

Legwork

- [Initial Steps](#)
- [Initial Login](#)
- [Jail Preparation/Setup](#)

Initial Steps

Domain

It was high time that I "owned" part of the internet, so I went on NameCheap.com and "bought" two URLs. I went with the WhoisGuard feature, and I set the domain registrations for auto-renewal.

VPS

I then registered w/ DigitalOcean and provided my credit card. Now I am ready to roll.

I started a new project on DigitalOcean, and created a new droplet. I selected FreeBSD 12.1 w/ ZFS (could also choose 11.3, and could choose UFS or ZFS, but ZFS is great, and I know I'll wish I had it once I get a little more advanced). I went with the \$5/mo plan (1 vCPU, 1 GB RAM, 25GB SSD). I was not provided a root password. Instead, I had to provide an SSH pubkey. All the better.

VPS Setup

It would be wise to pop into the Networking tab and create a basic firewall for the droplet. I chose to accept `https`, `http`, and `SSH`, all on default ports. FreeBSD doesn't have a firewall running out the box, so this step is probably wise.

Domain Setup

Shortly after creating the droplet, I brought the newly acquired IP address over to NameCheap and created A records.

- Created an A record for the domain where the host is simply `@`, meaning `mydomain.tld` will work as is.
- Created an A record for the domain where the host is `www`, so www.mydomain.tld will also work.
- Left in place, for now, the CNAME record to the NameCheap parking page. (I should have probably deleted right away.)
- Also left in place, for now at least, the URL redirect for the host `@` to www.mydomain.tld. I need to research this.

All Done

Well, not so fast. DigitalOcean and NameCheap both have the ability to enable 2FA. Maybe you should get that out of the way before you get too far. I did.

It's almost time to get various jails set up and piped into a reverse proxy (another jail), and then have CNAME records created for pointing subdomains (a subdomain for each jail) at the droplet's IP. You do the CNAME records in the Advanced DNS screen of the NameCheap site.

At this point, there's not much left to do but log in to the server. I tried ssh root@mydomain.tld right away, and it didn't work, and I had to use the IP address. But after a little while, it worked fine. DNS is fun, right?

Initial Login

Upon SSH'ing into the server, I am greeted w/ a cold black and white terminal. Excellent.

Housekeeping

Before moving forward, let's get the system up to date and grab a few packages we'll need momentarily.

```
freebsd-update fetch install
```

```
pkg update && pkg upgrade -y
```

```
pkg install -y git nano tmux vim-console lsblk
```

I was tempting to install `curl` and `wget`, but those are covered with `fetch`, which is part of the FreeBSD base system.

Get the clock up to date, just in case it's not.

```
tzsetup
```

For the host, you'll probably want to make sure you select the timezone where the VPS is located.

Enable the time service.

```
sysrc ntpd_enable="YES"
```

```
sysrc ntpd_sync_on_start="YES"
```

And then might as well start it now.

```
service ntpd start
```

What are we working with here (supposedly a 25G SSD, right?)... `lsblk` reveals:

DEVICE	MAJ:MIN	SIZE	TYPE	LABEL	MOUNT
vtbd0	0:69	25G	GPT	-	-
vtbd0p1	0:71	256K	freebsd-boot	gptid/fe84c375-529c-11ea-...	-
vtbd0p2	0:72	2.0G	freebsd-swap	gpt/swap0	-
vtbd0p3	0:89	23G	freebsd-zfs	gpt/disk0	<ZFS>

Running `swapinfo -h` confirms the swap space.

Quality of Life

I prefer to freshen up the `.cshrc` file. Here's the original.

```
# $FreeBSD: releng/12.1/bin/csh/dot.cshrc 338374 2018-08-29 16:59:19Z brd $
#
# .cshrc - csh resource script, read at beginning of execution by each shell
#
# see also csh(1), environ(7).
# more examples available at /usr/share/examples/csh/
#

alias h      history 25
alias j      jobs -l
alias la     ls -aF
alias lf     ls -FA
alias ll     ls -lAF

# A righteous umask
umask 22

set path = (/sbin /bin /usr/sbin /usr/bin /usr/local/sbin /usr/local/bin $HOME/bin)

setenv EDITOR vi
setenv PAGER less
setenv BLOCKSIZE K

if ($?prompt) then
    # An interactive shell -- set some stuff up
    set prompt = "%N@%m:%~ %# "
```

```

set promptchars = "%#"

set filec
set history = 1000
set savehist = (1000 merge)
set autolist = ambiguous
# Use history to aid expansion
set autoexpand
set autorehash
set mail = (/var/mail/$USER)
if ( $?tcsh ) then
    bindkey "^W" backward-delete-word
    bindkey -k up history-search-backward
    bindkey -k down history-search-forward
endif

endif

