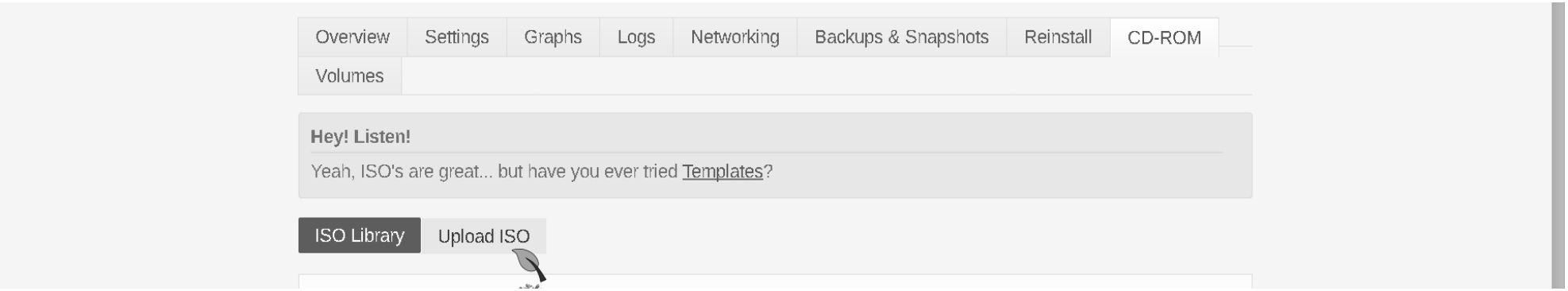
Self-hosting a static website in Slackware 15

Introduction

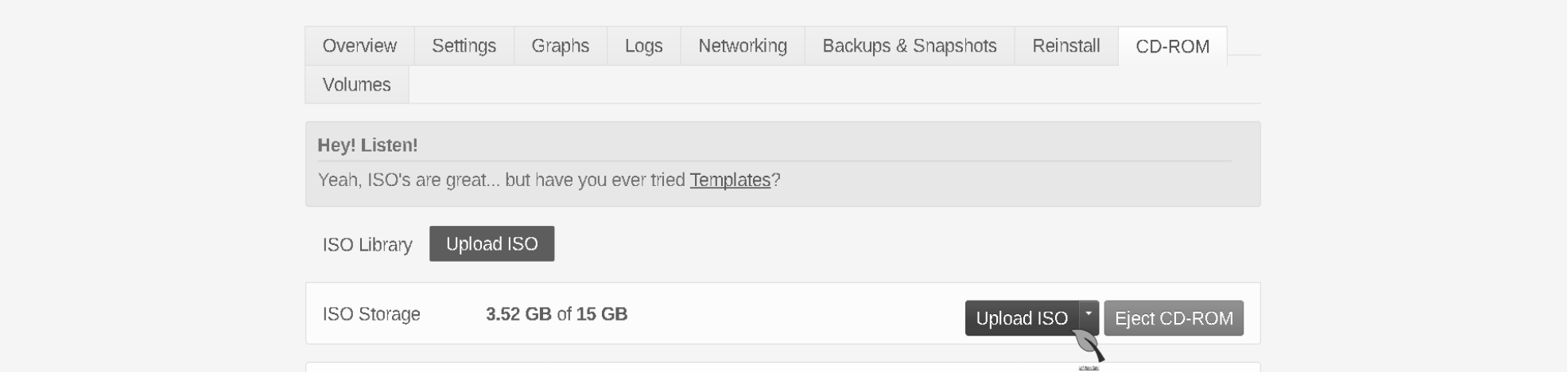
Did it again recently, and I thought I'll put the instructions up here just in case someone wants to try it too. The process is not that easy to figure out the first time, so I might as well save someone the trouble. Why even do it? Because having your own website is worth it, and a much better outlet than big tech social media, over which you have little control and can be kicked out from at any time. Why not Landchad? They cover stuff I don't need, or don't cover stuff I do need, or do it differently than I like. Don't get me wrong, it's a good project but I just like doing things my own way. And besides, some of their stuff is not applicable to Slackware. Why Slackware? Because it's all I know, and should be helpful to those familiar with it and not other distros / OSes. Anyway let's go:

Installing Slackware

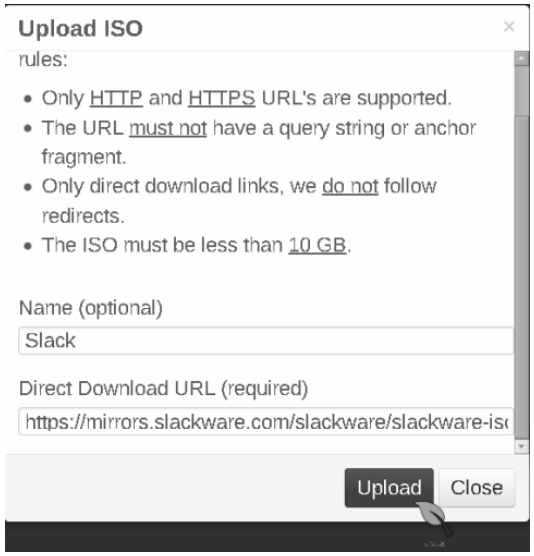
First of all, you either need a host supporting it by default (which AFAIK is only 99stack), or one that accepts custom ISOs. In case of the first option, you can skip this section; in case of the second, I will assume you picked BuyVM (however, any other will work - though portal options, etc might be slightly different). Login through Stallion, pick the *CD-ROM* menu option, and click *Upload ISO*:



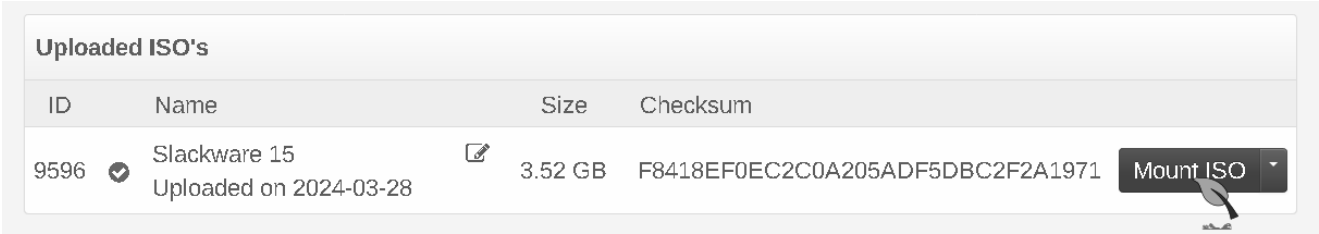
After that, click the *Upload ISO* that appeared further down:



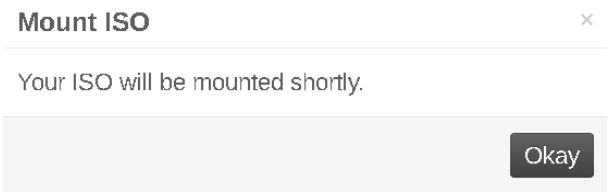
Then, input the Slackware 15 link (currently at https://mirrors.slackware.com/slackware/slackware-iso/slackware64-15.0-iso/slackware64-15.0-install-dvd.iso) into the second field, click *Upload*, and wait:



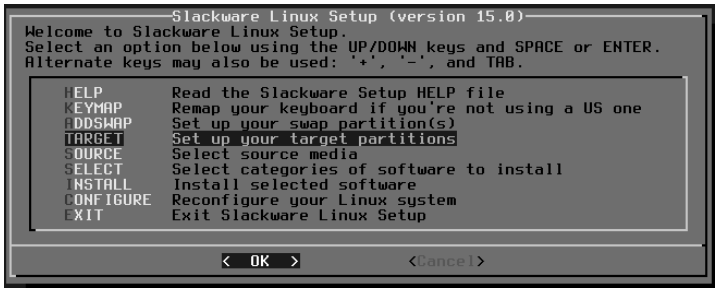
After it uploads, click *Mount ISO*:



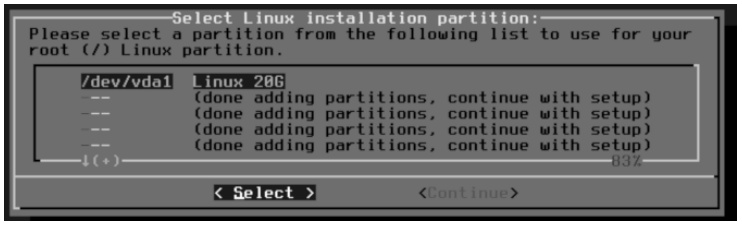
This should appear after:



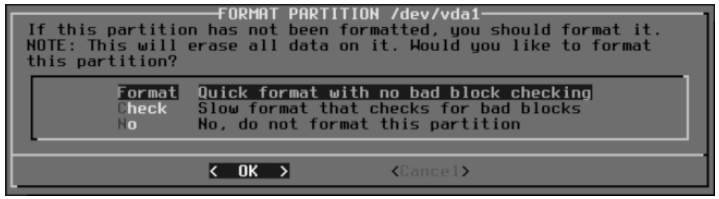
Login through Stallion's web-based console. You should see a message prompting you to choose a keyboard layout - just press Enter. Then type *"root"* and finally *"setup"* to start the Slackware installer. Pick *Target* on the screen that pops up:



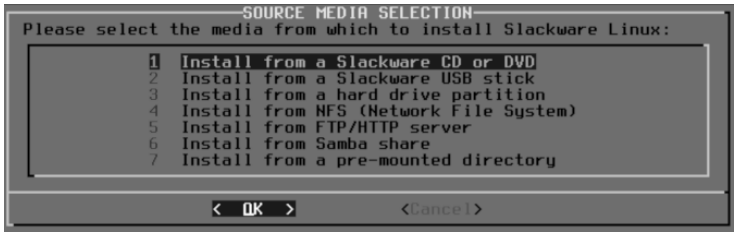
Pick */dev/vda1*, the only option:



Format as *ext4* (the others should work too, but I have no experience with them and they might differ in important ways)

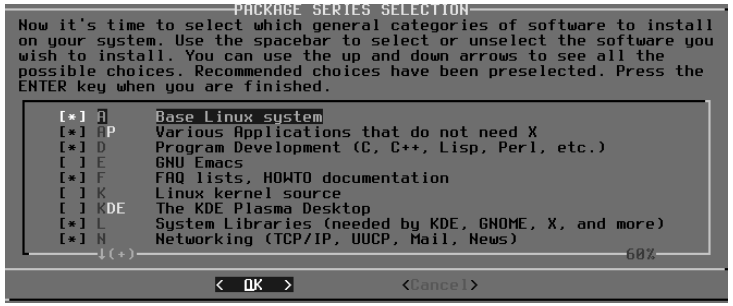


Install from CD:

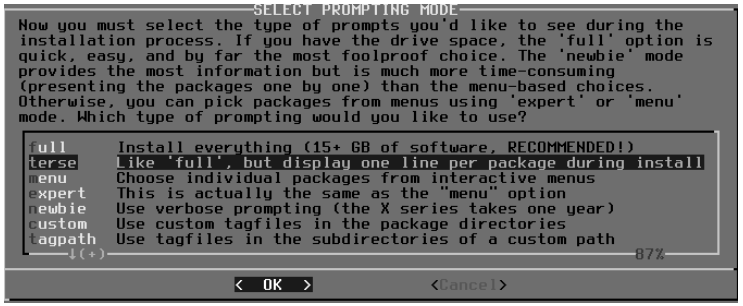


UPDATE: while setting up a third mirror I realized that this **might not always work**. You might have to mount the CD drive first, then install from the mounted directory (last option). But the rest of the steps will be the same. Moving on...

In the package series selection screen, you can freely disable *"E"*, *"K"*, *"KDE"*, *"T"*, *"TCL"*, *"X"*, *"XAP"*, *"XFCE"* and *"Y"* series. Theoretically, you can also skip the *"D"* series, but then you'll need to download the perl package (the only one you'll need from that series) later:

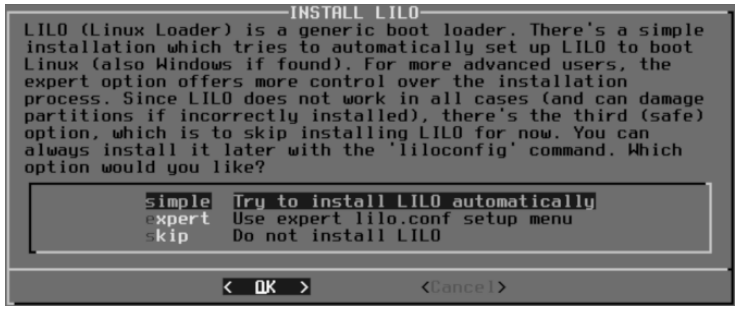


After you're done, pick *"terse"* and wait.

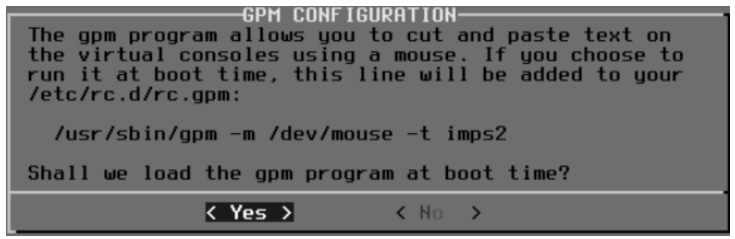


"Skip" making the USB boot stick on the next screen.

Since we only have one OS and partition, lilo can be setup automatically.

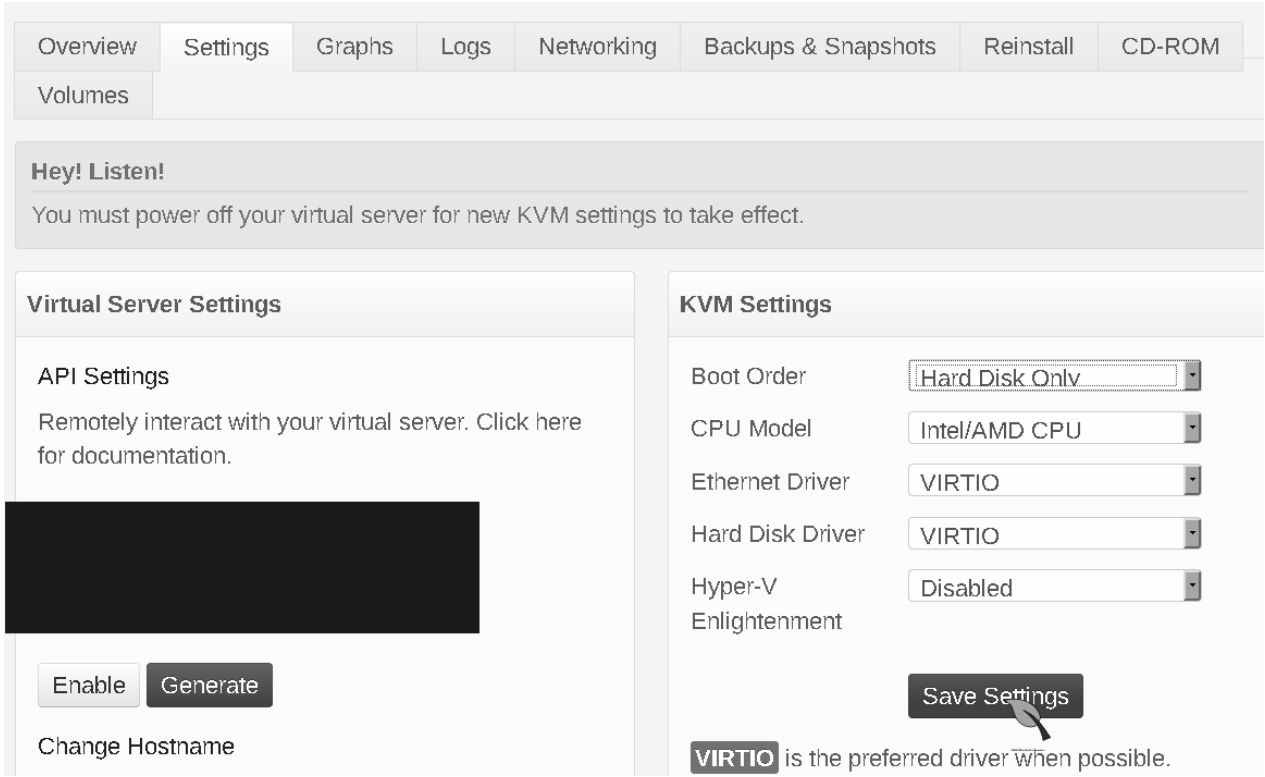


Next three screens can be OK-ed through. After that, choose to start gpm at boot time:



On the remaining screens, don't *"configure your network"*. Startup services can be left as they are, as well.

