoproject.com/en/3.3/topics_COMMANDS/) is a good place to start, but if it is too long for you, this is a short know how for Mailman admins.
There are three variants for running management commands, all of which can be use interchangeably, depending on how you installed. Package managers, Docker images and mailman-web all provide different ways to run the same commands.

- **python3 manage.py --settings settings --pythonpath <parent dir of settings.py> <command>:** This command is probably the _most_ common one that you can see. You will notice that it accepts --settings and --pythonpath flags to provide the settings module of Django. You can also set the environment variables DJANGO_SETTINGS_MODULE and PYTHONPATH instead of passing the values to command.

- **django-admin --settings settings --pythonpath <parent dir of settings.py> <command>:** This is exactly same as the above, replace the “manage.py” file with an installed command django-admin, obviating the need to find the manage.py file in your current installation. The same environment variables mentioned in the above marker works here as well.

- **mailman-web <command>:** This is the simplified version of the above two commands, which comes with the mailman-web package. It defaults the settings.py to be present at /etc/mailman3/settings.py and automatically sets up the PYTHONPATH for Django to be able to import. You can also set MAILMAN_WEB_CONFIG environment variable to point to a different config file (not module!) and it will take care of adding it to the PYTHONPATH and setting DJANGO_SETTINGS_MODULE.

The documentation of Mailman is slowly going to lean towards using mailman-web command as it is simple, but third party packages and Django can emit errors that use any of the above three notations. Which is why it is good to know them. You can use `mailman-web <command>` for commands from Django (like, migrate or createsuperuser for example) or from third party packages (like qcluster or qinfo).

### 2.1.3 Running Django

Running Django based web applications follows a Python standard mechanism called WSGI, which describes how a web server communicates with Python applications. There is also ASGI for async based applications, but Mailman currently doesn’t use it.

You can use any WSGI server to run mailman-web. Our current documentation recommends using uwsgi, but gunicorn is also a great option. Some web servers like Apache2 come with wsgi modules, which obviate the need for a separate WSGI server like uwsgi, gunicorn. So, if your current infrastructure is already using Apache2, this might be an option for you.

It is good to refer to uwsgi docs or gunicorn docs if you want to configure the aspects of running Django that aren’t already done in Django, which includes things like, access/error logging, number of threads and processes to run which can handle your expected load, SSL configuration (if you want SSL all the way through, instead of terminating at web server) etc.

Most (all?) WSGI servers will require only the path to the (1) Django’s settings and (2) WSGI “module”. This wsgi module, is often times just a wsgi.py file somewhere on PYTHONPATH. If you are using mailman-web then it comes pre-installed and you can refer to it as mailman_web.wsgi, without worrying about the file itself.

Since web servers are more hardened in terms of security and more capable of handling high load, _typically_, WSGI servers listen on a local addresses and Web server proxy requests from Internet to the Wsgi server.

### 2.2 Installation Instructions

#### 2.2.1 Mailman 3 in Docker Containers

Abhilash Raj maintains container images for Mailman 3 which you can use directly without having to go through all the steps to download dependencies and configuring Mailman. You can find the detailed documentation on how
to use them. If you have any issues using the container images, please report them directly on the Github repo for docker-mailman.

### 2.2.2 Installing Mailman from Linux distro packages

Other distribution specific packages for Mailman 3 are not yet available. If you would like to help package Mailman 3 for your favorite Linux distro, please get in touch at mailman-developers@python.org.

#### Installing on Debian

Debian provides Mailman 3 packages. The meta-package `mailman3-full` depends on all components of a complete Mailman 3 suite. If you want to split the installation of the Mailman 3 core mailinglist delivery daemon and the Mailman 3 Django web suite with Postorius and Hyperkitty, take a look at the packages `mailman3` and `mailman3-web`.

The Debian Mailman 3 packages can be found here:

https://packages.debian.org/search?keywords=mailman3

#### 2.2.3 Virtualenv Installation

This is a step-by-step installation guide for Mailman Suite, also sometimes referred as Mailman 3 Suite or just Mailman 3. There are more than one ways to have Mailman 3 running on your machine.

The commands in this guide are tailored for Ubuntu/Debian systems.

**Dependencies**

**Python3.7+**. While Mailman supports any version of Python > 3.6, versions 3.7+ is recommended.

**MTA Setup** Mailman 3 in theory would support any MTA or mail server which can send emails to Mailman over LMTP. Officially, there are configurations for Postfix, Exim4, qmail and sendmail. Mailman Core has a fairly elaborate documentation on setting up your MTA. Look below at Installing Mailman Core to find out the location of configuration file `mailman.cfg` which is mentioned in the documentation above.

The Web Front-end is based on a Python web framework called Django. For email verification and sending out error logs, Django also must be configured to send out emails.

This guide uses setup instructions for Postfix.

**Sass compiler** A sass compiler. Syntactically Awesome Stylesheets or Sass is an extension of CSS, which is used to style webpages, that adds more power to the old CSS syntax. It allows using variables, nested rules, inline imports etc, which CSS originally doesn’t support. Hyperkitty uses this to generate CSS styles.

You can use the C/C++ implementation. Please look at the installation guide for sass to see how you can get one.

**Python development packages** Python3 dev package. This is required for building Postorius.

**Fulltext search** A full text search engine like Whoosh or Xapian. Whoosh is the easiest to setup and can be installed in the virtualenv.

**See also:**

Virtualenv setup on how to setup the virtualenv.
**Lynx**  An HTML to plaintext converter like lynx is required by Mailman Core if you have configured it to convert emails to plaintext.

## Installing Dependencies

To install Mailman Core, you need to following system packages:

```
$ sudo apt install python3-dev python3-venv sassc lynx
```

## Setup database

This guide is based off on Postgresql database engine. Mailman also supports MySQL/MariaDB in case you have that already running for other applications. The steps shown for Postgresql are generic SQL. If you are using a different database, just substitute its command line utility for `psql`. A few known differences between Postgresql and other database engines are noted below.

```
$ sudo apt install postgresql libpq-dev
$ sudo -u postgres psql
psql (12.5 (Ubuntu 12.5-0ubuntu0.20.04.1))
Type "help" for help.
postgres=# create database mailman;
CREATE DATABASE
postgres=# create database mailmanweb;
CREATE DATABASE
postgres=# create user mailman with encrypted password 'MYPASSWORD';
CREATE ROLE
postgres=# grant all privileges on database mailman to mailman;
GRANT
postgres=# grant all privileges on database mailmanweb to mailman;
GRANT
postgres=# \q
```

**Note:** Replace ‘MYPASSWORD’ with a secret password.

We created two databases named, `mailman` and `mailmanweb` for Mailman Core and Mailman Web (Django) respectively. We also created a new user `mailman` and granted it privileges to both the databases.

## Setup Mailman user

Create a new user to run all the Mailman services:

```
$ sudo useradd -m -d /opt/mailman -s /usr/bin/bash mailman
$ sudo chown mailman:mailman /opt/mailman
$ sudo chmod 755 /opt/mailman
$ sudo su mailman
```

## Virtualenv setup

Virtualenv is Python’s mechanism to create isolated runtime environments.
Hint: If you are not familiar with virtualenv, checkout the user guide for virtualenv.

Note: Make sure that you are running commands as mailman user from here forth. Setup Mailman user.

Create the virtualenv for Mailman:

```bash
$ cd /opt/mailman
$ python3 -m venv venv
```

Activate virtualenv:

Activate the created virtualenv:

```bash
$ source /opt/mailman/venv/bin/activate
```

Note: The rest of this documentation assumes that virtualenv is activated. Whether or not virtualenv is activated can be seen by a (venv) before the shell prompt.

You can setup mailman user’s shell to automatically activate the virtualenv when you switch to the user by running:

```bash
$ echo 'source /opt/mailman/venv/bin/activate' >> /opt/mailman/.bashrc
```

Installing Mailman Core

Mailman Core is responsible for sending and receiving emails. It exposes a REST API that different clients can use to interact with over an HTTP protocol. The API itself is an administrative API and it is recommended that you don’t expose it to outside of your host or trusted network. To install Core run:

```bash
(venv)$ pip install wheel mailman psycopg2-binary<2.9
```

This will install latest release of Mailman Core, and Python bindings for Postgresql database. Currently Django>=2.2,<3.1 requires psycopg2-binary<2.9. See this issue. Later versions of Django will work with psycopg2-binary 2.9.x. After this, create a configuration file at `/etc/mailman3/mailman.cfg` for Mailman Core:

```ini
# /etc/mailman3/mailman.cfg
[paths.here]
var_dir: /opt/mailman/mm/var

[mailman]
layout: here
# This address is the "site owner" address. Certain messages which must be
delivered to a human, but which can't be delivered to a list owner (e.g. a
# bounce from a list owner), will be sent to this address. It should point to
# a human.
site_owner: user@example.com

[database]
class: mailman.database.postgresql.PostgreSQLDatabase
url: postgresql://mailman:MYPASSWORD@localhost/mailman
```

(continues on next page)
[archiver.prototype]
enable: yes

# For the HyperKitty archiver.
[archiver.hyperkitty]
class: mailman_hyperkitty.Archiver
enable: yes
configuration: /etc/mailman3/mailman-hyperkitty.cfg
# And, create the /etc/mailman3/mailman-hyperkitty.cfg file containing
# these settings uncommented
#[general]
#base_url: http://127.0.0.1:8000/archives/
#api_key: Secret_Hyperkitty_API_Key

[shell]
history_file: $var_dir/history.py

[mta]
verp_confirmation: yes
verp_personalized_deliveries: yes
verp_delivery_interval: 1

See also:
The further configuration setup for Mailman Core Configuring Mailman Core.

Setup MTA

A Mail Transfer Agent (MTA) is responsible for sending and receiving Emails on the server. This guide is based off on Postfix MTA, but Mailman also supports other MTAs like Exim4 etc:

$ sudo apt install postfix

Choose “Internet Site” when prompted during installation for choosing Postfix configuration.
Enter the domain name you have chosen for Mailman in next step.

To configure Postfix to relay emails to and from Mailman add the following to Postfix’s configuration at /etc/
postfix/main.cf:

```sh
unknown_local_recipientRejectCode = 550
owner_request_special = no

transportMaps =
    hash:/opt/mailman/mm/var/data/postfix_lsmtp
local_recipientMaps =
    hash:/opt/mailman/mm/var/data/postfix_lsmtp
relayDomains =
    hash:/opt/mailman/mm/var/data/postfix_domains
```

Note that in the above, if your current main.cf contains settings for these items, including the default setting for local_recipient_maps, you should add the Mailman setting to the existing setting rather than replacing it. For example:

```sh
local_recipient_maps = proxy:unix:passwd.byname $aliasMaps
    hash:/path-to-mailman/var/data/postfix_lsmtp
```

See also:

See detailed documentation to setup Postfix with Core and some unusual configuration if you already have Postfix running.