```

And then I added some things, mostly inside the `if` block:

```

if ($?prompt) then

    #set prompt = "%N@%m:%~ %# " # comment this out, and replace it below

    # colors for prompt (0 for regular and 1 for bold, or use %B%b for bold)
    set red="%{\033[0;31m%}"
    set green="%{\033[0;32m%}"
    set yellow="%{\033[0;33m%}"
    set blue="%{\033[0;34m%}"
    set magenta="%{\033[0;35m%}"
    set cyan="%{\033[0;36m%}"
    set white="%{\033[0;37m%}"
    set end="%{\033[0m%}" # This is needed at the end... :(

    # Set username color. (Preferably, unique)
    if ( $USER == "root" ) then
        set user_color="${yellow}"
    else
        set user_color="${magenta}"
    endif

```

```
# prompt vars
set name = "${user_color}%B%n%b${end}"
set host = "${cyan}%m${end}"
set dir = "${red}%~${end}"

set prompt = "[${name}@${host}:${dir}]%# "

set complete = enhance

# Clean up...
unset red green yellow blue magenta cyan white end
unset name host dir

endif

# color in autocomplete
set color
# color in ls
alias ls      ls -G

# BSD colors for ls.  See https://geoff.greer.fm/lscolors/
setenv LSCOLORS gxfxcxdxbxegedabagacad
```

Turn off the mail notifications

Add the following to `/etc/rc.conf`, whether pasting or by `sysrc`.

```
sendmail_enable="NO"
sendmail_submit_enable="NO"
sendmail_outbound_enable="NO"
sendmail_msp_queue_enable="NO"
```

Security

More remote user access

I've got two local computers. I pasted my other comp's `SSH` pubkey in a shared file where this comp could copy it, and then I pasted it into `.ssh/authorized_keys`. Now both computers can `SSH` in.

Sweet.

SSH authentication

We may want to clean up the `/etc/ssh/sshd_config` by disallowing password logins. That particular setting is already set to 'no' but there are other things, like `ChallengeResponseAuthentication` and `UsePAM` which could also be set to 'no' (for a belt-and-suspenders approach, as they say).

System users

The default user is the root user. To continue the belt-and-suspenders approach theme, we could create a user for me (in the wheel group and operator group) and then disable root `SSH` login as well. Smart. But I'm skipping that.

Root password

More belt-and-suspenders - we could/should add a root password. Very smart. Why am I skipping this?

Obscurity

We could change the default listening port. This will decrease the noise, but is it really worth it? I mean, once the server is set up, you can just block SSH via the DigitalOcean firewall. It's like 2FA in that regard... sorta. Also inconvenient though.

Blocking bots

We could/should install Fail2Ban. Maybe later. For now (a bit below), we'll implement basic, built-in brute force protection.

Firewall

Here's how we're starting off with `/etc/pf.conf`:

```
# macros -> tables -> options -> traffic normalization -> queueing -> translation (NAT) -> packet filtering

## MACROS

# the external network interface to the internet
ext_if="vtnet0"
ext_if_ip="YOUR.NEW.IP.HERE"

# port on which sshd is running
ssh_port = "22"
```



```
# allowed inbound ports (host services)
# don't include ports that'll be redirected to jails
#inbound_tcp_services = "{auth, " $ssh_port " }"
#inbound_udp_services = "{dhcpv6-client}"
web_services = "{http, https}"

# jail IP addresses
#caddy_ip = "10.101.10.100"
#bookstack_ip = "10.101.10.110"

## TABLES

table <bruteforce> persist
table <webcrawlers> persist
table <rfc6890> { 0.0.0.0/8 10.0.0.0/8 100.64.0.0/10 127.0.0.0/8 169.254.0.0/16 \
172.16.0.0/12 192.0.0.0/24 192.0.0.0/29 192.0.2.0/24 192.88.99.0/24 \
192.168.0.0/16 198.18.0.0/15 198.51.100.0/24 203.0.113.0/24 \
240.0.0.0/4 255.255.255.255/32 }
# jails table is used by Bastille
table <jails> persist

## OPTIONS

# politely send TCP RST for blocked packets. The alternative is
# "set block-policy drop", which will cause clients to wait for a timeout
# before giving up.
set block-policy return

# log only on the external interface
set loginterface $ext_if

# skip all filtering on localhost
set skip on lo

## TRAFFIC NORMALIZATION

# reassemble all fragmented packets before filtering them
scrub in on $ext_if all fragment reassemble max-mss 1440
```

