

1 Arch Linux

1.1 Maintenance

```
#check file size  
du -sh .cache/  
#remove a file  
rm -rt .cache/  
#delete what you don't need in .config file
```

specific maintenance:

```
#check the failed systems  
systemctl --failed  
#check the systemd journal  
sudo journalctl -p 3-xb  
#if the system doesn't boots then ctrl+alt+shift then timeshift -restore  
#then update mirrors  
#clar cache  
  
#then to update the whole system use:  
sudo pacman -Syyu  
#to check system updates  
sudo pacman -Syu  
#if you want to remove all packages in the drive use  
sudo pacman -Scc  
#remove all unwanted dependencies  
paru -Yc  
#remove orphan packages  
sudo pacman -Rns \$(pacman -Qdtq)  
#sudo pacman -Syyy Synchronise data use "mirror1"
```

1.2 Print in arch linux

install packages: usbutils, lusb, cups
use this to make cups usable

```
sudo systemctl enable cups  
sudo systemctl start cups  
localhost:631  
  
lp -d HP_Officejey_Pro_8600]
```

1.3 configure date and time

```
hwclock --set --date = "04/32/2021 19:00:00"  
hwclock -hctosys
```

1.4 Configure wireless

```
#when entering an iso  
iwctl  
#then in the ui  
  
#to list all available devices  
device list
```

```

#to scan networks
station <device> scan

#to get networks
station <device> get-network

#to connect to a network
station <device> connect "<name of network>"

#to check if the connection is stable
ping -c s 8.8.8.8

#don't forget before rebooting the iso run
pacman nmtui

```

from Arch Water Linux

```

# to acces the gui for the internet
nmtui
# solve temporary failure in name resolution
# change the /etc/resolve.conf file to nameserver 8.8.8.8

# restart the resolved daemon
sudo systemctl restart systemd-resolved.service
# check that the daemon is running and active
sudo systemctl status systemd-resolved.service

```

dwm basic configuration

```

#MODKEY + shift + q to restart X server
startx # to start the X server

```

1.5 mount devices

mount usb sticks:

```

#to mount a usb stick
mount /dev/sdb1 /mnt/<destination folder>
#to unmount a sub stick
umount /dev/sdb1

```

mount an android device:

```

#to mount and android device
simple-mtpfs --device 1 tablet/

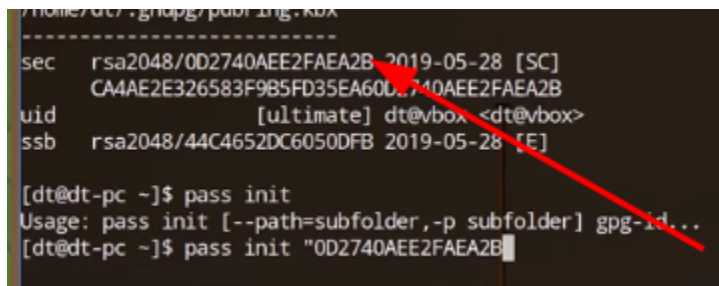
#to unmount an android device
fusermount -u /tablet

```

1.6 import export passwords from pass

export passwords:

```
# to list first the gpg keys
gpg --list-secret-keys --keyid-format LONG
```