Starting Mailman automatically

To start Mailman Core automatically on boot, you can setup a systemd service. Create a new file `/etc/systemd/system/mailman3.service`:

```sh
[Unit]
Description=GNU Mailing List Manager
After=syslog.target network.target postgresql.service

[Service]
Type=forking
PIDFile=/opt/mailman/mm/var/master.pid
User=mailman
Group=mailman
ExecStart=/opt/mailman/venv/bin/mailman start
ExecReload=/opt/mailman/venv/bin/mailman restart
ExecStop=/opt/mailman/venv/bin/mailman stop

[Install]
WantedBy=multi-user.target
```

If you are not using PostgreSQL as your RDBMS, you should substitute the appropriate service in the After= line.

In cases of abnormal shutdown, the RDBMS may need to replay transactions etc., which will take some time. It is reported that MariaDB (and probably MySQL) reports successful initialization before the recovery process is done, so Mailman fails to start because it can’t connect to the database. You may want to add the lines:

```sh
Restart=on-failure
RestartSec=15
```

to the [Service] section of mailman.service. (PostgreSQL probably does not suffer from this problem because it uses sd_notify(3) to indicate successful initialization rather than the exec of the daemon process.)

You can load this configuration by running:
Depending on where the mailman.service file is installed you may need to run systemctl enable mailman.service before daemon-reload. To start the systemd service and Mailman Core:

```bash
$ sudo systemctl start mailman3
# Verify that the service is running.
$ sudo systemctl status mailman3
```

After this, running mailman info (as mailman user with virtualenv active) should give you an output which looks something like below:

```
(venv)$ mailman info
GNU Mailman 3.3.2 (Tom Sawyer)
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0]
config file: /etc/mailman3/mailman.cfg
db url: postgresql://mailman:MYPASSWORD@localhost/mailman
devmode: DISABLED
REST root url: http://localhost:8001/3.1/
REST credentials: restadmin:restpass
```

**Warning:** Pay attention to the config file output above and make sure that you see /etc/mailman3/mailman.cfg there. If the config path is different, you should make sure to create a new config file at /etc/mailman3/mailman.cfg and re-run the command.

### Setup Cron Jobs

Mailman Core requires some cron jobs for periodic actions. To setup cron jobs for Mailman You can run:

```
sudo -u mailman crontab -e
```

Add the following in the editor:

```
@daily /opt/mailman/venv/bin/mailman digests --periodic
@daily /opt/mailman/venv/bin/mailman notify
```

This will run the commands at midnight every day, you can configure running them at specific time by replacing @daily with 0 8 * * * for running at 08:00 instead.

### Installing Web UI

Postorius and Hyperkitty are Mailman's official Web UI and Archiver. Mailman-web provides a convenient single package to install both of these.

To install the web components, run the following commands with virtualenv activated:

```
(venv) $ pip install mailman-web mailman-hyperkitty
```
Initial Configuration

Then, create a new configuration file at `/etc/mailman3/settings.py`. A sample configuration looks something like this:

```python
# Mailman Web configuration file.
# /etc/mailman3/settings.py

# Get the default settings.
from mailman_web.settings.base import *
from mailman_web.settings.mailman import *

# Settings below supplement or override the defaults.

#: Default list of admins who receive the emails from error logging.
ADMINS = ('Mailman Suite Admin', 'root@localhost'),

# Postgresql database setup.
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'mailmanweb',
        'USER': 'mailman',
        'PASSWORD': '$MYPASSWORD',
        'HOST': 'localhost',
        'PORT': '5432',
    },
}

# 'collectstatic' command will copy all the static files here.
# Alias this location from your webserver to `/static`
STATIC_ROOT = '/opt/mailman/web/static'

# enable the 'compress' command.
COMPRESS_ENABLED = True

# Make sure that this directory is created or Django will fail on start.
LOGGING['handlers']['file']['filename'] = '/opt/mailman/web/logs/mailmanweb.log'

#: See https://docs.djangoproject.com/en/dev/ref/settings/#allowed-hosts
ALLOWED_HOSTS = [
    "localhost", # Archiving API from Mailman, keep it.
    "127.0.0.1", # "lists.your-domain.org",
    # Add here all production domains you have.
]

#: See https://docs.djangoproject.com/en/dev/ref/settings/#csrf-trusted-origins
#: For Django <4.0 these are of the form 'lists.example.com' or
#: '.example.com' to include subdomains and for Django >=4.0 they include
#: the scheme as in 'https://lists.example.com' or 'https://*.example.com'.
CSRF_TRUSTED_ORIGINS = [
    "lists.your-domain.org",
    # Add here all production domains you have.
]
```

(continues on next page)
In the above configuration, make sure to

- Change the database password in DATABASES
- Set one or more admins in ADMINS for all the users who will receive emails about errors and exceptions in Django.
- Add any domains/IP addresses you want to serve Mailman web from into ALLOWED_HOSTS config.
- Make sure that the logging path is created. You can run `mkdir -p /opt/mailman/web/logs` to create the path.
- Also make sure the STATIC_ROOT directory is created.
- Set the SECRET_KEY to a random value.

See also:
All the default settings in mailman-web can be found here.

Run database migrations

To setup the database schema for Mailman’s web components, run:

```
(venv)$ mailman-web migrate
```

Collect static files

To copy all the static files (css, js, images) into the STATIC_ROOT from Initial Configuration run:

```
(venv)$ mailman-web collectstatic
```

Compress CSS files

To compress the various CSS files offline run:
Compile messages for I10n

To update the message catalogs for supported languages run:

```
(venv)$ mailman-web compilemessages
```

Setting up a WSGI server

See also:

*What is WSGI?*

These instructions are to setup your Django website behind a webserver. We are using uwsgi as the wsgi server to communicate between the webserver and Django, however *Setting up Gunicorn* is also a good choice. To install uwsgi, run:

```
(venv)$ pip install uwsgi
```

**Note:** The configuration below doesn’t serve static files, so if you are just “trying-it-out” and want static files to be served, you need to add some additional configuration and steps. See *serving static files with uwsgi*.

See also:

*Why does my django site look ugly?*

See also:

django uwsgi docs

Create a configuration file for uwsgi at `/etc/mailman3/uwsgi.ini`:

```
# /etc/mailman3/uwsgi.ini
#
[uwsgi]
# Port on which uwsgi will be listening.
hp-socket = 0.0.0.0:8000
# If running uwsgi from the virtual environment ...
virtualenv = /opt/mailman/venv/

module=mailman_web.wsgi:application
# Set PYTHONPATH
env = PYTHONPATH=/etc/mailman3/
# The default settings module.
env = DJANGO_SETTINGS_MODULE=settings

# Setup default number of processes and threads per process.
master = true
processes = 2
threads = 2

# Setup the django_q related worker processes.
attach-daemon = /opt/mailman/venv/bin/mailman-web qcluster
```

(continues on next page)
# Setup the request log.
req-logger = file:/opt/mailman/web/logs/uwsgi.log

# Log qcluster commands separately.
logger = qcluster file:/opt/mailman/web/logs/uwsgi-qcluster.log
log-route = qcluster uwsgi-daemons

# Last log and it logs the rest of the stuff.
logger = file:/opt/mailman/web/logs/uwsgi-error.log

You can run test run uwsgi using the following command:

(venv)$ uwsgi --ini /etc/mailman3/uwsgi.ini

Have a look at uwsgi documentation to learn more about different configuration options.

## Setting up Gunicorn

To use Gunicorn instead of uWSGI it isn’t necessary to install Gunicorn as it is already installed as a dependency of Mailman core. Instead of the above, create a configuration file for Gunicorn at /etc/mailman3/gunicorn.conf:

```
# /etc/mailman3/gunicorn.conf
#
bind = ['127.0.0.1:8000']
proc_name = "mailman-web"
chdir = "/opt/mailman/mm"
pidfile = "/opt/mailman/mm/var/gunicorn.pid"
accesslog = "/opt/mailman/mm/var/logs/access.log"
errorlog = "/opt/mailman/mm/var/logs/error.log"
```

See Gunicorn docs for more on configuring Gunicorn.

## Automatically starting Mailman-web

In order to automatically start at startup, you can create a second systemd service, mailmanweb.service. Create a new file /etc/systemd/system/mailmanweb.service:

```
[Unit]
Description=GNU Mailman Web UI
After=syslog.target network.target postgresql.service mailman3.service

[Service]
Environment="PYTHONPATH=/etc/mailman3/
User=mailman
Group=mailman
ExecStart=/opt/mailman/venv/bin/uwsgi --ini /etc/mailman3/uwsgi.ini
KillSignal=SIGINT

[Install]
WantedBy=multi-user.target
```

If using Gunicorn, the ExecStart line above becomes:
After creating this file, you can run following commands to start:

```bash
$ sudo systemctl daemon-reload
$ sudo systemctl start mailmanweb
```

Then, you check the status by running:

```bash
$ systemctl status mailmanweb
```

Also, if using Gunicorn, you need an additional service to start Django’s qcluster. You don’t need this with uWSGI because uWSGI is configured to do this itself. But, for Gunicorn, create another systemd service, qcluster.

Create the file `/etc/systemd/system/qcluster.service`:

```ini
[Unit]
Description=HyperKitty async tasks runner
After=syslog.target network.target postgresql.service mailman3.service

[Service]
ExecStart=/opt/mailman/venv/bin/mailman-web qcluster --pythonpath /etc/mailman3 --
--settings settings
User=mailman
Restart=always

[Install]
WantedBy=multi-user.target
```

After creating this file, you can run following commands to start qcluster:

```bash
$ sudo systemctl daemon-reload
$ sudo systemctl start qcluster
```

Then, you check the status by running:

```bash
$ systemctl status qcluster
```

Cron Jobs for Mailman Web

Mailman Web requires some cron jobs for periodic actions. To setup cron jobs for Mailman You can run:

```bash
sudo -u mailman crontab -e
```

Add the following in the editor:

```bash
* * * * * /opt/mailman/venv/bin/mailman-web runjobs minutely
0,15,30,45 * * * * /opt/mailman/venv/bin/mailman-web runjobs quarter_hourly
@hourly /opt/mailman/venv/bin/mailman-web runjobs hourly
@daily /opt/mailman/venv/bin/mailman-web runjobs daily
@weekly /opt/mailman/venv/bin/mailman-web runjobs weekly
@monthly /opt/mailman/venv/bin/mailman-web runjobs monthly
@yearly /opt/mailman/venv/bin/mailman-web runjobs yearly
```
Nginx Configuration

You can reverse proxy the requests to a uwsgi or gunicorn server using Nginx. Uwsgi has a special protocol called uwsgi protocol that is available in Nginx via the `ngx_http_uwsgi_module` module. That and the `uwsgi_params` file are included in a typical Nginx install, but the config below doesn’t use them and works just as well with gunicorn. Add or edit the configuration file at `/etc/nginx/sites-available/default`:

```nginx
server {
    listen 443 ssl default_server;
    listen [::]:443 ssl default_server;

    server_name MY_SERVER_NAME;
    location /static/ {
        alias /opt/mailman/web/static/;
    }

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $remote_addr;
    }

    ssl_certificate /path-to-ssl-certs/cert.pem;
    ssl_certificate_key /path-to-ssl-certs/privkey.pem;
}
```

For a more complete configuration, you can check out Nginx documentation for setting up SSL. Replace `MY_SERVER_NAME` with the appropriate domain name you intend to serve.