TRANSLATION

NAT out jail traffic

nat pass on \$ext_if from <jails> to any -> \$ext_if_ip

static port forwarding for http/https traffic to [reverse proxy] jail

#rdr pass inet proto tcp from any to any port {80, 443} -> \$caddy_ip

PACKET FILTERING

block forged client IPs (such as private addresses from WAN interface)

antispoof quick for \$ext_if

skip rfc6890 on external interface

block in quick on egress from <rfc6890>

block return out quick on egress to <rfc6890>

default behavior: block all traffic

block all

allow all icmp traffic (like ping)

pass quick on \$ext_if proto icmp

pass quick on \$ext_if proto icmp6

special pass rules for SSH

pass in quick on \$ext_if proto tcp to port \$ssh_port \

keep state (max-src-conn 6, max-src-conn-rate 4/10, \

overload <bruteforce>)

special pass rules for http/https (NOT SURE ABOUT THIS W/ REVERSE PROXY)

#pass in on \$vtnet0 proto tcp to port { 80 443 } \

keep state (max-src-conn 45, max-src-conn-rate 9/1, \

overload <webcrawlers> flush global)

allow incoming traffic to services hosted by this machine (to the host; not jails)

#pass in quick on \$ext_if proto tcp to port \$inbound_tcp_services

#pass in quick on \$ext_if proto udp to port \$inbound_udp_services

allow all outgoing traffic

pass out quick on \$ext_if

There's some nice foreshadowing in there.

There's a lot going on in that file. Here's a quick rundown:

1. Macros are for expanding into the rules below. Additionally, they make it so you can have rules that don't change between systems because you just have to update the macros for the system-specific variables.
2. Some of the macros are commented out for now. We may add them later. For now, this is all we need.
3. A table is kind of like a macro, but not really. For one, they don't expand out in the shell's variable substitution (which you'll see with `pfctl -vnf /etc/pf.conf`). They behave that way though, where a rule applies to every IP address or range in the table. The `bruteforce` and `webcrawler` tables are for keeping track of IP's that abuse the server. The `rfc6890` table is for IP's that we should never encounter anyway. The `jails` table is populated by Bastille (the jail/container manager we'll be using, which we'll talk more about later), and it contains all the Bastille jail IP's.
4. The options generally are already commented (not commented out... literally commented, so go read them).
5. The NAT rule (in combination with `gateway_enable` in `/etc/rc.conf`) allows jail traffic to pass out through the external interface and appear to come from the server's IP rather than the otherwise un-routeable jail's IP.
6. The rest is either commented, self-explanatory (perhaps with research), or commented on below.

Double check that the syntax is good and appears correct.

```
pfctl -vnf /etc/pf.conf
```

Assuming no errors, now it can be enabled.

```
sysrc pf_enable="YES"
```

And then, finally, PF can be started, which will boot us from our `SSH` session(s).

```
service pf start
```

It is wise to enable logging too. Besides, we'd need to parse the logs for Fail2Ban. (Can it parse binary logs?)

```
sysrc pflog_enable="YES"
```

The data in the `pflog` file is written in binary, so one way to view it is:

```
tcpdump -ner /var/log/pflog
```

May also want to install `pftop`.