A terminal window showing the output of the command 'gpg --list-secret-keys --keyid-format LONG'. The output lists a secret key (sec) and a subkey (ssb) for the user 'dt@vbox'. The secret key is rsa2048/0D2740AEE2FAEA2B, created on 2019-05-28, with a subkey CA4AE2E326583F985FD35EA60D2740AEE2FAEA2B. The subkey is also rsa2048/44C4652DC6050DFB, created on 2019-05-28. Below the key listing, the user runs 'pass init' and is prompted with 'Usage: pass init [--path=subfolder,-p subfolder] gpg-id...'. The user then runs 'pass init "0D2740AEE2FAEA2B"'.

```
-----
sec  rsa2048/0D2740AEE2FAEA2B 2019-05-28 [SC]
    CA4AE2E326583F985FD35EA60D2740AEE2FAEA2B
uid          [ultimate] dt@vbox <dt@vbox>
ssb  rsa2048/44C4652DC6050DFB 2019-05-28 [E]

[dt@dt-pc ~]$ pass init
Usage: pass init [--path=subfolder,-p subfolder] gpg-id...
[dt@dt-pc ~]$ pass init "0D2740AEE2FAEA2B"
```

```
# to create the export files
# save this files in a usb and use it later
gpg --output MY_FILENAME_public.gpg --armor --export GPG_PUB_KEY
gpg --output MY_FILENAME_secret.gpg --armor --export-secret-key GPG_PUB_KEY
# in other pc import the gpg keys
gpg --import MY_FILENAME_pub.gpg
gpg --allow-secret-key-import --import MY_FILENAME_sec.gpg
# now copy the .password-store folder from the main machine and paste it into t
```

2 Install python version

```
# download the python version you need from https://www.python.org/downloads/source/
# unpack in the .local/src/pythonversions/pythonVersion.tgz
tar zxvf pythonVersion.tgz
cd pythonVersion
# Install the python version
./configure
make
sudo make install
make clean
# check python version
python[pthon_version] --version
# create a python environment using that python version
python[pthon_version] -m venv venv/
# source the environment
source venv/bin/activate
# for deactivating
deactivate
```

2.1 removing bloatware from android

```
# install the android developer tools
paru -S android-tools
# in your android enable developer options by about phone -> build number 7 times
# then enable usb debugging

# now in your linux sistem type in your terminal
adb devices # to see if device is succesfully connected
```

```
adb shell # to start the shell

# to delete an app
pm uninstall -k --user -0 (package-name)

# to see the names of apps use app inspector from the google store
```

2.2 deploy python django application aws

```
## modify the django project

# settings.py

STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static')

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

## now collect static
source venv/bin/activate

python manage.py collectstatic

### aws
# create an account in aws
# find ec2 and click on launch instance
# select ubuntu server 18 free trial
# see all the instances you are running
# change the name of your instance (on the very left side of the row you can do that)
# you will be prompted to create a new id, so create the new keypair and save it to your linux machine
# the right click on the instance id and click connect
# paste the ssh code in the folder were your keypair is, for example:
ssh -i "password-generator-django.pem" ubuntu@ec2-54-242-121-76.compute-1.amazonaws.com

# now that you are connected sudo update and upgrade the server
sudo apt-get update
sudo apt-get upgrade -y

# you will have to use gnuix gunicorn
python3 --version
python3 -m venv venv/
apt-get install python3-venv
python3 -m venv venv/

ls
# use the environment
source venv/bin/activate

# install django
```

```
pip3 install django
```

```
# install necessary packages for python
sudo apt install python3-dev build-essential
sudo apt install libssl
sudo aptinstall libssl-dev
sudo apt install libmysqlclient-dev
```

```
# install the requirements
pip install -r requirements.txt
```

```
# install django-ckeditor
pip3 install django-ckeditor
```

```
# git clone your github django project
```

```
git clone url.git
```

```
cd url
```

```
# install modules for deploy
pip3 install gunicorn
sudo apt-get install -y nginx
```

```
# start nginx
sudo nginx
```