After editing the configuration file, you can restart Nginx to pickup the changes:

```sh
$ sudo systemctl restart nginx
```

After this, you should have Mailman running at https://MY_SERVER_NAME.

Apache Configuration

You can also reverse proxy the requests to a uwsgi or gunicorn server using Apache. At a minimum, you need to add the following to your existing Apache configuration at `/etc/apache2/sites-available/`:

```apache
<Directory "/opt/mailman/web/static" 
    Require all granted
</Directory>

<IfModule mod_headers.c>
    RequestHeader unset X-Forwarded-Proto
    <If "%{HTTPS} =~ /on/"> 
        RequestHeader set X-Forwarded-Proto "https"
    </If>
</IfModule>

<IfModule mod_proxy.c>
    ProxyPreserveHost On
</IfModule>
```

(continues on next page)
ProxyPass "/mailman3" "http://127.0.0.1:8000/mailman3"
ProxyPass "/archives" "http://127.0.0.1:8000/archives"
ProxyPass "/accounts" "http://127.0.0.1:8000/accounts"
ProxyPass "/admin" "http://127.0.0.1:8000/admin"
ProxyPass "/user-profile" "http://127.0.0.1:8000/user-profile"
</IfModule>

See Apache docs for more on configuring Apache.

Depending on your setup and experience administering a machine, you can choose one of the following installations methods:

- Mailman 3 in Docker Containers
- Installing Mailman from distro packages:
  - Installing on Debian
- Virtualenv Installation, recommended for Production

### 2.3 Upgrading Mailman 3

This includes the requirements to upgrade to Mailman 3 suite, which includes:

- Mailman Core 3.x
- Postorius 1.x
- MailmanClient 3.x
- Django-mailman3 1.x
- Hyperkitty 1.x

If you are interested in the full change log for each component, please refer to documentation of each project.

#### 2.3.1 Requirements

**Python 3**

Postorius, Django-mailman3 and Hyperkitty have been ported to Python 3 now. Generally the last few versions of Python and Django are supported but the supported versions keep on changing over time. Release announcements should have information about which versions of Python are supported.

#### 2.3.2 Migration Steps

**Distro packages**

If you are using distro packages, they should handle the upgrade process themselves. You may need to perform some changes after the upgrade is done, please check with the distro package’s documentation for those instructions.
Docker images

If you are using container images please refer to their documentation or release notes on steps to upgrade a new version.

Virtualenv Install

If you are following virtualenv installation guide from here, these are the steps to upgrade to a newer version of Mailman Core.

Pre-upgrade

First, you need to stop the running Mailman services before the upgrade to make sure that you don’t end up in a bad state. You can stop the two systemd services, for Core and Mailman-Web:

```bash
$ sudo systemctl stop mailman3
$ sudo systemctl stop mailmanweb
```

Then you need to switch to mailman user, activate the virtualenv:

```bash
$ sudo su mailman
$ source /opt/mailman/venv/bin/activate
```

Upgrade package

Finally, upgrade your packages:

```
(venv) $ pip install -U mailman postorius django-mailman3 hyperkitty mailman-web
```

**Warning:** It is important to upgrade all the packages together since they depend on each other for critical functionality. There are no compatibility guarantees with a new version of one package and an old version of another package.

We often update the minimum version required for a dependency in Packages’ metadata but that doesn’t always result in new versions being installed. There is also no direct dependency between django packages and mailman package, but they still expect the latest version of Mailman core.

Post upgrade

Post upgrade, you need to do a bunch of other tasks. First would be to make sure that your database schema is updated to handle the new versions:

```
(venv) $ mailman-web migrate
```

Then, make sure that all the static files, like stylesheets, javascript, images etc are copied to a single place. This is required to have the CSS and Javascript displayed correctly:

```
(venv) $ mailman-web compress
(venv) $ mailman-web collectstatic
```
Update your i18n translation caches to make sure that your installation can pickup any new changes to strings and their translations:

```
(venv) $ mailman-web compilemessages
```

Finally, you can re-start the services to bring up the Web Interface and Mailman Core:

```
$ sudo systemctl start mailman3
$ sudo systemctl start mailmanweb
```

### 2.3.3 Upgrade to 3.2.0

These are specific instructions to upgrade to Mailman Core 3.2.0 or 1.2.0 for mailman-web components (like, Posto- rius, Hyperkitty) from an older version.

**Configuration**

Postorius added a new *required* configuration flag `POSTORIUS_TEMPLATE_BASE_URL` which should be set to URL that Postorius is expected to be listening at. You should set it to whatever URL your WSGI server is listening at.

**Extra Migration Steps**

- Full text index for Hyperkitty needs to be re-built since our indexing engine doesn’t maintain compatibility between index created in Python 2 and Python 3.
  
  Simplest way to do this to run `python manage.py rebuild_index` in your Django project. Note that if your project has a huge number of lists, this will take a lot of time.

  A new command `python manage.py update_index_one_list <listname@example.com>` was added to Hyperkitty so that you can rebuild your index one-by-one, if you prefer that.

### 2.3.4 Upgrade from <3.3.1 to >=3.3.1

Mailman core 3.3.1 is the first release that actually processes bounces. Prior to this release, bounces were recorded in the `bounceevent` table in the database, but not processed. This leads to a situation where upgrading to core >=3.3.1 can cause many old, stale bounces to be processed resulting in disabling delivery to some list members who are not currently bouncing.

There are two ways to avoid this when upgrading. The preferred way is to enable VERP probes. Instead of immediately suspending delivery when a member’s bounce score reaches threshold, a probe message is sent to the member and delivery is suspended only if the probe bounces. This is done by putting:

```
[mta]
verp_probes: yes
```

in `mailman.cfg` before upgrading.

The other way to avoid this is to set the `processed` flag on all the rows of the `bounceevent` table before upgrading with a query like:

```
UPDATE bounceevent SET processed = 1;
```
2.4 Frequently Asked Questions (Admin)

Frequently asked questions by Mailman 3 administrators.

- 1. Confirmation emails to Users has wrong domain name (example.com!)
- 2. The domain name displayed in Hyperkitty shows example.com or something else.
- 3. How to enable debug logging in Mailman Core?
- 4. How to print logs to standard out instead of regular files in Mailman Core?
- 5. What is Mailman-web?
- 6. How to disable signup?
- 7. How to change language in Mailman 3?

2.4.1 1. Confirmation emails to Users has wrong domain name (example.com!)

This happens when your reverse (SSL) proxy isn’t setting up the correct headers when proxying requests. Fix this by setting the right `proxy_set_header` directives:

```sh
# For Nginx.
location / {
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_set_header X-Forwarded-Host $host;
    proxy_pass http://localhost:8000/;
}
```

Appropriate headers for different web servers needs to be set if something other than Nginx is being used.

2.4.2 2. The domain name displayed in Hyperkitty shows example.com or something else.

The name of the domain comes from the `SITE_ID` which should be set to the domain you want to display. If you have Postorius >= 1.3.3 The Domains view will show you the `SITE_ID` for the domain and has an Edit link to the Django admin UI entry for that site.

Otherwise, you can go to the Django admin Sites view (`/admin` or `/<prefix>/admin`) and see the DOMAIN NAME and DISPLAY NAME for each defined site, and you can edit the domain name.

Note that setting `SITE_ID = 0` in Django’s settings will cause HyperKitty to display the DISPLAY NAME for the domain whose DOMAIN NAME matches the accessing domain. However, do not set `SITE_ID = 0` in a new installation without any existing Sites as this will cause an issue in applying migrations. Only set `SITE_ID = 0` after there are domains defined in the Django admin Sites view.

2.4.3 3. How to enable debug logging in Mailman Core?

There are a couple of loggers in Mailman Core which can be individually configured with log levels. Available loggers are:

- archiver – All archiver output
• bounce – All bounce processing logs go here
• config – Configuration issues
• database – Database logging (SQLAlchemy and Alembic)
• debug – Only used for development
• error – All exceptions go to this log
• fromusenet – Information related to the Usenet to Mailman gateway
• http – Internal wsgi-based web interface
• locks – Lock state changes
• mischief – Various types of hostile activity
• plugins – Plugin logs
• runner – Runner process start/stops
• smtp – SMTP activity
• subscribe – Information about leaves/joins
• vette – Message vetting information

You can set their log levels by adding the following to your mailman.cfg:

```
[logging.http]
level: DEBUG
```

### 2.4.4 4. How to print logs to standard out instead of regular files in Mailman Core?

For each of the available loggers (listed in FAQ #3), you can set the path for each logger separately in mailman.cfg:

```
[logging.root]
path: /dev/stdout
```

### 2.4.5 5. What is Mailman-web?

Mailman web is basically a set of configuration files for deploying Mailman’s web frontend packaged in a single Python package. It provides a command line, mailman-web which can be used to run administrative commands.

If you are familiar with Django, or have seen older versions of this documentation, mailman-web command is equivalent of running `django-admin --pythonpath /etc/mailman3/ --settings settings`. It just makes it simpler to run administrative commands and simplify user documentation.

See documentation for Mailman-web.

### 2.4.6 6. How to disable signup?

To disable signup/registration for new users, while continuing to allow signed up users to login, you can add the following to your settings.py:

```
# /etc/mailman3/settings.py
ACCOUNT_ADAPTER = 'django_mailman3.views.user_adapter.DisableSignupAdapter'
```
To disable only social account signups, but keep the signups open, you can add the following to your `settings.py`:

```python
# /etc/mailman3/settings.py
SOCIALACCOUNT_ADAPTER = 'django_mailman3.views.user_adapter.DisableSocialSignupAdapter'
```

You would need `django-mailman3 >= 1.3.6` for these settings to work.

### 2.4.7 7. How to change language in Mailman 3?

TODO
Mailman 3 can be configured in a wide variety of ways. After you have installed Mailman 3, you can now proceed to configure it for production use.

### 3.1 Configuring Mailman Core

If you are here, it means that you have successfully installed complete Mailman 3 or some of its components. To verify that you have Mailman Core installed try:

```bash
$ mailman info
GNU Mailman 3.1.0 (Between The Wheels)
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170118]
config file: /etc/mailman.cfg
db url: sqlite:///var/lib/mailman/data/mailman.db
devmode: DISABLED
REST root url: http://localhost:8001/3.1/
REST credentials: restadmin:restpass
```

There are a few important parameters in the above command that can help you debug problems if they occur. Most important of which is the configuration file that is being used, pointed to by `config_file` that you see above.