PF Tables

We're getting brute force protection w/ a `bruteforce` table. This is already included above.

```
table <bruteforce> persist
table <webcrawlers> persist
table <rfc6890> { 0.0.0.0/8 10.0.0.0/8 100.64.0.0/10 127.0.0.0/8 169.254.0.0/16 \
172.16.0.0/12 192.0.0.0/24 192.0.0.0/29 192.0.2.0/24 192.88.99.0/24 \
192.168.0.0/16 198.18.0.0/15 198.51.100.0/24 203.0.113.0/24 \
240.0.0.0/4 255.255.255.255/32 }

{...}

# skip rfc6890 on external interface
block in quick on egress from <rfc6890>
block return out quick on egress to <rfc6890>

{...}

# special pass rules for SSH
pass in quick on $ext_if proto tcp to port $ssh_port \
keep state (max-src-conn 6, max-src-conn-rate 4/10, \
overload <bruteforce> )
# the very end could have 'flush global', but I'd risk ending my
# own connection if I accidentally triggered it

# could do a similar rule for http/https using <webcrawlers>
```

We'll periodically empty/clear the `bruteforce` table (since brute forcing tends to not happen from the same IP for long periods of time), so we'll write a shell script to do this for us.

```
vim /usr/local/bin/clear_overload.sh
```

In the script, we'll keep the IP's that have been around for 2 weeks or less, and flush/expire all else.

```
#!/bin/sh

pfctl -t bruteforce -T expire 1209600
```

It needs to be executable.

```
chmod 755 /usr/local/bin/clear_overload.sh
```

And then we put it in a `cron` job to run every first day of the week on the zeroth minute of 2am (with `crontab -e`).

```
# minute hour mday month wday command

0 2 * * 1 /usr/local/bin/clear_overload.sh
```

Other Config

It's worth looking at the `/etc/rc.conf`. It comes like this:

```
hostname="freebsd-hostname"
cloudinit_enable="YES"
sshd_enable="YES"
ifconfig_vtnet0="DHCP"
digitaloceanpre="YES"
digitalocean="YES"
zfs_enable="YES"

# DigitalOcean Dynamic Configuration lines and the immediate line below it,
# are removed each boot.

# DigitalOcean Dynamic Configuration
defaultrouter="YOUR.NEW.IP.HERE"
# DigitalOcean Dynamic Configuration
ifconfig_vtnet0="inet YOUR.NEW.IP.HERE netmask 255.255.240.0"
# DigitalOcean Dynamic Configuration
ifconfig_vtnet0_alias0="inet 10.10.0.5 netmask 255.255.0.0"
# DigitalOcean Dynamic Configuration
ifconfig_vtnet0_ipv6="inet6 2604:A880:0400:00D0:0000:0000:1B07:F001 prefixlen 64"
# DigitalOcean Dynamic Configuration
ipv6_defaultrouter="2604:A880:0400:00D0:0000:0000:0000:0001"
# DigitalOcean Dynamic Configuration
ipv6_activate_all_interfaces="yes"
```

And we have added:

```
ntpd_enable="YES"
ntpd_sync_on_start="YES"

sendmail_enable="NO"
sendmail_submit_enable="NO"
sendmail_outbound_enable="NO"
sendmail_msp_queue_enable="NO"

pf_enable="YES"
pflog_enable="YES"
```

And we'll soon be adding (more foreshadowing!):

```
bastille_enable="YES"
cloned_interfaces="lo1"
ifconfig_lo1_name="bastille0"
gateway_enable="YES"
```

Bonus Round

I've been using `vim` more than usual during this process.

1. It would be helpful to begin to put together a config file (`.vimrc`).
2. It would be wise to add it to my `custom_cshrc.sh` script, so I have it in jails too. (More foreshadowing!)

Anywoo, this is still a todo.

Jail Preparation/Setup

Filesystem

We'll want a dataset to store data that will exist outside the jails. Yay for ZFS (for reasons I'm glossing over...).

```
zfs create -o compress=lz4 -o atime=off -o mountpoint=/usr/local/data zroot/data
```

(Did I have to create `/usr/local/data` before doing the above? I don't recall... but I'm pretty sure no.)

We expect to have a BookStack jail, which has a database.

```
zfs create -o compress=lz4 -o atime=off zroot/data/dbs
```

And we can have a dataset for the BookStack db, specifically.

```
zfs create -o compress=lz4 -o atime=off zroot/data/dbs/bookstack
```

We'll get to this later, but now (later) we can `nullfs`-mount the dataset inside the jail (in its `fstab`) like so:

#	Device	Mountpoint	FStype	Options	Dump	Pass#
	/usr/local/data/dbs/bookstack	/usr/local/bastille/jails/bs_jail/root/var/db/mysql	nullfs	rw,late	0	0

And in case you jump ahead, you'll also need to know that `mysql` needs to own the directory.

```
cd /usr/local/data/dbs/  
chown -R 88:88 bookstack/
```

Speaking of jumping ahead... I had issues when using a newer version of MariaDB a few months later. The MariaDB setup might need more to get it working. It may have to do with changes to Mysql.