Slackware has now been installed. Go back to Stallion and switch to hard drive boot again (otherwise, it will go back to the install screen after reboot).



Now boot inside the web VNC console at the top right of Stallion. There you'll be able to type commands before ssh is installed.

Updating OS certs and setting up SSH

The first thing we're going to do once Slackware is installed is **upgrade the ca-certificates package** - as without that, some hosts will fail to verify their SSL connections. Go to https://slackware.nl/slakfinder/ and find the newest version. Since we don't have SSH yet, we're going to have to use the direct VNC (or the web console) connection for this. Type *"wget [package_address]"*, then *"installpkg [package_name]"*. Currently, the link to the newest version is http://ftp.osuosl.org/pub/slackware/slackware64-current/slackware64/n/ca-certificates-20250131-noarch-1.txz, so the commands would be *"wget http://ftp.osuosl.org/pub/slackware/slackware64-current/slackware64/n/ca-certificates-20250131-noarch-1.txz"* and *"upgradepkg ca-certificates-20250131-noarch-1.txz"*. This might change at any time though, so do not rely on this direct link.

By default, trusted SSL certificates are stored in the folders */etc/ssl/certs* and */usr/share/ca-certificates/mozilla*. But for some reason, only certs from the folder /usr/local/share/ca-certificates get detected. Yet, it doesn't exist by default, so create it (*"mkdir /usr/local/share/ca-certificates"*). Then copy the certs from folder #1 to #2; *"cd /usr/share/ca-certificates/mozilla/; cp * /usr/local/share/ca-certificates/"*.

After this is done, actually run the script by typing *"update-ca-certificates"*. This where you'll need the Perl package if you've skipped the *"D"* serie. If the command worked properly, you should get a message like *"241 added, 0 removed, done"*.

By default, root is the only available Slackware user, but it's not good practice to allow him to ssh in, so let's create a regular user (**all commands from now on will assume being run as root**):

```
adduser luna
```

You can enter past everything other than the password field (remember to pick a strong one). After you're done, edit your ssh config file at */etc/ssh/sshd_config* (by eg. nano) to allow server administration without VNC. You need this section somewhere in it:

```
Port 281
AddressFamily any
ListenAddress 0.0.0.0:281
```

The port can be any, but should probably be nonstandard to discourage hacking attempts. After that, add your user to the list of allowed users:

```
AllowUsers luna
```

Save. Restart sshd with *"sh /etc/rc.d/rc.sshd restart"*. Put *"dhcpcd"* in your */etc/rc.d/rc.local* script. Reboot in Stallion. Now trash the VNC, as you will be able to login with this ssh command: *"ssh ssh://user@ipaddress:port"*, for example *"ssh ssh://luna@71.24.85.36:281"*. If anonymity is a concern (and it should be) you can put *"proxychains4"* before the first *"ssh"* (proxychains is explained here).

Buying domains

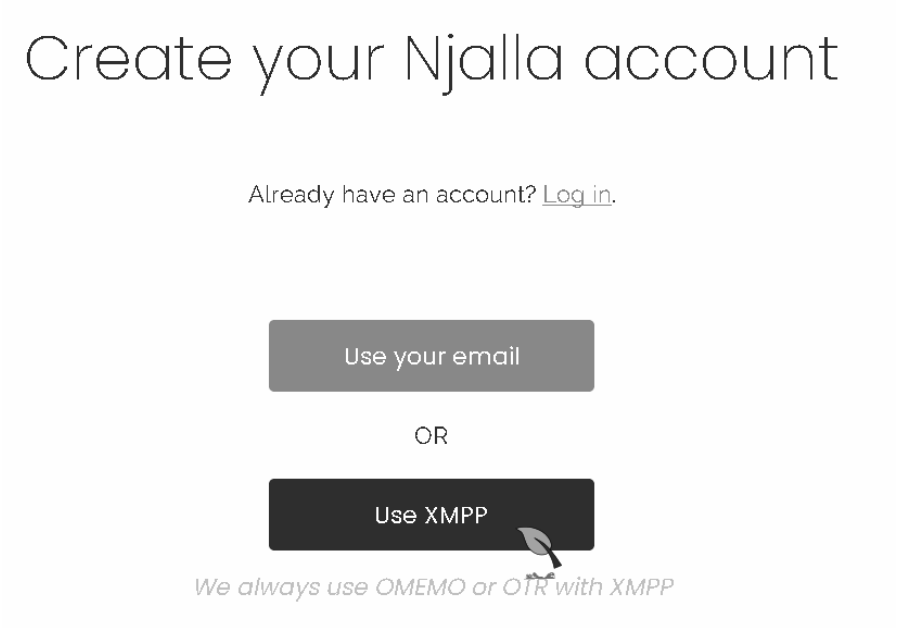
Though you can technically have a website without it - if you are ever hoping to have actual viewership - then a domain is an absolute necessity. It is what allows people to type in your website address (eg *digdeeper.love*) instead of an unrememberable IP address. Basically the domain provider could be considered analogous to a translator of a really hard language (IP numbers) to an easy one (domain names), so that your viewers only need to know the easy one. Anyway, my favorite domain registrar is njalla, for these reasons:

- They don't require any personal data (as one of the few such registrars)
- They accept monero and other cryptocurrency
- Non-restrictive ToS so you're unlikely to get canceled (**UPDATE June 2026**: recently I've come across some reports (archive) of domains getting arbitrarily deleted without recourse - *"This happened around a year ago. We ran a site for people recovering from addictions. Someone reported us and Njalla shut us down. We contacted them and pointed out that we do not allow or promote illegal content. They apologized and restored the service, and then within a day shut it down again and told us to get lost. They never mentioned the ToS was violated and just said that they "don't want to host us"."*; please think long and hard whether you want to host

your domains here, knowing this)

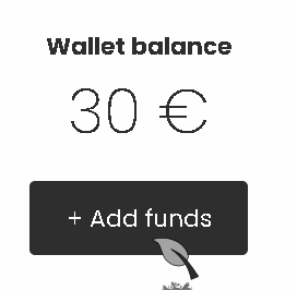
- Site is fast and functional (incognet is NOT!!), even the JS-dependent domain search
- Confirmation through XMPP is allowed - I don't think anyone else is doing that
- Messages from them to you can be sent back in encrypted form; I don't think anyone else is doing that, either
- They double up as a DNS provider (redirection service); incognet does NOT!!
- They allow domain transfer away from themselves (incognet does NOT!!)

There is a caveat that has to be mentioned in regards to the first point. Namely that the reason this is even possible is so that **njalla substitutes their own data in place of yours**, to the **actual** domain registrar, that they are only a middleman to. This means that **you are not the legal owner of your domain** - but if you're here, you're surely an outlaw for whom this does not matter. And if you one day decide to reveal yourself (or get doxed), njalla will allow you to claim the ownership of your domain with your real identity. This will mean - though - that governments will be suddenly able to grab you for the wrongthink in there. But hey, at least njalla offers this option. Anyway, click here to register:

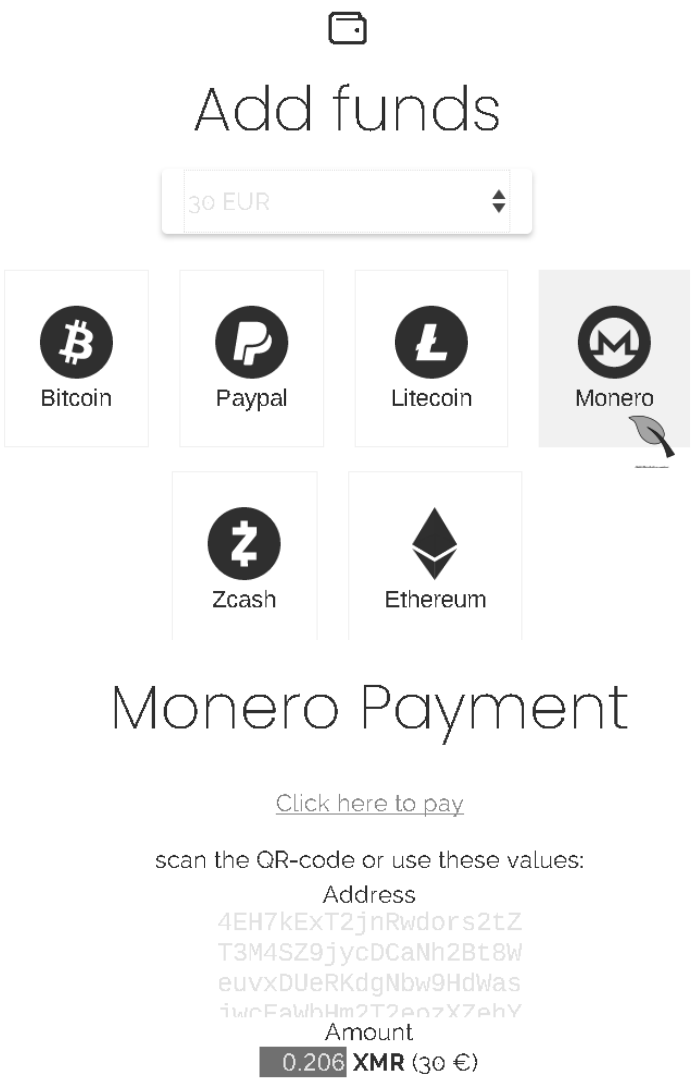


Now input your XMPP address and password (for the njalla account, not your XMPP one), and pick your encryption type (**your client and server must support it, or else the confirmation message won't come**!):

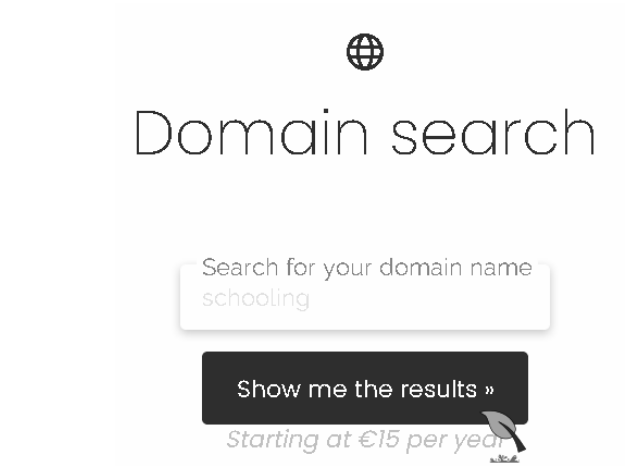
After this, a message from *no-reply@njal.la* should immediately appear inside your XMPP client, containing a confirmation link - click it. Now go to the login page, fill in the details, and click *Sign in*. Direct your eyes to the right part of the page:



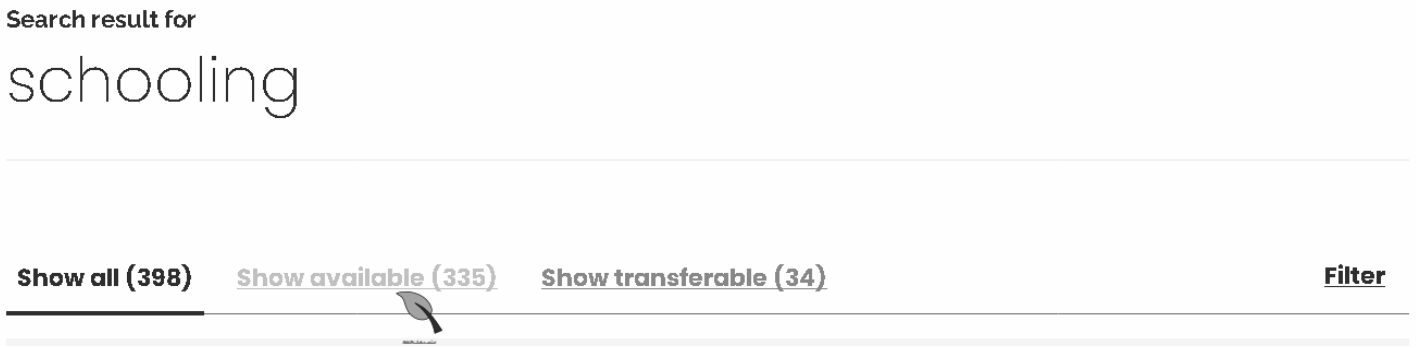
I have some funds already transfered, but for you the amount will be zero - so let's add some. Choose the amount you'd like to add (30 euros should be enough to buy the average-cost domain names for one year, and the cheap ones for two), then pick Monero.



Just send the listed amount from your Monero wallet to the address that appears, and wait until confirmation. In Pale Moon the entire address appears to not display; to grab it you can also visit the *"Click here to pay"* link, and copy what's between *"monero:"* and *"?"* - which should look like *"4EH7kExT2jnRwdors2tZT3M4SZ9jycDCaNh2Bt8WeuvxDUeRKdgNbw9HdWasjwcEaWbHm2T2eozXZehYhHvAx7xdF5ZWwKM1Hdf4Dt9Y2* (seems to be 106 characters every time), for example. If you have a monero wallet mapped inside your browser, then it should simply open. Might get a screenshot at some point, but not now. Also, actually getting the Monero is out of the scope of this guide. Once it confirms, go here and type the first part (the one before the TLD, or dot) of the domain that you want:



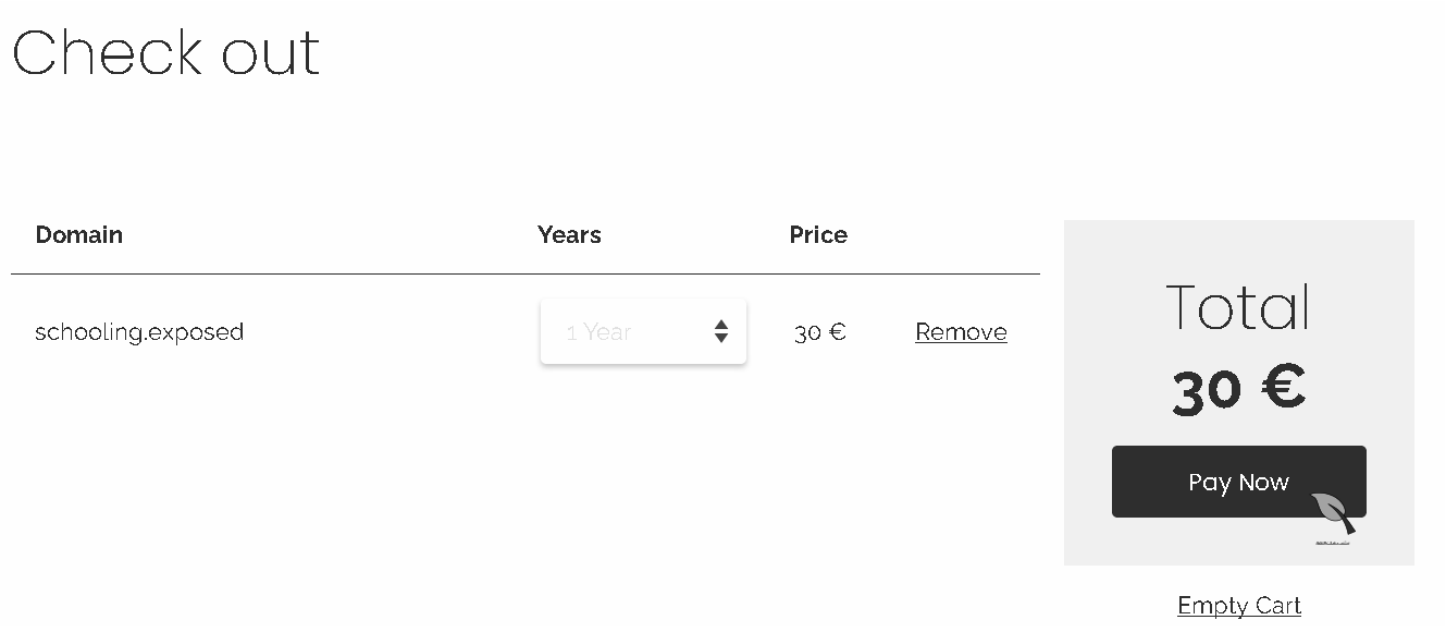
I thought long and hard about what domain to use as an example, as I wanted to actually buy it for this guide's purposes, so it had to somehow be useful to me. Then I remembered that *.exposed* is an actual TLD (which is hilarious as fuck). I tried wikipedia.exposed but the psychopaths running it grabbed it already, same with mozilla. Capitalism works but is kinda boring; then I remembered that I have an article about the schooling system, so I thought to try that. Anyway - after the results display - click the *"Show available"* filter category, so that you're not bothered with the domains that are already taken - unless you just want to see what TLDs exist at all:



Scrolling down a little reveals *schooling.exposed* as available. Beautiful! Let's grab it:

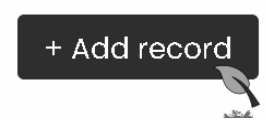
Scroll as far up as you can and click *“Check out!”*:

I'm only going to buy it for one year (default), as I lack anymore funds. I'm just a starving artist, you see:

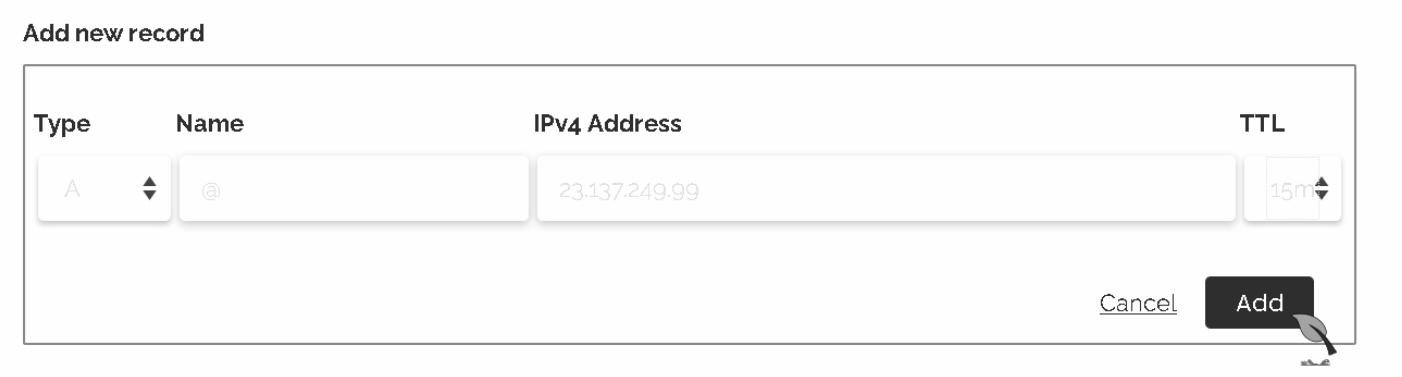


After buying the domain, go to the domain list and click *“Manage”* for the relevant one:

Now click this button:



Add an *"A"* record for the domain, which basically means "redirect this address to this IP number" (you can find your VPS's IP inside Stallion or other web control panel):



If you did it properly, in some time you should be able to connect to your website by typing *"http://schooling.exposed"* (or whatever domain you bought) into your browser's address bar. But it will do nothing without...

Setting up a website server (darkhttpd)

A webserver is what actually serves (shows) the website files to the viewer. We'll be using darkhttpd because it's small, easy to install and configure, and has everything required for a static site. Our current system doesn't even allow us to install packages at all, so let's fix that.

You will need spkg (a dependency for slapt-get), and slapt-get itself. Download the former with *"wget https://download.salixos.org/x86_64/15.0/salix/a/spkg-1.7-x86_64-2gv.tgz"* and the latter with *"wget https://download.salixos.org/x86_64/15.0/salix/ap/slapt-get-0.11.6-x86_64-2gv.txz"*. Then install both with installpkg. Update the repositories with *"slapt-get --update"* and you're good to go. Now install darkhttpd with *"slapt-get --install darkhttpd"*. Then make an user and a group that will use it:

```
groupadd darkhttpd
useradd --no-user-group --system -s /sbin/nologin --gid darkhttpd darkhttpd
```

This theoretically improves security, in that darkhttpd will have access only to folders containing the website files. Though I suspect that the practical effect won't be that noticable, because user submissions are not accepted due to the static-only nature. Still a good practice in case malicious code ends up in the binary by repo compromise or whatever. Anyway, create a folder that will contain your website files:

```
mkdir /var/www/mysite
```

To be able to later upload files conveniently (and you'll be doing that a lot, so it better be convenient), it's recommended (by me :D) to use a graphical FTP program like gFTP. Again, it's the most secure to create a separate user for it, so let's do it now:

```
useradd --no-user-group --gid darkhttpd -s /bin/bash -d /home/gftp gftp
passwd gftp
```

We're making darkhttpd his main group so that when a new file is added to the website folders, it will automatically have read permissions for the server applied (otherwise, with default group "users", you'd have to change the ownership of every newly uploaded file, which gFTP can't do). As usual, remember to pick a strong password. After you're done, change the permission of the website folder like this:

```
chown -R gftp:darkhttpd /var/www/mysite
chmod -R 640 /var/www/mysite
chmod -R ug+X /var/www/mysite
```

This accomplishes three very important things. The first is that you will be able to upload files through your ftp client (thanks to 6 - AKA "read plus write" - permissions for the owner). The second is that the website server gets only read access (4 permissions) to the website folders (it doesn't need more because user submissions are - again - not possible due to its static nature). The last command is so that both the owner (you, the uploader) and the server (AKA website visitor) will be able to actually enter folders (execute permission), but not execute files (that's why the X is large instead of small, so the execute permission is only applied to folders). The gFTP program actually uses the ssh protocol to upload, so you need to add the gftp user to */etc/ssh/sshd_config*. Just change *"AllowUsers luna"* to *"AllowUsers luna gftp"* and save. Finally, run the server with:

```
darkhttpd /var/www/mysite --addr 0.0.0.0 --uid darkhttpd --gid darkhttpd --daemon&
```

You can now visit your site at your server's IP address. If you have a domain pointed to it, you can also use that.

To enable serving .xhtml files, you need to create a file named *mimetypes* (in this example, in the root of your website folder) with the contents *"text/html xhtml "*. Then run the server with command:

> *darkhttpd /var/www/mysite --addr 0.0.0.0 --uid darkhttpd --gid darkhttpd --daemon* ***--mimetypes /var/www/mysite/mimetypes --index index.xhtml****&*

The last part tells it to pick index.xhtml as the default index file, instead of html as it would usually be. Anyway, let's say you bought a domain entirely so that it would redirect to a single article on a domain you already have (like I did with schooling.exposed). Doing that in darkhttpd is kind of hacky, but indeed possible. You'll need to create a folder such as */var/www/mysite/schooling* and put in there an index.xhtml file with these contents:

> *<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"> <html xmlns = "http://www.w3.org/1999/xhtml"> <head> <meta http-equiv= "Refresh" content= "0; url='./articles/school.xhtml'" /> </head> </html>*

Then run darkhttpd with this command:

> *darkhttpd /var/www/mysite --addr 0.0.0.0 --uid darkhttpd --gid darkhttpd --daemon --mimetypes /var/www/mysite/mimetypes* ***--forward schooling.exposed https://yourcurrentdomain.com/schooling/*** *--index index.xhtml&*

What this does is basically, whenever darkhttpd detects a request for the domain *schooling.exposed*, it will instead send the client to *https://yourcurrentdomain.com/schooling/*. Obviously replace that with your already existing one. The reason this is so convoluted is that somehow darkhttpd's forwarding option fails when a file is given. So, the easiest way to bypass it is to give it a folder, where the default loaded file is the index, which will send visitors somewhere else via the meta tag. The "sending page" will be visible in the address bar for a short while, but shouldn't be too noticable. Of course, replace the school.xhtml file with wherever you want to redirect. You can repeat this for more domains than one, if you have them.