Mailman 3 uses a total of two main configuration files, the first one is for Mailman Core and the other is Django’s `settings.py` used to deploy the web front end.

`mailman.cfg` is the main configuration file for Mailman Core. Core will look for this file at several different places in order like:

- Filesystem Path pointed to by `MAILMAN_CONFIG_FILE` environment variable.
- `mailman.cfg` in the current working directory.
- `var/etc/mailman.cfg` relative to the current working directory.
- `$HOME/.mailman.cfg`
• /etc/mailman.cfg
• ../../etc/mailman.cfg relative to the working directory of `argv[0]` i.e. the working directory of the `mailman` command script.

Alternatively, you can specify the absolute path of the configuration file using the `-C` flag in the command line.

### 3.1.1 Configuring Filesystem Paths

Mailman by default puts all the configuration, logs and lock files in the current working directory under a `var` directory. Each `var` directory belongs to either a running or previously run Mailman Core instance.

While it is often confusing to several people, the reason it is default is because it helps a lot with debugging and testing. In production you want a static location for your configuration, locks and logs.

All the different paths that Mailman creates are, by default, configured relative to `var_dir`. For example, `paths.fhs` which is based on Filesystem Hierarchy Standard. To use it add the following to your configuration:

```ini
[mailman]
layout: fhs
```

This is equivalent to the following configuration:

```ini
bin_dir: /sbin
var_dir: /var/lib/mailman
queue_dir: /var/spool/mailman
log_dir: /var/log/mailman
lock_dir: /var/lock/mailman
etc_dir: /etc
ext_dir: /etc/mailman.d
pid_file: /var/run/mailman/master.pid
```

Most of the path names are self explanatory, like all the runnable scripts are put under `/sbin` and all the logs are placed under `/var/log/mailman`.

If you want a custom directory layout you can define a new layout and use that instead. For example:

```ini
[paths.custom]
var_dir: /home/user/.mailman

[mailman]
layout: custom
```

If you are installing Mailman from source (or using Pip), it is recommended to use `paths.local`. To use it, add the following to your `mailman.cfg`:

```ini
[mailman]
layout: local
```

It is equivalent to the following configuration:

```ini
var_dir: /var/tmp/mailman
bin_dir: $argv
log_dir: $var_dir/logs
lock_dir: $var_dir/locks
data_dir: $var_dir/data
cache_dir: $var_dir/cache
etc_dir: $var_dir/etc
```

(continues on next page)
3.1.2 Configuring MTA

The first step to get Core to work is to enable it to talk to the Mail Transport Agent or MTA. It supports various different MTAs.

**Postfix** Core automatically generated transport maps (`postfix_lmtp` and `postfix_lsmtp`) to be used by Postfix at `var/data/`. To configure Postfix, add the following configuration to `main.cf`:

```
# Support the default VERP delimiter.
recipient_delimiter = +
unknown_local_recipient_reject_code = 550
owner_request_special = no
transport_maps =
    hash:/path-to-mailman/var/data/postfix_lmtp
local_recipient_maps =
    hash:/path-to-mailman/var/data/postfix_lsmtp
relay_domains =
    hash:/path-to-mailman/var/data/postfix_domains
```

Mailman’s `var` directory can vary according your source of installation. Please refer to the documentation provided with your source or ask Mailman Developers at mailman-developers@python.org.

Note that in the above, if your current `main.cf` contains settings for these items, including the default setting for `local_recipient_maps`, you should add the Mailman setting to the existing setting rather than replacing it. For example:

```
local_recipient_maps = proxy:unix:passwd.byname $alias_maps
    hash:/path-to-mailman/var/data/postfix_lsmtp
```

To configure Core to use Postfix, add the following configuration to your `mailman.cfg` configuration, be sure to replace `mail.example.com` with your email domain:

```
[mta]
incoming: mailman.mta.postfix.LMTP
outgoing: mailman.mta.deliver.deliver
lsmtp_host: mail.example.com
lsmtp_port: 8024
smtp_host: mail.example.com
smtp_port: 25
```

**Exim4** To setup Exim4, add the following files to your configuration as file names mentioned above:

```
# /etc/exim4/conf.d/main/25_mm3_macros
# The colon-separated list of domains served by Mailman.
domainlist mm_domains=list.example.net
MM3_LMTP_PORT=8024
# MM3_HOME must be set to Mailman's var directory, wherever it is
```

(continues on next page)
According to your installation. If you are following these instructions on Debian, it might be MM3_HOME=/opt/mailman/mm/var, for example.

```
MM3_HOME=/path-to-mailman/var
```

The configuration below is boilerplate:

- The path to the list receipt (used as the required file when matching list addresses)

```
MM3_LISTCHK=MM3_HOME/lists/${local_part}.${domain}
```

```
# /etc/exim4/conf.d/router/455_mm3_router
mailman3_router:
  driver = accept
  domains = +mm_domains
  require_files = MM3_LISTCHK
  local_part_suffix_optional
  local_part_suffix = 
  -bounces : -bounces++ : 
  -confirm : -confirm++ : 
  -join : -leave : 
  -owner : -request : 
  -subscribe : -unsubscribe
  transport = mailman3_transport

# /etc/exim4/conf.d/transport/55_mm3_transport
mailman3_transport:
  driver = smtp
  protocol = lmtp
  allow_localhost
  hosts = localhost
  port = MM3_LMTP_PORT
  rcpt_include_affixes = true
```

You should note that Exim4 configuration above doesn’t support multiple domains like Postfix does. Please change the variables in the configuration above before installing it.

To configure Mailman to use Exim4 add the following to your `mailman.cfg`

```
[mta]
incoming: mailman.mta.exim4.LMTP
outgoing: mailman.mta.deliver.deliver
lmtp_host: mail.example.com
smtp_host: mail.example.com
lmtp_port: 8024
smtp_port: 25
```

Other MTAs Mailman also supports Sendmail and qmail. Please check the Mailman Core documentation for sendmail setup and qmail setup to configure them.

### 3.1.3 Configuring REST API

Core presents a HTTP Rest API which clients can use to interact with it. Note that this is an administrative API and must not be exposed to the public internet. It has a very basic HTTP Basic Authentication which can be configured using the configuration below:
If you need to bind to a different host or port, you can change the configuration above according to your needs.

There are several different other ways to configure Core and you can find more details about it in the Core’s documentation.

### 3.1.4 Configure Templates

You can configure templates for headers, footers, and automated emails generated by Mailman. By default, the only templates available are in English, but you can add them other languages too. You can add templates to the `var` directory of Mailman, which will be picked up by Core.

- sitewide templates can be put in `$var/templates/site/LC/`,
- domain specific templates in `var/templates/domains/DOMAIN/LC/`
- list specific templates in `var/templates/lists/LIST-ID/LC/`

List overrides domain overrides site overrides defaults.

### 3.1.5 Configuring Databases

Core supports Sqlite, PostgreSQL and MySQL and each of them are tested. By default, without any configuration, Core uses Sqlite database. For a production use, a more efficient database like PostgreSQL or MySQL is highly recommended. Please have a look at the setting up your database in Core’s documentation for the same.

### 3.1.6 Configuring Cron Jobs

Depending on your configuration, Core has some periodic tasks that need to be run using job schedulers like `cron`. Right now, the two tasks required to be run routinely for Core are periodic sending of digests for mailing lists that have `digest_send_periodic` set to `true` and periodic sending of notices of pending requests to list moderators. These can be customized to send at any periodic time, the following format sends out digests at midnight:

```
@daily /path/to/mailman digests --periodic
```

This format will send moderator notices at 08:00:

```
0 8 * * * /path/to/mailman notify
```

If your installation has lists which make use of the NNTP Gateway to gate messages from Usenet to one or more lists, you also need to periodically run the `mailman gatenews` command to gate messages from Usenet to the list(s). This format will run the command every 5 minutes:

```
*/5 * * * * /path/to/mailman gatenews
```

Note: Be careful about mailman.cfg locations.

3.1. Configuring Mailman Core
If your mailman.cfg is not found in a standard location, mailman will silently write a new empty mailman.cfg in ~/.var/etc and terminate successfully. If your notify and digests jobs are failing silently, try adding the -C option to specify the config file to use.

### 3.2 Configure Web Frontend

Mailman 3 has a web frontend which can be used to administer the Core, manage subscriptions and view the Archives. There are two principal components, Postorius, the Mailman’s web frontend, and Hyperkitty, the Mailman’s official Archiver.

Both Postorius and Hyperkitty are built atop Django, a Python based web framework.

See also:

*What is Django?*

Django is generally configured using a python module called settings.py which is usually present at the root of the Django project. It doesn’t have to be at the root of the project, but anywhere importable by python.

Assuming that you have already cloned or installed the Django project and know their location, we can now start configuring Mailman Components. If you haven’t have a look at Setting-up-Django-Project.

See also:

All the default settings in mailman-web can be found here. Some of the following configuration is already present in mailman-web.

#### 3.2.1 Setting up Admin Account

To create a superuser account in Django, run the following interactive command as mailman user:

```
(venv)$ mailman-web createsuperuser
```

Before you can login with this user, you have to configure Django to send emails, so that it can verify the email provided for the superuser account above.

In Postorius, a superuser is assumed to be the Site Owner and has access to all the domains, mailing lists and their settings. List Owners and Moderators can be added based on per-list basis and don’t need to have a superuser account in Django.

#### 3.2.2 Scheduled Tasks (required)

There are some routine tasks that need to be run alongside Django, most of which are meant to do some specific routine functions in Hyperkitty. You can add the following to your crontab run them using other cron-like interfaces:

```
# This goes in /etc/cron.d/mailman

@hourly mailman /opt/mailman/venv/bin/mailman-web runjobs hourly
@daily mailman /opt/mailman/venv/bin/mailman-web runjobs daily
@weekly mailman /opt/mailman/venv/bin/mailman-web runjobs weekly
@monthly mailman /opt/mailman/venv/bin/mailman-web runjobs monthly
@yearly mailman /opt/mailman/venv/bin/mailman-web runjobs yearly
* * * * * mailman /opt/mailman/venv/bin/mailman-web runjobs minutely
2,17,32,47 * * * * mailman /opt/mailman/venv/bin/mailman-web runjobs quarter_hourly
```
To Check what jobs do exist and will run on scheduled time, you can run:

```
(venv) $ mailman-web runjobs -l
Job List: 11 jobs
  appname - jobname - when - help
  --------------------------
  django_extensions - cache_cleanup - daily - Cache (db) cleanup Job
  django_extensions - daily_cleanup - daily - Django Daily Cleanup Job
  hyperkitty - empty_threads - monthly - Remove empty threads
  hyperkitty - new_lists_from_mailman - hourly - Import new lists from Mailman
  hyperkitty - orphan_emails - daily - Reattach orphan emails
  hyperkitty - recent_threads_cache - daily - Refresh the recent threads
  hyperkitty - sync_mailman - daily - Sync user and list properties with Mailman
  hyperkitty - thread_order_depth - yearly - Compute thread order and depth for all threads
  hyperkitty - thread_starting_email - hourly - Find the starting email when it is missing
  hyperkitty - update_and_clean_index - monthly - Update the full-text index and clean old entries
  hyperkitty - update_index - hourly - Update the full-text index
```

### 3.2.3 Setting up Email

It is important that Django be able to send emails to verify the addresses that are subscribing to the Mailman. This configuration is separate from what is done in Core. Please have a look at how to set up email backend for Django. A simple configuration would look something like this for a mail server listening on localhost:

```
# To be added to Django's settings.py

EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'localhost'
EMAIL_PORT = 25
EMAIL_HOST_USER = <username>
EMAIL_HOST_PASSWORD = <password>
```

**Note:** Mailman Suite project in Gitlab disables sending of emails when `DEBUG=True` is set in `settings.py`, and instead prints the emails to a directory `emails` under `mailman-suite_project`. If you don’t see any outgoing emails, please check your `settings.py` and set `DEBUG=False`.