Jail Management

It's tempting to manage jails by hand, but I'll leave that exercise to my local server. We'll use this script-based tool instead.

```
pkg install bastille
```

We want the Bastille jails to start up upon system reboot, so we add it to the `rc.conf` file.

```
sysrc bastille_enable=YES
```

And we can hop right into the Bastille configuration that defines the jails' default parameters.

```
vim /usr/local/etc/bastille/bastille.conf
```

The notable changes are:

```
bastille_zfs_enable="YES"
bastille_zfs_zpool="zroot"

bastille_jail_addr="10.101.10.10" # not sure if this is even used or makes sense
```

Something of note is it uses a particular loopback device that must be created (added to `rc.conf`).

```
sysrc cloned_interfaces+=lo1
```

```
sysrc ifconfig_lo1_name="bastille0"
```

And since the jails are on a separate loopback network and need to be NAT'd, we probably need this:

```
sysrc gateway_enable="YES"
```

And then the cloned interface can be brought up.

```
service netif cloneup
```

We would then update `pf.conf` accordingly to allow jail traffic if the example we started with didn't already have this.

```
table <jails> persist

{...}
```



```
nat on $ext_if from <jails> to any -> $ext_if_ip
```

```
## static port forwarding for sending http/https to [reverse proxy] jail
```

```
## rdr pass inet proto tcp from any to any port {80, 443} -> 10.17.89.45
```

When searching online, you may find that NAT rules end with `-> ($ext_if)`, but that will include all aliases, which will make the NAT-ting most likely not behave as intended. You want to NAT on the external IP (`-> $ext_if_ip`).

Base Jail

(For creating quickly update-able thin jails later)

It's pretty simple to create the base jail. This will download a fresh base install, basically.

```
bastille bootstrap 12.1-RELEASE
```

You'll want to occasionally update this with:

```
bastille bootstrap 12.1-RELEASE update
```

We should now be ready to create jails.

(On a 25G instance, `ZFS list` currently reflects there is 16.8G remaining space.)

Initial Jail Creation

More foreshadowing!

```
Usage: bastille create [option] name release ip [interface].
```

`Options` - Empty, Thick, VNET (none of these)

`Interface` - vnet (no!), bastille0 (yes, but implied)

```
bastille create caddy_jail 12.1-RELEASE 10.101.10.100
```

```
bastille create bs_jail 12.1-RELEASE 10.101.10.110
```

```
bastille create bw_jail 12.1-RELEASE 10.101.10.120
```

```
bastille create thelounge_jail 12.1-RELEASE 10.101.10.130
```

```
bastille create website_jail 12.1-RELEASE 10.101.10.140
```

And after creating five thin jails, the remaining space is still 16.8G. Yay, ZFS, again!

Quick quality of life improvement in the jails

Let's create a `.cshrc` for copying into the jails. It's the same as the regular one, but it uses different prompt colors.

```
# $FreeBSD: releng/12.1/bin/csh/dot.cshrc 338374 2018-08-29 16:59:19Z brd $
#
# .cshrc - csh resource script, read at beginning of execution by each shell
#
# see also csh(1), environ(7).
# more examples available at /usr/share/examples/csh/
#

alias h      history 25
alias j      jobs -l
alias la     ls -aF
alias lf     ls -FA
alias ll     ls -lA

# A righteous umask
umask 22

set path = (/sbin /bin /usr/sbin /usr/bin /usr/local/sbin /usr/local/bin $HOME/bin)

setenv EDITOR vi
setenv PAGER less
setenv BLOCKSIZE K

if ($?prompt) then
    # An interactive shell -- set some stuff up
```

```
# colors for prompt (0 for regular and 1 for bold, or use %B%b for bold)
set red="%{\033[0;31m%}"
set green="%{\033[0;32m%}"
set yellow="%{\033[0;33m%}"
set blue="%{\033[0;34m%}"
set magenta="%{\033[0;35m%}"
set cyan="%{\033[0;36m%}"
set white="%{\033[0;37m%}"
set end="%{\033[0m%}" # This is needed at the end... :(

# prompt vars
set name = "${red}%B%n%b${end}"
set host = "${red}%m${end}"
set dir = "${cyan}%~${end}"

set prompt = "[${name}@${host}:${dir}]%# "

#set prompt = "%N@%m:%~ %# "
set promptchars = "%#"

set complete = enhance

set filec
set history = 1000
set savehist = (1000 merge)
set autolist = ambiguous
# Use history to aid expansion
set autoexpand
set autorehash
set mail = (/var/mail/$USER)
if ( $?tcsh ) then
    bindkey "^W" backward-delete-word
    bindkey -k up history-search-backward
    bindkey -k down history-search-forward
endif

# Clean up...
unset red green yellow blue magenta cyan white end
unset name host dir
```

```
endif

# color in autocomplete
set color
# color in ls
alias ls      ls -G

# LS colors, made with https://geoff.greer.fm/lscolors/
setenv LSCOLORS      gxfxcxdxbxegedabagacad
```