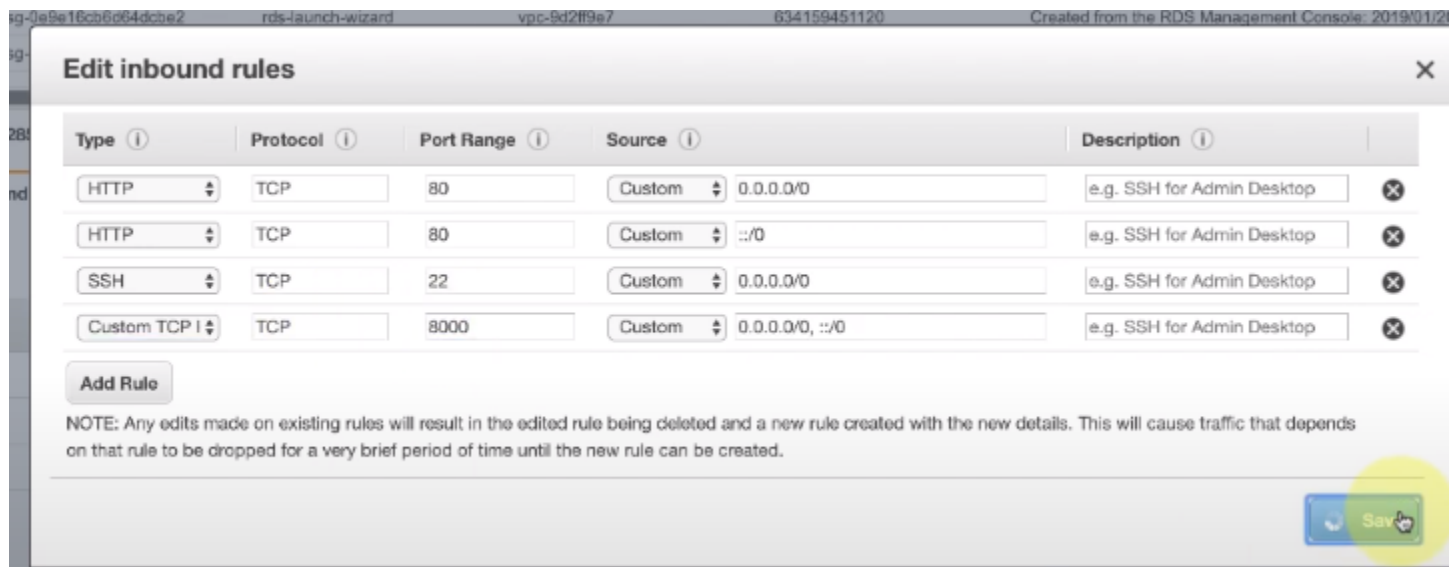
```
# configure security groups in for https and http
right click(on the instance row) -> networking -> change security group -> see the launch wizard associated
```

```
click security groups -> launch wizard n -> inbound -> edit -> add rule
```

Type ⓘ	Protocol ⓘ	Port Range ⓘ	Source ⓘ
SSH ▾	TCP	22	Custom ▾ 0.0.0.0/0
HTTP ▾	TCP	80	Anywhere ▾ 0.0.0.0/0, ::/0

Add Rule

NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new configuration. This may result in traffic on that rule to be dropped for a very brief period of time until the new rule can be created.



```
# to connect the gunicorn (which is the wsgi interface)
# tell gunicorn to use the wsgi.py
cd project/project
gunicorn --bind 0.0.0.0:8000 project.wsgi:application
# after this you should be able to access your web app using the link
# use the port 8000 example: google.com:8000

# supervisor makes sure your application is running always
sudo apt-get install -y supervisor
# create a configuration for supervisor
cd /etc/supervisor/conf.d/
touch gunicorn.conf
sudo touch gunicorn.conf

# edit it
sudo nvim gunicorn.conf

# inside the file
[program:gunicorn]
directory=/home/ubuntu/project
command=/home/ubuntu/env/bin/gunicorn --workers 3 --bind unix:/home/ubuntu/project/app.sock
testproject.wsgi:application

autostart=true
autorestart=true
stderr_logfile=/var/log/gunicorn.err.log
stdout_logfile=/var/log/gunicorn.out.log

[group:gunicorn]
programs:gunicorn

# save and exit
# outside the file
sudo mkdir /var/log/gunicorn
sudo supervisorctl reread
```

```

sudo supervisorctl update

sudo supervisorctl status

# nginx configuration
cd ~
cd /etc/nginx/sites-available
sudo touch django.conf
sudo nvim django.conf

# paste the code

server{
    listen 80;
    server_name ec2-3-91-188-252.compute-1.amazonaws.com;
    location / {
        include proxy_params;
        proxy_pass http://unix:/home/ubuntu/personal_portfolio/app.sock;
    }
    location /static/ {
        autoindex on;
        alias /home/ubuntu/personal_portfolio/static/;
    }
    location /media/ {
        root /home/ubuntu/personal_portfolio/;
    }
}

# save and exit
sudo nginx -t

# enable the link
sudo ln django.conf /etc/nginx/sites-enabled/

# save and exit
sudo nginx -t

sudo service nginx restart


#####
## now for setting static files
#####

# in settings.py
STATIC_URL = '/static/'