Read more about `DEBUG`.

Here are some settings that determine how your emails from them look like:

- **DEFAULT_FROM_EMAIL:** This is the default address that is used as the FROM header in all the emails from your django installation.
- **SERVER_EMAIL:** This is the address from which the errors emails will be sent to you.

**Note:** In order to make django send mails, you need to change `EMAIL_BACKEND` from `django.core.mail.backends.console.EmailBackend` to `django.core.mail.backends.smtp.EmailBackend` in the `mailman-suite` project, if you are using that instead of Mailman-web.
Note that both of these are general Django related settings and will affect other parts of your Django installation too. You can check if your Email is setup correctly by running:

```
(venv)$ mailman-web sendtestemail youremailid@domain
```

This should send you test email.

### 3.2.4 Running the task queue

Hyperkitty uses `django_q` as a task queue. It supports various different backends like Redis, Disque, IronMQ, SQS etc. Please check the documentation to better understand how to configure it. The most basic setup where it uses Django orm as the queue can be configured using the settings below:

```
Q_CLUSTER = {
    'retry': 360,
    'timeout': 300,
    'save_limit': 100,
    'orm': 'default',
}
```

You will also have to run `python manage.py qcluster` command along with Hyperkitty. See Hyperkitty’s docs about asynchronous tasks.

### 3.2.5 Enable full text search (required)

Hyperkitty uses `django-haystack` for its full text search. There are several full text engines that can be used. See `django-haystack` documentation for more details about the different backend engines that it supports. For the most basic setup, you can use `whoosh` backend. To install the library try:

```
(venv)$ pip install whoosh
```

Then add the following configuration to the Django’s settings.py to enable whoosh engine for full text search:

```
HAYSTACK_CONNECTIONS = {
    'default': {
        'ENGINE': 'haystack.backends.whoosh_backend.WhooshEngine',
        'PATH': os.path.join(BASE_DIR, "fulltext_index"),
        # You can also use the Xapian engine, it's faster and more accurate, 
        # but requires another library.
        # http://django-haystack.readthedocs.io/en/v2.4.1/installing_search_engines.html
        #xapian
        # Example configuration for Xapian:
        # 'ENGINE': 'xapian_backend.XapianEngine'
    },
}
```

### 3.2.6 Configure Login to Django

Postorius & Hyperkitty both use `django-allauth` for authentication because it supports a wire variety of social providers and also allows users to sign up with their email if they desire.

Note that if you have any problems with the account signup/signin related emails, you should look the documentation for `django-allauth`. 
Some of the very basic settings that are required to be set for Postorius & Hyperkitty to work are mentioned below:

- ACCOUNT_AUTHENTICATION_METHOD = "username_email"
- ACCOUNT_EMAIL_REQUIRED = True
- ACCOUNT_EMAIL_VERIFICATION = "mandatory"
- ACCOUNT_DEFAULT_HTTP_PROTOCOL = "http"
- ACCOUNT_UNIQUE_EMAIL = True

### 3.2.7 Configure Social Login

Yes, so the way social login works in Mailman is by using a library called django-allauth. There are a few social providers already “enabled” in the Django configuration for the container images, to add more you would have change INSTALLED_APPS in your settings_local.py. There is no way to “add” any apps as the one defined in settings_local.py will override the original one (they are just python variables), so you’d have to copy the entire INSTALLED_APPS3 and then add whatever new ones you want.

To see a list of all the providers, please have a look at the documentation of django-allauth. Make sure that the one you choose provides “email” as part of user data, otherwise it won’t work with Mailman. e.g. Twitter doesn’t give out emails.

Now to be able to use any provider, you’d have to configure them in your site. django-allauth documentation2 does provide instructions and direct urls to configure each one. To summarize the documentation here are the steps:

Go to your Django admin interface (located at ‘/admin’, like http://example.com/admin) and login with whatever superuser you created. Scroll down to the section “SOCIAL ACCOUNTS” and enter the one saying “Social applications”. The relative url for that is /admin/socialaccount/socialapp/ Click on “ADD SOCIAL APPLICATION” button on the top left of the page. Fill out the details from your social provider and choose the “Sites” you want to use for that particular social provider. You can have separate credentials for separate “Sites”.

That should be all. Make sure when you request client_id and client_secret from the social providers, you provide the correct callback_url. The documentation for django-allauth has the correct url for each provider, but it looks basically like

http://mysite.com/accounts/<provider>/login/callback

and you replace <provider> with amazon, facebook, google or whatever provider.

### 3.2.8 Configure Postorius & Hyperkitty

See also:

Here are the parameters that will affect Postorius and Hyperkitty will function. These parameters are configured in your Django's settings.py.

- MAILMAN_REST_API_URL Complete URL to the Core’s REST API Server. Usually, Mailman Core listens on port 8001 for REST API. e.g. http://localhost:8001/
- MAILMAN_REST_API_USER Username for the Core’s REST API, default value in core is ‘restadmin’ if not set.
- MAILMAN_REST_API_PASS Password for Mailman REST API User, default value in core is ‘restpass’ if not set.
- MAILMAN_ARCHIVER_KEY The Key used to authenticate the emails for archiving in Hyperkitty. Its value should be exactly same as set in the mailman-hyperkitty.cfg file. (See Connecting to Mailman.)
Mailman Suite Documentation, Release 3.3

Also note that the value in settings.py will be within single quotes, but in mailman-hyperkitty.cfg it will be without any quotes.

**FILTER_VHOST** Filter the list of available lists in Postorius and Hyperkitty depending on the domain they are being currently served from. Mailman 3 supports multiple domains in a single installation.

**LOGIN_URL** ‘account_login’

**LOGIN_REDIRECT_URL** ‘list_index’

**LOGOUT_URL** ‘account_logout’

**STATICFILE_FINDERS** Add compressor.finders.CompressorFinder to your STATICFILES_FINDERS.

See also STATICFILE_FINDERS.

**COMPRESS_PRECOMPILERS** This setting is for django-compressor which is used here to compile and compress static files:

```python
COMPRESS_PRECOMPILERS = (
    ('text/x-scss', 'sassc -t compressed {infile} {outfile}'),
    ('text/x-sass', 'sassc -t compressed {infile} {outfile}'),
)
```

**POSTORIUS_TEMPLATE_BASE_URL** should be set to URL that Postorius is expected to be listening at. You should set it to whatever URL your WSGI server is listening at.

**TIME_FORMAT** Defines the time format which will be displayed in the Web interface wherever a time is present. See also, https://docs.djangoproject.com/en/3.1/ref/settings/#time-format

**DATETIME_FORMAT** Similar to TIME_FORMAT, but for date & time. See also, https://docs.djangoproject.com/en/3.1/ref/settings/#std:setting-DATETIME_FORMAT
Migrating to Mailman 3 from Mailman 2.1

4.1 Migrating from Mailman 2.1 to Mailman 3

This guide covers the migration steps required for porting from Mailman 2.1 to Mailman 3.

4.1.1 Why Upgrade?

Mailman 3 is the new version and actively developed, as compared to 2.1, which is now in maintenance mode and won’t receive any feature updates.

Some of the reasons that might convince you to migrate to Mailman 3 include:

- a well tested and rewritten code base,
- support for REST API (if you need to integrate with other services)
- support for multiple domains in same installation (without listname collisions)
- A real database backend for settings and configuration
- Modern Web UI for list administration and subscription management
- Support for social logins (and possibly LDAP/openid integration)
- Much improved and interactive archiver/forum with ability to reply from within the Archiver

4.1.2 Other considerations

Before you upgrade, you should consider a few things like:
• URLs to archived messages will break, unless you take extra steps to keep them around. Upgrade mechanism makes sure to import all your archived messages in the new system, but, all the URLs to the new messages are going to be different.

If you need your URLs for Mailman 2 archives to work, you can keep the HTML files generated for the archives around and your web server configuration for the archives intact (possibly with a notice to viewers that it is now a read-only archive, see this list for example).

• The above mechanism won’t work for private archives since the archives are gated with password and without a Mailman 2 list, there is no password. You can however import them to Mailman 3.

• Some configuration and settings aren’t available in Mailman 3’s UI yet, so even though those settings will be migrated to Mailman 3, you may not be able to change them from the Web UI today. All of those settings should be exposed in the UI very soon.

4.1.3 Before you upgrade

• Make sure that you have no pending subscription requests as those will not be ported over to Mailman3.

• Make sure that you don’t have any pending emails in the digest mbox, if there are, you can force send the digests before moving to Mailman 3, as those won’t be upgraded to Mailman3.

4.1.4 Upgrade strategy

As of now, there isn’t a turn key solution to migrate all your lists from Mailman 2 to Mailman 3, although, the process is fairly automated. You need shell access to your installation in order to perform the upgrade.

Mailman 3 is split in two main parts, the Core engine which includes all the lists and their configuration and the Web UI, which includes the archiver and UI for information in Core.

Before you start with migration, you need a working Mailman 3 instance, you can see here for recommendations on installing Mailman.

Some key information that you should know about your Mailman 2 and Mailman 3 installations before you do the migration:

• Location of list configuration in Mailman 2: This is typically at $var_prefix/lists/LISTNAME/config.pck where $var_prefix is an installation dependent directory which is typically /usr/local/mailman or /var/lib/mailman.

• Location of list archives in Mailman 2: This is typically at $var_prefix/archives/private/LISTNAME.mbox/LISTNAME.mbox where $var_prefix is as above.