Getting an I2P address

Instructions adapted from Darknet setup in Slackware-based distros. First install i2pd:

> *sudo slapt-src --install i2pd:2.48.0*

By default, the slackbuild puts the config file into to */usr/doc* for some insane reason. So move it from there by *"mv /usr/doc/i2pd-2.48.0/i2pd.conf /etc/"*. Then edit the file (*/etc/i2pd.conf*) to uncomment these lines:

> *# tunconf = /var/lib/i2pd/tunnels.conf*

 # tunnelsdir = /var/lib/i2pd/tunnels.d
 # pidfile = /run/i2pd.pid
 # logfile = /var/log/i2pd/i2pd.log
 # daemon = true
 # port = 4567

Change the pidfile line to *"/var/run/i2pd/i2pd.pid"*. This is because the i2p user that we'll be making needs to own the folder, and he can't own /run. Then change the port to something else. Now type:

sudo mkdir /var/lib/i2pd; sudo mkdir /var/log/i2pd; sudo mkdir /var/run/i2pd

This creates the folders i2pd will use (edit: you might want to insert the /var/run one in */etc/rc.d/rc.local*, as it seems to disappear on reboot). Next, create the *i2p* user and group that will run the daemon:

sudo groupadd i2p; sudo useradd i2p --gid i2p --base-dir /var/lib/i2pd --system --no-user-group --shell /bin/sh --comment "I2P Daemon"

We're giving him a real shell since i2pd (unlike tor) does not support downgrading privileges, so we will have to *su* into the user to run it - which would not have been possible with a false shell. Now move the other necessary files to the folders used by i2p (I don't get why all this isn't done automatically, but whatever):

sudo cp -R /usr/doc/i2pd-2.48.0/tunnels.d /var/lib/i2pd/; sudo cp /usr/doc/i2pd-2.48.0/tunnels.conf /var/lib/i2pd/; sudo cp -R /usr/doc/i2pd-2.48.0/certificates /var/lib/i2pd/

Modify the ownership of the relevant folders and files:

sudo chown -R i2p:i2p /var/log/i2pd; sudo chown -R i2p:i2p /var/lib/i2pd; sudo chown -R i2p:i2p /var/run/i2pd; sudo chown i2p:i2p /usr/bin/i2pd; sudo chown i2p:i2p /etc/i2pd.conf

And the permissions (no need to give any other user access at all):

sudo chmod 700 -R /var/log/i2pd; sudo chmod 700 -R /var/lib/i2pd; sudo chmod 700 -R /var/run/i2pd; sudo chmod 700 /etc/i2pd.conf; sudo chmod 700 /usr/bin/i2pd

This will only allow us to visit other I2P sites, but of course we want our own. To accomplish that, edit the file */var/lib/i2pd/tunnels.conf* and add this section:

[WEBSITE]
type = http

host = 127.0.0.1
port = 80
keys = site-keys.dat

Now switch to the user that's going to run the daemon:

sudo su i2p

And finally run it:

i2pd --conf /etc/i2pd.conf --datadir /var/lib/i2pd/ --certsdir /var/lib/i2pd/certificates/ --service --daemon

To find out the address of your newly created I2P access point, enter the I2P console with eg *"lynx 127.0.0.1:7070"* and keep pressing the down arrow until you reach the menu option *I2P tunnels*, then press Enter. There you will find something like *WEBSITE => ius6h6cdf6rguuw65aktnpofjm5cn2my74s26ak7k5exgpk42odq.b32.i2p:80*. That's your address that you can put into your browser's address bar, if it's configured to use I2P as the proxy - or just share it with others (can skip the 80, and remember to add http:// at the beginning).

Getting an SSL certificate

But that's not very useful alone. Anyone in between your reader and your server potentially gets to view - or even modify - the connection traffic. Defending from this is possible by getting a HTTPS certificate. To do so, install acme.sh (*"slapt-get --install acme.sh"*). Then issue a cert with this command (you need a domain first, with an A record pointing to the server IP):

acme.sh --issue --domain schooling.exposed --accountkeylength 8192 --standalone

You will get a message similar to this when it finishes:

Your cert is in: /root/.acme.sh/schooling.exposed_ecc/schooling.exposed.cer
Your cert key is in: /root/.acme.sh/schooling.exposed_ecc/schooling.exposed.key
The intermediate CA cert is in: /root/.acme.sh/schooling.exposed_ecc/ca.cer
And the full-chain cert is in: /root/.acme.sh/schooling.exposed_ecc/fullchain.cer

It is also possible to generate a self-signed certificate instead of requesting it from a cert authority, but it's a lot more involved and will also bring up errors in the viewers' browsers (though the security level is the same), so we will skip it for now. Anyway - because darkhttpd doesn't support SSL certificates by default (hey, I said it's lean...) - we need an additional piece of software called stunnel; so install it by typing *"slapt-get --install stunnel"*. As is tradition, we will make it run as a separate user:

groupadd stunnel

useradd --no-user-group --gid stunnel stunnel -s /bin/bash -d /home/stunnel

Now move the previously created certificates to stunnel's folder so it can pick them up:

cp /root/.acme.sh/schooling.exposed_ecc/fullchain.cer /etc/stunnel
cp /root/.acme.sh/schooling.exposed_ecc/schooling.exposed.key /etc/stunnel

Stunnel needs a folder to put its pid file in, so create it:

mkdir /var/run/stunnel
chown -R stunnel /var/run/stunnel
chmod 755 -R /var/run/stunnel

By default, stunnel has a sample config file at */etc/stunnel/stunnel.conf-sample*, but it's not being used as it's the "wrong" name, so change it to stunnel.conf (*"mv /etc/stunnel/stunnel.conf-sample /etc/stunnel/stunnel.conf"*). Then edit it with nano or anything else. First, change the *";pid = /var/run/stunnel.pid"* part to *"pid = /var/run/stunnel/stunnel.pid"* (it won't work with the original folder due to permissions issues, I think). After that, put these in somewhere:

setuid = stunnel
setgid = stunnel

[https]
client = no
accept = 443
connect = 0.0.0.0:80
cert = /etc/stunnel/fullchain.cer
key = /etc/stunnel/schooling.exposed.key

The first two lines tell stunnel to run as the user stunnel. All others give SSL capabilities to your webserver (assuming it runs on port 80, which is the default for root-started darkhttpd). If you run stunnel, you should be able to connect to your website through https now.

Authority-assigned certificates don't stay valid forever, so you'll need to renew them with these commands every so often:

cd /root/.acme.sh/
sh acme.sh --renew -d schooling.exposed

And copy them to stunnel's folder with previously mentioned commands, then restart:

pkill stunnel; pkill darkhttpd

stunnel&
darkhttpd /var/www/mysite --addr 0.0.0.0 --uid darkhttpd --gid darkhttpd --daemon&

You can put all that in a cronjob at */etc/cron.monthly/renewcert.sh* for example. HTTP (or onion) connections will always work even with expired certs.

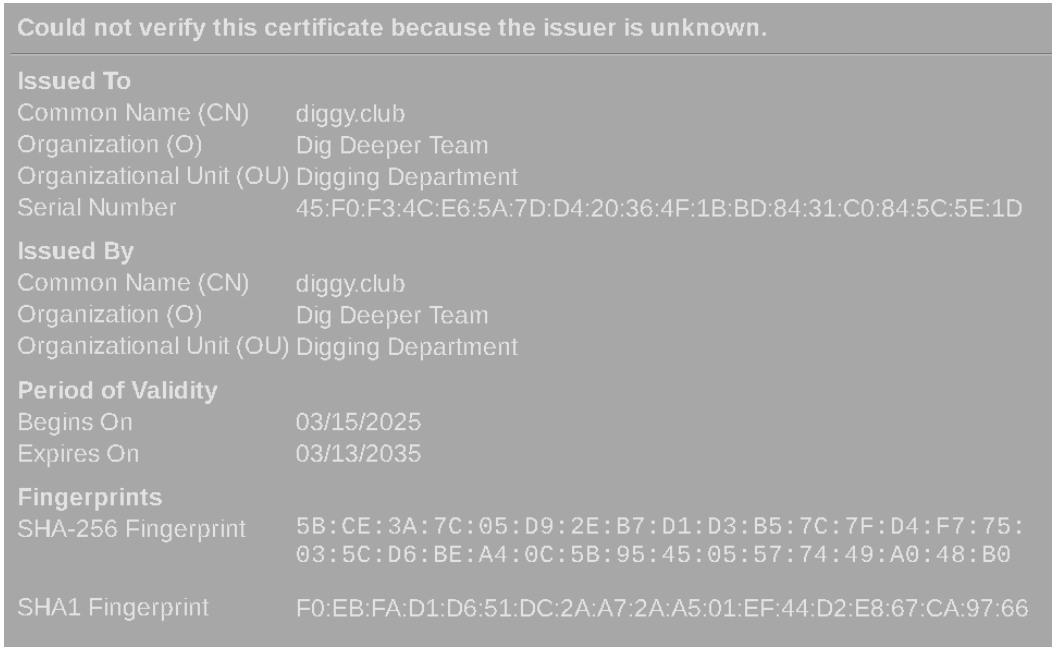
If you want to use stunnel with two (or more) domains, add this to the config file:

[school.sucks]
sni = https:school.sucks
connect = school.sucks:80
cert = /etc/stunnel/sucks_fullchain.cer
key = /etc/stunnel/sucks.key

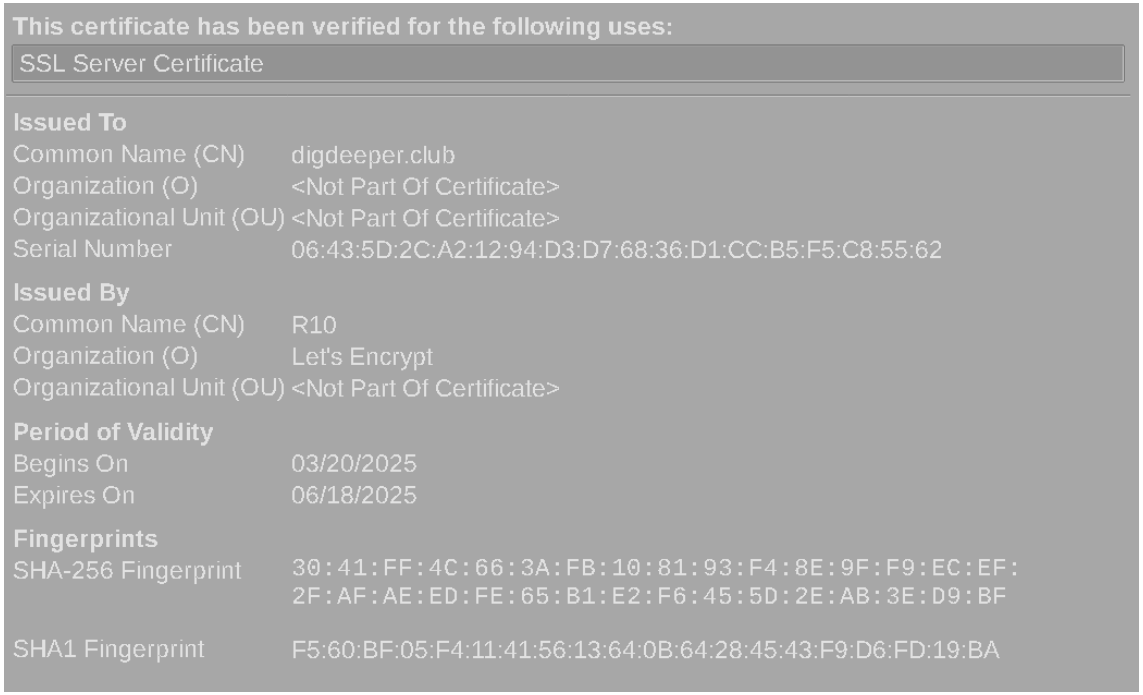
Of course, the domain has to exist, and point to the same server. You will also need to generate and move the keys for it, like earlier. And regenerate them when they expire, as usual. I tried to accomplish this through darkhttpd's *forward* options but they don't seem to work well. I wanted to redirect .club to .love so I wouldn't have to get another cert, but it either complains about non-matching cert or spits the "redirect forever" error in Pale Moon. I also tried to find options in stunnel that allow handling of unencrypted requests, but they don't seem to exist. No matter, we can just create the cert, and have it work this way.

Self-signed

You do not need to rely on a certificate authority to support encrypted connections - you can generate the cert yourself. This has some advantages in that you don't have to rely on a third party that can go down at any time, or just be hacked and compromised by "cybercriminals". ~~You can have stronger encryption than the maximum offered by acme / Let's Encrypt (8192 vs 4096)~~ not relevant anymore. You can have your cert be valid for whatever length of time you want to, instead of the puny month or 3 months or whatever of LE; you will never be canceled, either. And finally you can fill the data with funny stuff like this:

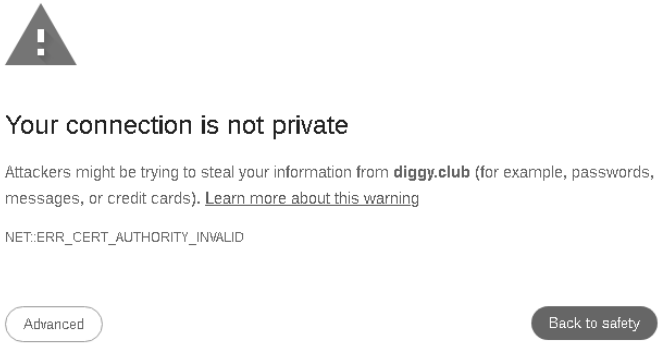


Compare to a Let's Encrypt-assigned one:



So what's the catch? Because there has to be one, right? There indeed is, and a very serious one. Namely that **all mainstream browsers**

display a big scary warning when they encounter a self-signed cert. Like this:



You can click *Advanced* and proceed to my site anyway, but every normie will retreat to his "safe" Google-shaped cage instead. So if you want any actual viewership, you can't rely on this at least unless you have another host with an "official" cert. Which is the way I recommend it - one LE host for the normies, one self-signed for the tech nerds. Anyway, I've rambled enough; let's proceed to generating a self signed certificate:

> *cd /etc/stunnel (this is so that the certs will be put inside the required folders after being generated)*
> *openssl genrsa -des3 -out selfsignedcert.key 8192*
> *(now choose a strong password)*
> *openssl req -key selfsignedcert.key -new -out selfsignedcert.csr*
> *openssl x509 -signkey selfsignedcert.key -in selfsignedcert.csr -req -days 3650 -out selfsignedcert.crt*
> *(3650 is the length of time in days the cert is going to be valid for)*
> *(now comes the fun part: filling your cert with information that will be shown inside the "View certificate" screens in browsers. Be careful because you can't go back! But you can put whatever you want in there, really)*

Finally, you'll need to edit */etc/stunnel/stunnel.conf* to point to the newly created certs:

> *[https]*
> *client = no*
> *accept = 443*
> *connect = 0.0.0.0:80*
> *cert = /etc/stunnel/selfsignedcert.crt*
> *key = /etc/stunnel/selfsignedcert.key*

You can still leave the old certs in the folder, just in case you want to switch back. Restart stunnel to apply, of course.

Uploading files

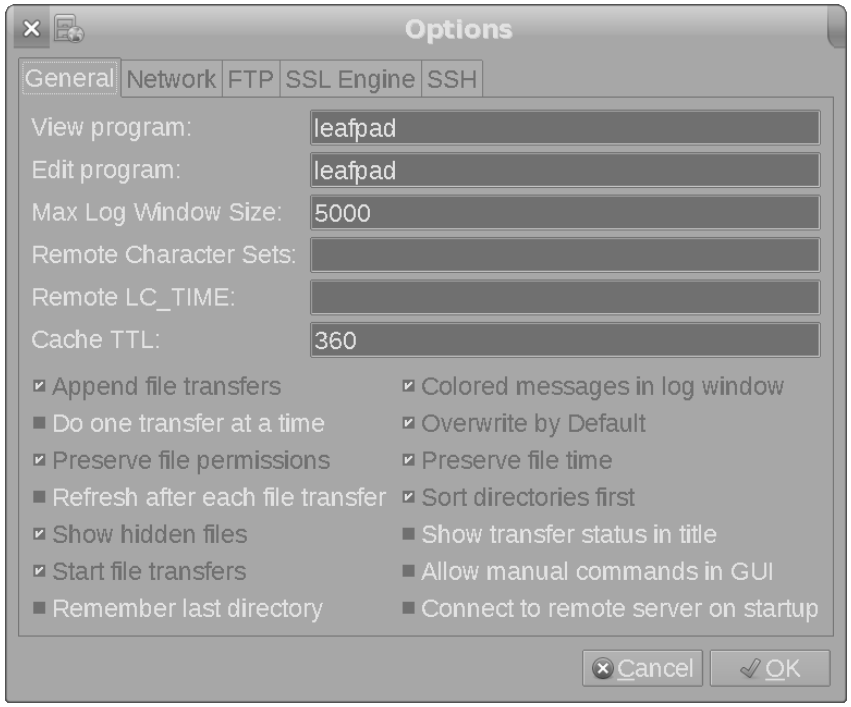
Let's finally get something up on that server, shall we? My favorite way of doing so is *gFTP* - it's graphical, it's easy, and does everything needed. So install it **on your local machine** (not the server!) by typing:

slapt-get --install gftp

Then run it with:

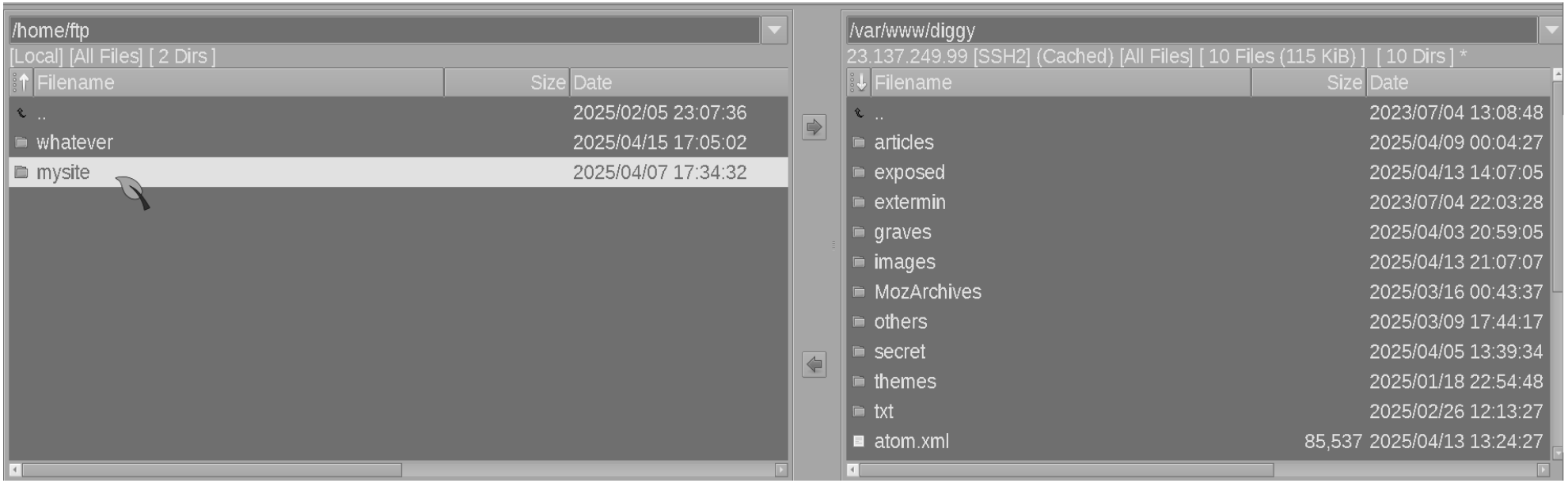
proxychains4 gftp

Proxychains needs to be installed and configured to use TOR - I've shown how to do this here. This is essential because you really don't want to reveal your real IP to your VPS; even using a VPN doesn't seem like enough when a file you upload there could get you busted by feds. In my proxychains guide I also explain how to make programs always run through it, with a specifically setup graphical icon in the panel. Anyway - after turning it on - go to *gFTP* -> *Preferences* and set them like this:

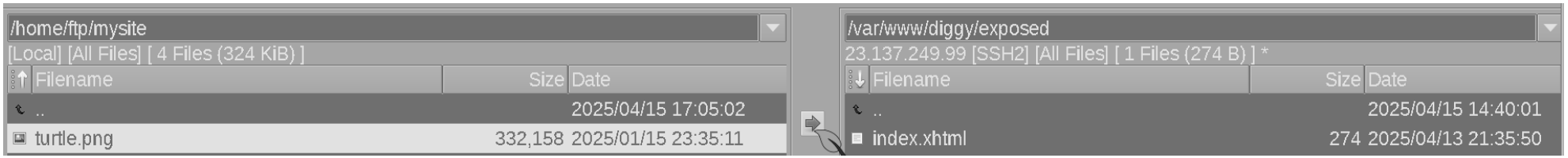


You can use whichever editing program you want, of course (it won't be relevant often if you do it like me anyway). The other options I think are just the most sane ones; you can hover your cursor over them if you want to know what they do exactly. Click *OK* and fill the upper form fields like this:

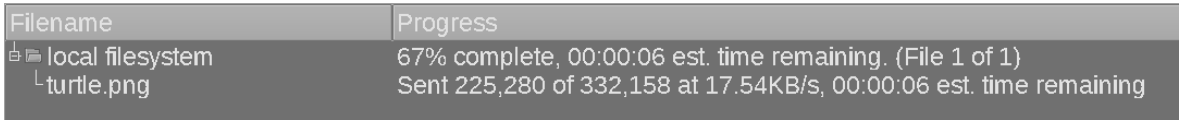
Of course, they have to correspond to what you really have on the server. Anyway, click the "two computers" button and you should end up connected:



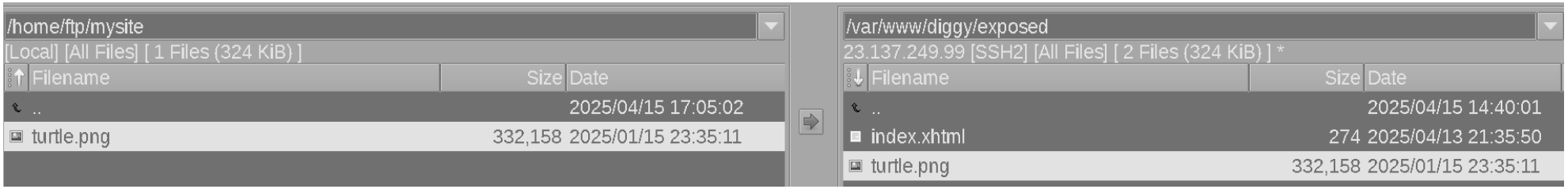
Left side contains the folders on your computer, right side - the ones on your VPS. Double click the folder you want to enter, on both sides. Then single-click the file you want to upload to your VPS (this selects it), and finally begin the transfer by clicking the arrow button:



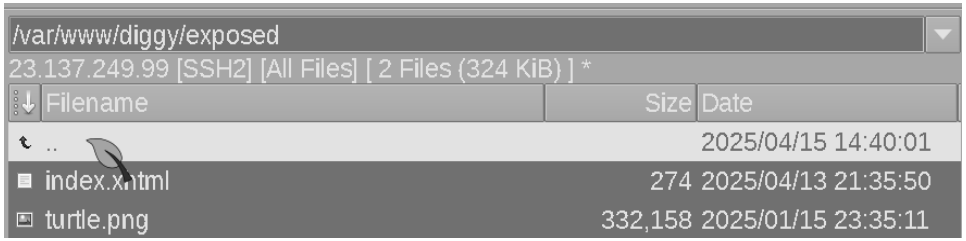
Notice how the paths in the path fields changed (you can also modify them directly if your files are stored somewhere else). Anyway - while the file is transfering - this will be displayed below the folder listings:



When it finishes, gFTP will automatically refresh and you will see the file on the other side:



Meaning you can access it at https://digdeeper.club/exposed/turtle.png. To go back (on either side), double click the line with the black arrow (no need to aim at it directly):



What else can gFTP do? Basically anything you'd want to while managing a site. Deleting files by right mouse button+delete. Mass transfer by shift+left mouse button, or ctrl+LMB if you want to skip some files. Editing remotely though I recommend doing it locally, then uploading - much more convenient and organized because you know you always have the same version everywhere. Renaming with RMB+rename, transfering back to your local computer with the reverse arrow nearer to the bottom (so you can use your VPS as an easy secondary backup, though space is very limited). Creating new folders, modifying permissions (including en masse by shift+LMB, like with deletions or transfers). In general, gFTP is faster and much less annoying than fiddling with the command line, which it pretty much deprecates (in terms of site management only, and not the myriad of other things you might want to do on the server, of course).

TBD: tor access, who knows what else...

Back to the front page