```

```

# in the html you are doing
{% load staticfiles %}

# in the templates section
'DIRS':[os.path.join(BASE_DIR, 'TestProject/templates')],

# open the server with the keys and cd
nvim /etc/nginx/sites-enabled/django.conf

# append

location /static/ {
    autoindex on;
    alias /home/ubuntu/ProjectFolder/MainProjectFolder/static/;
}

# outside the file
# open the nginx configuration to allow big pictures

cd /etc/nginx
sudo vim nginx.conf

#inside the http or server paste
client_max_body_size 4M;

sudo systemctl reload nginx ;

#####
## now for setup the database with django
#####

# create database
dtaabase section ->
RDS ->
create database ->
mysql ->
only enable options eligible for free usage ->
next ->
select the specific mysql version ->

```


database instance offered on the free tier ->
allocated storage
configure the name,username,password etc
allow public accesibility
choose default existing vpc security groups
database name

```
# now get the latest code in your github
git pull
# configure the database for the server
DATABASES = {
    # name
    'default' : {
        'ENGINE':'django.db.backends.mysql',
        'NAME':"database_name",
        'USER':"database_user",
        # change this manually in the server
        'PASSWORD':"*****",
        # click on the database, check endpoint & port for configuring
        'HOST','host',
        'PORT':'12312'
    }
}
# activate the environment
source venv/bin/activate

# install all the modules in the requirements
pip install django-mysql

# make the migrations
python manage.py makemigrations name_of_main_app
python manage.py migrate

# restart the server
sudo supervisorctl reload

sudo service nginx restart
```

2.3 use custom domain name for django

```
# search for a service called route 53
# create hosted zone with the domain name

# create record
simple routing -> copy the ipv4 address

click define simple record
```

```

# create an other record

simple routing ->
# in blog type www
# record type CNAME
# in the big box below CNAME
example.com
define record

# now in hosted zone details see the name servers

# copy the name servers into namecheap

# in namecheap manage the domain and change to custom dns
# paste the dns's

# now connect to the instance
cd /etc/nginx/sites-available/
sudo vim django.conf
# update the domain name
# exit
sudo service nginx restart

```

2.4 enable https

```

# search for certificate manager
# provision certificates
# request public certificate
# place domain name
example.com
# use dns validation
# confirm and request

## the request will be in progress

# open the dns menu

## add the CNAME to the dns configuration of the domain

# create record in route 53
# press continue

## validation not complete message

## in other tab open the route 53 console

## in the certificates tab reload table

## it will appear ineligible

```

```

## create tab and select ec2 instance
click Load balances

# create load balancer

name: example.com

click add listener -> https

# check all availability zones

# next configure security settings

# choose a certificate from ACM

# configure security groups

# change to the launch wizard of your instance

# next configure routing

# change the name to anything-tg

#select the instance and add to register

# select the instance now in the upper table

# next review

# create

# once created copy the dns name abd add it as a record

# open the route 53 tab

# create recod -> simple routing -> define simple record ->

# choose endpoint Ip adress or another value depending the record

# in the big box paste the newly copied dns

# create record

# if you get an error edit the record row that TTL = 300

# change to Ip adress or another ... to Alias aplication and clasic balancer

# region asia pacific mumbai

# choose the only load balancer available

# wait some minutes and it should be done

```

2.5 add https support

first on aws add to the security group (generally launch wizard) https support for a

now inside the server just add

```
sudo apt install certbot python3-certbot-nginx
```

```
sudo certbot --nginx -d yoursite.yourExtension -d www.yousite.yourExtension
```