• Location of the bin/ commands in Mailman 2: This is at $prefix/bin where $prefix is an installation dependent directory which is typically /usr/local/mailman or /usr/lib/mailman

Steps for migration:

• Create the list you are trying to migrate in Mailman 3, for the purposes of this guide, we will call it foo-list@example.com

• Migrate the list configuration from Mailman 2 to Mailman 3 by running the following command\(^1\)

\[
$ mailman import21 foo-list@example.com /path/to/mailman2/foo-list/config.pck
\]

\(^1\) This, and in general all, mailman commands should be run as the Mailman user and not as root. Running some of these commands as root can create files owned by root that can’t be read by the Mailman user.
• Migrate the list archives from Mailman 2 to Mailman 3 by running the following command\(^2\):

\[
\$ \text{python manage.py hyperkitty_import} -l \text{foo-list@example.com} \text{ $var\_prefix/archives/} \\
\text{→private/foo-list.mbox/foo-list.mbox}
\]

After this, you will need to rebuild the index for this list:

\[
\$ \text{python manage.py update_index_one_list} \text{ foo-list@example.com}
\]

After this, you should be able to search your messages in Hyperkitty.

- Delete Mailman 2 list:

\[
\$ \text{$prefix/bin/rmlist} \text{ foo-list}
\]

After this, you may need some additional steps based on if you want to keep your old archives around or not. To add a notice to your list archives, you can edit index.html available at the root of your mailing list archives.

You may also want to add a redirect at old list page to automatically redirect your users to Mailman3 list-info page. However, this process is fairly manual depending on type of webserver you are using.

### 4.2 FAQ (2.1 to 3 Migration)

Frequently asked questions about migration from Mailman 2.1 to 3 and related to their feature sets post migration.

- **1. List password and Moderator passwords are not working**

#### 4.2.1 1. List password and Moderator passwords are not working

Mailman 3 has a completely new account management system, none of the passwords from Mailman 2 will work with Mailman 3. Each user needs to sign up in Postorius for a new account to manage their memberships on all lists on a single instance.

Administrators and moderators also can sign up with their email accounts to manage their lists without having to share a common moderator or admin password like Mailman 2.1.

---

\(^2\) If the Mailman 2 list does not predate Mailman 2.1, its LISTNAME.mbox file is probably in good shape, but all mailboxes should be checked for defects before importing. Certain defects such as missing Message-ID: headers or missing or unparsable Date: headers will be corrected or ignored by the import process. The one defect that will definitely cause problems is lines beginning with From in message bodies. These will be seen as the start of a new message. There is a Mailman 2 script at $prefix/bin/cleanarch. That can identify and fix most such lines, but it is not perfect. Cases have been observed where a post includes in its body a copy of some other message including the From separator. This will normally occur only on an old list which includes spam messages or other email problems in its subject matter, but is something to be aware of. Certain other message defects can cause the import to abort. There is a check_hk_import script in hyperkitty/contrib that can find and report messages with these defects.
5.1 GNU Mailman 3.1 - List Member Manual

Mailman is free software for managing electronic mail discussion and e-newsletter lists. Mailman is integrated with the web, making it easy for users to manage their accounts and for list owners to administer their lists. Mailman supports built-in archiving, automatic bounce processing, content filtering, digest delivery, spam filters, and more.

This guide contains instructions for members of Mailman mailing lists so they can learn to use the features available to them. This focuses on the web interface and includes sections on joining and leaving lists, editing options and other subscriber-level tasks.

This guide is written for Mailman 3.1. If you are using Mailman 2.1 (our popular previous stable release), you should see the Mailman 2.1 Members Manual.

5.1.1 Introduction to Mailman Suite

It may be easier to think of Mailman 3 as a single piece of software that does mailing list management, but it’s actually a set of interconnected pieces of software under the hood.

The big pieces you care about as a user are as follows:

- **Mailman Core** - This is the “core” of Mailman that handles getting and sending email and stores all your email-related preferences.

- **Postorius** - This is the web interface to Mailman, which allows users to subscribe and unsubscribe from mailing lists and set preferences from the web.

- **Hyperkitty** - This is the archiver for Mailman, which allows users to view and interact with list archives from the web.

These were divided up so that you could replace pieces of Mailman as needed. For example, if a site already had a user settings page, you might want to run a modified version of Postorius so that users can set all their preferences in one place, or you might want to replace the web interface entirely.
This document is going to assume you're using these pieces together in their default states. If your Mailman installation is customized or has replaced any of these pieces, you'll need to adjust the instructions accordingly.

**Mailing List Terminology**

- A "**post**" typically denotes a message sent to a mailing list.
- People who are part of an electronic mailing list are usually called the list's "**members**" or "**subscribers**.”
- "**List administrators**” are the people in charge of maintaining that one list. Lists may have one or more administrators.
- A list may also have people in charge of reading posts and deciding if they should be sent on to all subscribers. These people are called "**list moderators**.”
- Often more than one electronic mailing list will be run using the same piece of software. The person who maintains the software which runs the lists is called the “**site administrator**.” Often the same person who acts as site administrator also administrates individual lists.

**Translating from our examples to real lists**

Often, it’s easier to simply give an example than explain exactly how to find the address for your specific list. As such, we’ll frequently give examples for a fictional list called LISTNAME@DOMAIN whose list information page can be found at http://WEBSERVER/mailman3/lists/LISTNAME.DOMAIN.

Neither of these are real addresses, but they show the form of a typical list address. The capital letters used for the list-specific parts of each address should make it easier to see what should be changed for each list. Although specific list configurations may be different, you will probably be able to just replace the words given in capital letters with the appropriate values for a real list:

- **LISTNAME** The name of your list.
- **DOMAIN** The name of the mail server which handles that list.
- **WEBSERVER** The name of the web server which handles the list web interface. This may be the same as DOMAIN, and often refers to the same machine, but does not have to be identical.

As a real-life example, if you are interested in the mailman-users list that runs on Mailman3.org, you’d make the following substitutions: LISTNAME=mailman-users, DOMAIN=mailman3.org, WEBSERVER=lists.mailman3.org. As such, for the mailman-users mailing list on mailman3.org, the list information page can be found at the URL http://lists.mailman3.org/mailman3/lists/mailman-users.mailman3.org/. (These, unlike most of the examples given in this document, are real addresses.)

Most lists will have this information stored in the List-* headers. Many mail programs will hide these by default, so you may have to choose to view full headers before you can see these informational headers.

**5.1.2 I need to talk to a human!**

If you have any trouble with any of these commands, you can always reach the person or people in charge of a list by using the list administrator email address. The list administrators can help you figure out how to do something, subscribe/unsubscribe you, or change your settings if you are unable to change them yourself for some reason. Please remember that many mailing list administrators are volunteers who are donating their spare time to run the list, and they may be very busy people.

This list administrator email address is in the form LISTNAME-owner@DOMAIN, where LISTNAME is the name of the list and DOMAIN is the name of the server. So for an example list called <wolfhounds@example.com> the administrators could be reached using <wolfhounds-owner@example.com>
5.1.3 Making a Mailman account

In order to manage your options and easily subscribe to or unsubscribe from Mailman lists, you typically want to make an account. There is a “sign up” link on most list pages (usually displayed in the upper right of the page), or you can go directly to the sign up interface at a URL that will be something like http://WEBSERVER/accounts/signup/

If you’ve been subscribed to a list without making an account (because you did this yourself or because your lists were migrated from a Mailman 2.1 setup) you can make an account using the same email address and once you’ve confirmed that you have access to that email, you’ll be able to edit all the associated options.

Note that in Mailman 3, you can actually have multiple email addresses associated to the same user account, so you don’t need to make many separate accounts to handle your permissions.

5.1.4 Subscribing and Unsubscribing

How do I join a list?

1. Go to the list information page for the list you want to join. This will be something like http://WEBSERVER/mailman3/lists/LISTNAME.DOMAIN

2. Usually, it is a good idea to make an account first using the “sign up” option (on the upper right). This account will allow you to change your settings later and make it easier for you to unsubscribe.

3. Once you are signed in, go back to the list information page and there will be a large section labelled “Subscribe to this list” where you can choose the email address you want to use and optionally choose a display name. Fill out this form and click the “subscribe” button.

4. Alternatively, you can also join a list without signing in by using the subscribe boxes at the bottom of the list information page. If you need to edit your settings later, you will need to create an account associated with the same email address.

How do I leave a list?

1. Go to the list information page for the list you want to leave. This will be something like http://WEBSERVER/mailman3/lists/LISTNAME.DOMAIN

2. Log in to confirm that you are the owner of the address that you wish to unsubscribe. If you don’t already have an account associated with that address, you may need to make one to prove that you are the correct owner of that email address.

3. Once you are logged in, there will be a section marked “Subscription / Unsubscription” that shows you the address which is subscribed to that list and a large “unsubscribe” button you can click to leave the list.

Subscribing via Primary Address

Mailman 3 includes an option to subscribe to Mailing Lists via Primary Address. This way, your subscription changes automatically when you switch your primary address to different email address, without having to update your address on every subscription.

Subscribing via primary address would also preserve your list settings associated with your subscription when you change your primary address and simply change delivery to your new primary address.
5.1.5 Changing your list settings

Mailman has a number of different settings for list subscribers as follows:

**Delivery status**  Set this option to Enabled to receive messages posted to this mailing list. Set it to Disabled if you want to stay subscribed, but don’t want mail delivered to you for a while (e.g. you’re going on vacation). If you disable mail delivery, don’t forget to re-enable it when you come back; it will not be automatically re-enabled.

**Delivery mode**  If you select summary digests, you’ll get posts bundled together (usually one per day but possibly more on busy lists), instead of singly when they’re sent. Your mail reader may or may not support MIME digests. In general MIME digests are preferred, but if you have a problem reading them, select plain text digests.

**Receive own postings**  Ordinarily, you will get a copy of every message you post to the list. If you don’t want to receive this copy, set this option to No. Note that some mail services (most prominently Gmail) will suppress this copy no matter what you do. If you need to know when your email went through and your mail provider is blocking or removing the copy, you can also use the “Acknowledge posts” option to get a separate email acknowledging your post.

**Acknowledge posts**  Receive acknowledgement mail when you send mail to the list? The options are yes or no. This is useful if your mail provider is making it difficult for you to know if your post has gone through.

**Hide address**  When someone views the list membership, your email address is normally shown (in an obscured fashion to thwart spam harvesters). If you do not want your email address to show up on this membership roster at all, select Yes for this option.

**Avoid Duplicates**  When you are listed explicitly in the To: or Cc: headers of a list message, you can opt to not receive another copy from the mailing list. Select Yes to avoid receiving copies from the mailing list; select No to receive copies.

Each of these settings can be set globally, per address, or per list. Your per-list settings over-ride the per-address settings which over-ride the global settings.

**How do I view my list settings?**

To change any settings, you can go to your settings page.

1. Log in to Mailman.
2. Click on the dropdown menu by your username (in the upper right) and select “Mailman settings”
3. Alternatively, the URL for this page will be something like http://WEBSERVER/mailman3/accounts/subscriptions/LISTNAME.DOMAIN

**How do I disable/enable my mail delivery?**

You may wish to temporarily stop getting messages from the list without having to unsubscribe. If you disable mail delivery, you will no longer receive messages, but will still be a subscriber and will retain your other settings.