Then `mv` each jail's `.cshrc` as `.cshrc.orig`, and then `cp` the `.cshrc.jail` as each jail's new `/root/root/.cshrc`. See below for a script to do this quickly and easily.

Other Bits

It may be a good time to reboot the server. You've made several changes to the system, and you'll want to make sure they stuck and are working correctly.

Changes to `/etc/pf.conf` require `pfctl -f /etc/pf.conf` **. Changes to `/etc/rc.conf` require... something. Changing the `.cshrc` requires sourcing it or logging in fresh. The jails need to be started. Rebooting will do all this, including starting the jails.

** Just make sure you at least have already run `pfctl -vnf /etc/pf.conf` to make sure the config works.

Common Initial Jail Setup

The beginning steps are mostly the same across the jails. Before jumping in, if you haven't already, remember to `mv` the jail's `.cshrc` as `.cshrc.orig`, and then `cp` the host's `.cshrc.jail` as the jail's new `/root/root/.cshrc`.

In fact, here's a script (that magically worked perfectly the first time I ran it), that I just saved in `/usr/local/scripts`.

```
#!/bin/sh

# Copies custom .cshrc from /root/.cshrc.jail in place of the
# jail's default .cshrc, and renames the default as .cshrc.orig.
```

```
# Exit script if error (non-zero return code)
set -e

# check for a single arg (the name of the jail)
if [ "$#" -ne 1 ]; then
    echo "Usage: $0 JAIL_NAME" >&2
    exit 1
fi

# Variables to be used
jail_name="$1"
jails_dir="/usr/local/bastille/jails"
jail_dir="${jails_dir}/${jail_name}/root/root"

# check that the directory exists
if [ ! -d "${jail_dir}" ]; then
    echo "Directory ${jail_dir} doesn't exist." >&2
    exit 1
fi

# check that the original .cshrc exists
if [ ! -f "${jail_dir}/.cshrc" ]; then
    echo "File ${jail_dir}/.cshrc doesn't exist." >&2
    exit 1
fi

# check that the custom .cshrc exists
if [ ! -f "/root/.cshrc.jail" ]; then
    echo "Custom .cshrc.jail in /root doesn't exist." >&2
    exit 1
fi

mv ${jail_dir}/.cshrc ${jail_dir}/.cshrc.orig

cp /root/.cshrc.jail ${jail_dir}/.cshrc

# Write to log briefly what happened
echo "Added custom .cshrc to ${jail_name}."

exit 0
```

Don't forget to `chmod +x` it. Then you just run it with `/usr/local/scripts/custom_cshrc.sh <jail_name>`.

Misc

Another initial jail setup task may be to set up the timezone. You can (unlikely, but possible) have weird internet problems if your time is off. The host time being right is the most important, but feel free to check the current date and time with the `date` command. If you need to update things, run `tzsetup` and choose your timezone.

Also, you may update the jail. If you just created or updated your base jail, or if this is a thin jail, then there is actually no reason for this. But if you do need/want to do an update...

Updates cannot be installed when the system `securelevel` (`jail.conf` setting) is greater than zero.

So we must first edit `.../jails/$jail/jail.conf` to change `securelevel` from `2` to `0`, then restart the jail.

Then the updating can happen.

```
bastille cmd $jail freebsd-update fetch install
bastille pkg $jail update
bastille pkg $jail upgrade -y
```

Then we edit `.../jails/$jail/jail.conf` again to change `securelevel` from `0` to `2`, then restart the jail again.

And now you have a current, clean slate upon which to build.

Resources

The Bastille docs are great. <https://bastillebsd.org/>