To disable or enable mail delivery from the web interface:

1. Log in and go to your list settings page
2. There is an option labelled “Delivery status” which you can enable or disable on your preferences tabs, either globally, per address, or per list subscription.

This can be handy in a many different cases. For example, you could be going on vacation or need a break from the list because you’re too busy to read any extra mail. Many mailing lists also allow only subscribers to post to the list, so if you commonly send mail from more than one address (e.g., one address for at home and another for when you’re travelling), you may want to have more than one subscribed account, but have only one of them actually receive mail.
You can also use this as a way to read private archives even on a list which may be too busy for you to have sent directly to your mailbox. All you need to do is subscribe, disable mail delivery, and use your password and email to log in to the archives.

**How can I start or stop getting the list posts grouped into one big email?**

Groups of posts are called “digests” in Mailman. Rather than get messages one at a time, you can get messages grouped together. On a moderately busy list, this typically means you get one email per day, although it may be more or less frequent depending upon the list.

To change your digest settings:

1. Log in and go to your list settings page
2. There is an option labelled “Delivery Mode” which you can set on your preferences tabs, either globally, per address, or per list subscription.

There are a number of different options:

**Regular**  You get an email every time the list sends one out.

**Mime Digests**  MIME is short for Multipurpose Internet Mail Extensions. It is used to send things by email which are not necessarily simple plain text. (For example, MIME would be used if you were sending a picture of your dog to a friend.) A MIME digest has each message as an attachment inside the message, along with a summary table of contents.

**Plain Text Digests**  A plain text digest is a simpler form of digest, which should be readable even in mail readers which don’t support MIME. The messages are simply put one after the other into one large text message.

Most modern mail programs do support MIME, so you only need to choose plain text digests if you are having trouble reading the MIME ones.

**How do I stop or start getting copies of my own posts?**

By default in Mailman, you get a copy of every post you send to the list. Some people like this since it lets them know when the post has gone through and means they have a copy of their own words with the rest of a discussion, but others don’t want to bother downloading copies of their own posts.

To receive or stop receiving your own posts:

1. Log in and go to your list settings page
2. There is an option labelled “Receive own postings” which you can set to yes or no on your preferences tabs, either globally, per address, or per list subscription.

Note: This option has no effect if you are receiving digests.

Despite the availability of this option, some mail hosts (such as Gmail) will hide these posts from you. You may wish to see “How can I get Mailman to tell me when my post has been received by the list?” as an alternative solution if your posts are being eaten by your mail host.

**How can I get Mailman to tell me when my post has been received by the list?**

On most lists, you will simply receive a copy of your mail when it has gone through the list software, but if this is disabled, your list mail delivery is disabled, you use a mail host such as Gmail which blocks copies of your post from being received, or you simply want an extra acknowledgement from the system, this option may be useful to you.

To receive or stop receiving your own posts:
1. Log in and go to your list settings page

2. There is an option labelled “Acknowledge Posts” which you can set to yes or no on your preferences tabs, either globally, per address, or per list subscription.

**How can I hide my email address on the subscriber list?**

When someone views the list membership, your email address is normally shown (in an obscured fashion to thwart spam harvesters), but you can hide the address if you want:

1. Log in and go to your list settings page

2. There is an option labelled “Hide address” which you can set to yes or no on your preferences tabs, either globally, per address, or per list subscription.

Note that this does NOT hide your address in the list archives (if the list has archives) or when it’s sent out in emails, so a dedicated spammer could probably get your address in other ways.

**How can I avoid getting duplicate messages?**

Mailman can’t completely stop you from getting duplicate messages, but it can help. One common reason people get multiple copies of a mail is that the sender has used a “group reply” function to send mail to both the list and some number of individuals. If you want to avoid getting these messages, Mailman can be set to check and see if you are in the To: or CC: lines of the message. If your address appears there, then Mailman can be told not to deliver another copy to you.

To avoid duplicates:

1. Log in and go to your list settings page

2. There is an option labelled “Avoid Duplicates” which you can set to yes or no on your preferences tabs, either globally, per address, or per list subscription.

Documentation for Mailman 3 List Owners and Site Administrators is not yet complete.
6.1 Internationalization (i18n)

GNU Mailman project uses Weblate for translations. If you are interested to port Mailman to various languages, please head over to Weblate and create an account to get started with translations.

Weblate has very good documentation on how to use it:

https://docs.weblate.org/en/latest/user/translating.html

Please do not create Merge Requests for translations since it can create merge conflicts when pulling changes from Weblate and break automation which pulls and pushes changes between Gitlab and Weblate.

6.1.1 Offline translations

If you have existing translated .po files, or you would prefer to work Offline, you can download the .po files from Weblate and upload them through the web interface when you are done. This helps with keeping the merge request based workflow to ingest translations from Weblate and still allowing folks to use their preferred local editing environment.

Please see Weblate’s documentation on downloading and uploading po files.

6.1.2 Integration with Weblate

Integration with source control in Gitlab and translation project in Weblate works using webhooks and some scripts.

Weblate supports webhooks for notifications when there are changes to the source control. This allows pulling changes to source strings from Gitlab by adding a webhook notification in Gitlab.

Translations are converted to Git commits in Weblate and it is configured to send back those commits as Merge Requests, every 24 hours, to Mailman’s Gitlab projects. Each commit corresponds to a single Author and Language. We do not squash these commits so as to retain the original commits with Authorship info.
6.2 The Contributor Guide

Mailman 3 consists of a collection of separate-but-linked projects, each of which has its own development setup guide. This makes sense when you want to focus on a single piece of Mailman, but can make setting up the entire Mailman Suite in one go somewhat confusing. This guide attempts to move all the information currently in the wiki and various package documentation into a single “definitive” guide.

Main package documentation on Readthedocs.io:
- Mailman core start guide
- Mailman core “web ui in 5” guide
- Mailman core “archive in 5”
- Postorius dev guide
- Hyperkitty dev guide

6.2.1 Getting prerequisites

For the most part, setup for each project will download any needed packages. However, you will need a few system packages to be sure you’ve got the necessary version of Python and its tools, git (to get the source code), postfix (a mail server), and a few other tools that are used during setup.

On Fedora, you probably want to run:

```bash
$ sudo yum install python3-setuptools python3-virtualenv python3-devel git gcc nodejs-less postfix python3-tox
```

On Debian and Ubuntu, this may be something like:

```bash
$ sudo apt install python3-setuptools python3-virtualenv python3-dev git gcc nodejs-less postfix tox
```

On macOS, if you have Homebrew installed, you may run:

```bash
$ brew install tox python node gcc git
$ npm install --global less
$ # Note: postfix is pre-installed on macOS.
```

If you prefer, you can substitute Exim4 for Postfix. Postfix is the MTA used by most Mailman developers, but we do support Exim 4. (Sendmail support is very much desired, but the Mailman core developers need contributors with Sendmail expertise to help.) For development purposes it doesn’t matter, since we will mock all interactions to external MTA.

You will need tox to run tests.

HyperKitty also needs sassc. You can install sassc using your OS package manager

For Fedora/CentOS:

```bash
$ sudo dnf install sassc
```

Sassc is available in the newer versions of Debian(9)/Ubuntu(18.04):

```bash
$ sudo apt install sassc
```

On macOS with Homebrew, you may run:
You can also install `sassc` from source as per their build documentation.

### 6.2.2 Gitlab Setup

We use Gitlab for source code hosting and our CI. You can fork any of the projects you want and start working on it. If you don’t already have an account on Gitlab, please create one, you will need that for contributing code or participating in any other way.

We also use Gitlab for code reviews. Our workflow looks very similar to the official Gitlab Workflow. Please remember to enable shared runners on your fork, it will be used to build your code and run unit tests on pull requests that you will make. It is mandatory that you have runners enabled before you send any pull requests.

### 6.2.3 Set up a directory

Setting up the whole Mailman suite means you have to pull code from a bunch of different related repositories. You can put all the code anywhere you want, but you might want to set up a directory to keep all the pieces of mailman together. For example:

```bash
$ mkdir ~/mailman
# cd ~/mailman
```

For the rest of this development guide, we are going to assume you’re using `~/mailman` as your directory, but you can use whatever you want.

### 6.2.4 Set up virtual environments

All parts of Mailman support only Python 3.6+. For your development, it is advised that you create a virtualenv so that the packages you install don’t break any of the system packages using Python.

To create the virtualenv run the following command:

```bash
$ python3 -m venv venv3
```

To activate a virtualenv, you need to run the appropriate activate script:

```bash
$ source venv3/bin/activate
```

You must use `source` (or `.`, if your shell is a pure POSIX shell) everytime you want to activate your development environment. To make your life easier when managing virtualenvs, see virtualenvwrapper.

### 6.2.5 Set up and run Mailman Core

First, get the code:

```bash
$ cd ~/mailman
$ git clone https://gitlab.com/mailman/mailman.git
```

To set up Mailman Core, you’ll need to switch to your Python 3 virtualenv:
$ source venv3/bin/activate

Then, go into the mailman directory, run setup, and then run mailman info to be sure everything is set up correctly, and that the right settings are in place:

$ cd mailman
$ pip install -e .
$ mailman info

You can edit your mailman.cfg file to make any necessary changes. By default, during development, it is located at var//etc/mailman.cfg. Then start things up:

$ mailman start
$ cd ..

Note that mailman just makes a var/ directory wherever you start it and uses that to store your data. This is great for the purposes of testing so you can easily make fresh installs, but might be confusing if you restart your instance later from a different location and don’t have your original mailman.db file, or if you start looking around and finding var/ directories everywhere.

Later on, if you need to restart Mailman (i.e. if you get the error “Mailman REST API not available. Please start Mailman core.”) then you can also do that by calling the mailman executable from the venv as follows:

$ ~/mailman/venv3/bin/mailman start

Note that the mailman executable has several sub-commands. One that is particularly useful for debugging is mailman shell.

**Note:** If you like IPython shell (like I do!), you add the following to your mailman.cfg:

```
[shell]
use_ipython: yes

Also, remember to install ipython using pip:
```

$ pip install ipython

### 6.2.6 Testing Mailman Core

You can run tests for Mailman Core (or any Mailman project) using tox <https://tox.readthedocs.io/en/latest/>:

$ tox -e py37-nocov

This requires that you have Python3.7 installed. You change it to py36-nocov and py35-nocov to run tests with Python 3.6 and 3.5 respectively.

For contributing to Mailman it is incredibly helpful, time saving and resource-saving, running a specific test case, which only tests your contributions without testing every other component of Mailman. To run test cases from a specific file, for example the Mailman Core’s src/mailman/commands/tests/test_cli_members.py file, run a command like this:

$ tox -e py37 -- mailman.commands.tests.test_cli_members
You can omit the \texttt{--e py37} (Or any environment, not just \texttt{`py37`}) option and tox will test your files with every environment specified in \texttt{tox.ini}.

\begin{verbatim}$ tox -- mailman.commands.tests.test_cli_members\end{verbatim}

The option \texttt{--p auto} parallelizes the tests:

\begin{verbatim}
$ tox --p auto
\end{verbatim}

Of course, you can mix options to run all environments in parallel on one file:

\begin{verbatim}
$ tox --p auto -- mailman.commands.tests.test_cli_members
\end{verbatim}

### 6.2.7 Set up Mailman Client

Get the code:

\begin{verbatim}
$ cd ~/mailman
$ git clone https://gitlab.com/mailman/mailmanclient.git
\end{verbatim}

Then set up mailmanclient:

\begin{verbatim}
$ cd mailmanclient
$ pip install --e .
$ cd ..
\end{verbatim}

To run the tests:

\begin{verbatim}
$ tox --e py37
\end{verbatim}

### 6.2.8 Set up Django-mailman3

This package holds the Django libraries and templates used by Postorius and HyperKitty.

Get the code and set it up:

\begin{verbatim}
$ cd ~/mailman
$ git clone https://gitlab.com/mailman/django-mailman3.git
$ cd django-mailman3
$ pip install --e .
$ cd ..
\end{verbatim}

To run the tests:

\begin{verbatim}
$ tox --e py37-django21
\end{verbatim}

### 6.2.9 Set up and run Postorius

The Postorius documentation, including a more extensive setup guide, can be found here: \url{http://postorius.readthedocs.org/}
Make sure to install mailmanclient and django-mailman3 before setting up Postorius. (If you’re following this guide in order, you’ve just done that.)

Get the code and run setup. Make sure you’re in venv which has Python 3.5+ for Postorius:

```
$ cd ~/mailman
$ git clone https://gitlab.com/mailman/postorius.git
$ cd postorius
$ pip install -e .
$ cd..
```

Postorius and HyperKitty both come with `example_project` directories with basic configuration so you can try them out. For this tutorial, however, we’ll be using a project that combines both instead.

You can run tests using:

```
$ tox -e py37-django21
```

### 6.2.10 Set up a Fake mail server

To be able to actually receive emails, you need to setup a mail server. Mailman core receives emails over LMTP Protocol, which most of the modern MTAs support. However, setup instructions are provided only for Postfix, Exim4 and qmail. Please refer to the MTA documentation at Mailman Core for the details.

You will also have to add some settings to your django configuration. The setup instructions are provided in django’s email documentation.

For development setup, you don’t _have_ to install a working MTA. You can add the following to your `mailman.cfg` to make sure that it doesn’t try to send emails out:

```
[devmode]
enabled: yes
recipient: you@yourdomain.com
```

```
[mta]
smtplib: 9025
lmtp_port: 9024
incoming: mailman.testing.mta.FakeMTA
```

Also, in Django you can add the following configuration to your `settings.py`:

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

This writes everything to `stdout`. There are other email backends available to use for testing like `django.core.mail.backends.filebased.EmailBackend` that one can use to write outgoing emails to a file on disk. Please see the docs for other options.

### 6.2.11 Set up and run HyperKitty


Make sure to install mailmanclient and django-mailman3 before setting up Hyperkitty. (If you’re following this guide in order, you’ve just done that.)

HyperKitty’s default configuration uses the Whoosh search engine in the backend. Install Whoosh using:
Get the code and run setup:

```
$ cd ~/mailman
$ git clone https://gitlab.com/mailman/hyperkitty.git
$ cd hyperkitty
$ pip install -e .
$ cd ..
```

Postorius and HyperKitty both come with example_project directories with basic configuration so you can try them out. By default, they both use port 8000, so if you do want to run both example projects at the same time, do remember that you’ll need to specify a different port on the command line for one of them.

You can run tests using:

```
$ tox -e py37-django21
```

However, we’re going to run them both in a single Django instance at the end of this guide, so don’t worry about ports right now.

### 6.2.12 Set up mailman-hyperkitty

`mailman-hyperkitty` is the package that actually sends the incoming emails to HyperKitty for archiving. Note that this is one of the components that uses Python 3.

Setting it up:

```
$ cd ~/mailman
$ git clone https://gitlab.com/mailman/mailman-hyperkitty.git
$ cd mailman-hyperkitty
$ pip install -e .
$ cd ..
```

You’ll need to fix the default `mailman-hyperkitty.cfg` file to use the correct url for HyperKitty. If you’re running it on `http://localhost:8002` then you need to change `base_url` to match that.

You can run tests using:

```
$ tox -e py37-coverage
```

### 6.2.13 Link Mailman to HyperKitty

Now you have to enable HyperKitty in Mailman. To do that, edit the `mailman.cfg` (in `~/mailman/mailman/var/etc`, or wherever the output of `mailman info` says it is) and add the following config. Note that you need to fill in the absolute path to your `mailman-hyperkitty.cfg` in the configuration below:

```
# mailman.cfg
[archiver.hyperkitty]
class: mailman_hyperkitty.Archiver
enable: yes
configuration: <absolute path to mailman-hyperkitty.cfg>
```
6.2.14 Run the Mailman Suite (combined hyperkitty+postorius)

You can run HyperKitty and Postorius as separate applications, but many developers are going to want to run them on a single server. The configuration files for this are in a repository called mailman-suite.

The first time you run the suite, you will want to set up a superuser account. This is the account you will use in the web interface to set up your first domains. Please enter an email address otherwise the database won’t be setup correctly and you will run into errors later:

```
$ cd ~/mailman
$ git clone https://gitlab.com/mailman/mailman-suite.git
$ cd mailman-suite/mailman-suite_project
$ python manage.py migrate
$ python manage.py createsuperuser
```

You’ll want to run the following commands in a window where you can leave them running, since it dumps all the django logs to the console:

```
$ python manage.py runserver
```

At this point, you should be able to see Mailman Suite running! In the default setup, you can go to http://127.0.0.1:8000 and start poking around. You should be able to use the superuser account you created to log in and create a domain and then some lists.

The default config file uses a dummy email backend created by this line in settings.py:

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

Using this backend, all emails will be printed to the Postorius console (rather than sent as email) so you can get the url to verify your email from the console. You can also use FileBackend to write emails to a file on disk.

Don’t leave the console email backend configured and running once you get to the point where you want to send real emails, though!

6.3 Contributing to Mailman Documentation

GNU Mailman project reStructuredText (.rst) markup language to write documentation. Each Mailman subproject is hosted on Gitlab under GNU Mailman Group.

This is a basic getting started guide for reStructuredText.

6.3.1 0. Create an issue

This is a Step 0 because it can be skipped for things that already have an issue. It is important to first open an issue before working on something to save yourself from going into a direction that Maintainers don’t think is best and also to let others know you are working on fixing/adding something so they don’t duplicate your efforts while you are at it.

This step can also be skipped for trivial changes like typos, grammar fixes, formatting fixes etc. But for any substantial change like proposing to use different method to do things or changing the default recommendations, please open a issue first.

6.3.2 1. Edit on Gitlab

Each project can have different documentation structure, but it is easy to find the exact file to edit by simply going to the right documentation page and looking for the Edit on Gitlab in the top right corner of the page.
This should take you to the source of the current page in Gitlab.

### 6.3.3 2. Fork the project

If this is the first time you are doing this, you might be asked to *Fork* the project. A Fork is a copy of source code that is copied into your own account so that you can make changes and propose it to be included upstream using a Merge Request.

### 6.3.4 3. Make changes and commit

Once Gitlab is done forking the project, you will be able to Edit the file in your Fork. This will happen only once for a single project and subsequent times you want to edit, it will skip the “Fork” step.
You can preview your changes as rendered HTML. This will catch some of the syntax errors in the changes you made. Once you are satisfied with your changes, you scroll down. There are two fields:

- **Commit Message**: This is a brief description of the changes you made. You want to write a single line summary, leave a blank line and write more if you have to. For example, this is a sample commit message:

```
Update packages requires for installing on Debian

Since cryptography now requires Rust build toolchain, this adds instructions to install Rust and Cargo in Debian.
```

- **Target Branch**: This is simply a unique name, without spaces. Each branch is a unique workspace where you can track individual works separately. This allows you to work on more than one changes without one interfering with other.

- **Start a new merge request**: Make sure that this box is checked. This will take you to create a “N”ew merge request” page in the next step.

Finally, you can click on the **Commit changes** button. This will take you to a page where you can propose your edits to be added to upstream project. This is called a “Merge Request”.

---

60 Chapter 6. The Contributor Guide
6.3.5 4. New merge request

There are a lot of options on this page, but the most important ones are:

- **Title**: This should be pre-filled with the first line of your commit, but you can edit this to be something else.
- **Description**: This is a short description of your changes, something that you can use to describe why the changes were made if there isn’t an issue associated with it.

At the bottom of the same page, you will see all the changes you are proposing to be included upstream if you go to the *Changes* tab at the bottom.

Finally, you can click the *Submit merge request* button to create a new merge request for the upstream project.

6.3.6 5. Preview your merge request

Using our integration with ReadTheDocs, where our documentation is hosted, you can preview your changes in a completely built format. You can go to the ReadTheDocs project page and look for the build of your merge request. You will see any build failures reported there.
After you choose your specific build, that will take you to the build output page.

Configure your documentation builds! Adding a `readthedocs.yml` file to your project is the recommended way to configure your documentation builds. You can declare dependencies, set up submodules, and many other great features.

When you find the build for your documentation, you can click at the `view docs` to look at a preview version of entire...
documentation with your changes. Make sure the Merge Request number matches your own.

6.3.7 6. Make changes to merge request

If you want to make more changes, you can do that by going to Changes tab on your Merge Request page and Clicking on the vertical three dot.

From the dropdown, choose the Edit in single-file editor, which will take you to the same page as Step 3. You can go through the same steps, just make sure to commit to the same branch you created for the merge request so that all the changes are stacked in the same merge request.

After this, Mailman maintainers will either accept the Merge Request or request some changes. You can go through the Step 6 and then Step 3 to make any requested changes.
C
COMPRESS_PRECOMPILERS
  setting, 40

D
DATETIME_FORMAT
  setting, 40

F
FILTER_VHOST
  setting, 40

L
LOGIN_REDIRECT_URL
  setting, 40
LOGIN_URL
  setting, 40
LOGOUT_URL
  setting, 40

M
MAILMAN_ARCHIVER_KEY
  setting, 39
MAILMAN_REST_API_PASS
  setting, 39
MAILMAN_REST_API_URL
  setting, 39
MAILMAN_REST_API_USER
  setting, 39

P
POSTORIUS_TEMPLATE_BASE_URL
  setting, 40

S
setting
  COMPRESS_PRECOMPILERS, 40
  DATETIME_FORMAT, 40
  FILTER_VHOST, 40

T
TIME_FORMAT
  setting, 40