How To Image FreeBSD Systems Using `tbku`

This document describes how to use the TundraWare Inc. `tbku` utility to "image" or "clone" FreeBSD systems.

> **Warning**
> What follows is a description of activities that can (and will) clobber the contents of a hard drive. Never do any of this until you understand what's going on fully. Obviously, you should have backups of whatever machine you're targeting so that if (when) you make a mistake, you can recover your data. YOU HAVE BEEN WARNED! If you proceed, you do so at your own risk ... and no, I will *not* come to your house and help you recover your hard drive.

Why Bother Imaging?

Suppose we need to build a new instance of a `FreeBSD` system. Perhaps we need to replace one that just had a hard drive failure. Maybe we want to build a new server that is based on our "standard" system configuration. In other words, we want to go from "bare metal" hardware to a fully running *and configured* system as quickly as possible.

There are a number of commercial and open source solutions to this problem, but they all have one thing in common: We want to minimize the amount of manual labor needed to install, configure, and otherwise customize the final system. This is especially important in large data centers where it is impractical to manually (re)install each and every server, its applications, and its customization information.

"Imaging" or "Cloning" allows us to keep a copy of the entire OS *as configured* - that means with all its applications and configuration options set up as desired. We then load or "Provision" a new hard drive with this image and *voila'*, "instant" running system.

When Does Imaging NOT Make Sense?

Imaging works best when the system you are targeting is very similar or identical to the system that made the image in the first place. For example, Imaging is a great way to restore a single machine from its own backups - say after a hard disk crash or upgrade.

Imaging is more complex when the source of the image and the target machines are different. The more different they are, the harder it will be to get the image running on the new target machine.

As a practical matter, production Data Centers tend to keep a separate restore image around *for each different system variant*. So, for example, you might find a separate image for IBM web servers, IBM applications servers, Dell database servers, Toshiba laptops, and so on.

Imaging may- or may not make sense when initially installing a new configuration. Say you have a system that is a web server, but you now want to build a separate machine that is a database server. Typically, you would initially install FreeBSD with

the installation disk, configure the database and *then* create a system image of your database server. However, this is kind of time consuming. It might simpler to image the target machine with your web server image, boot it, reconfigure it as a database server, and then take an image of your newly configured server for future installations.

What Is `tbku`?

`tbku` is a shell script that makes it easy to create tarballs of some of all of your filesystems. `tbku` does not help you with *restoring* your image, it's just handy for creating the image in the first place.

If you've never used it before, take a moment to download it and read the documentation. You'll find the latest copy at:

http://www.tundraware.com/Software/tbku

There is no fee for using `tbku` in any context, personal or commercial. However, there are some licensing terms you have to abide by to use it, so take a moment to read the license in the distribution tarball.

> **Note**
> You don't *have* to use `tbku` to create your backup image. The description below should work fine so long as you have a backup of all the relevant files that preserves all the appropriate file information such as ownership and permissions. `tbku` just makes it easy to automate the creation of such backups.

The Big Picture

Before diving into the details, it's good to get a sense of the overall process. Imaging a system requires the following steps:

A. Create the master image:

- Create a baseline system configured as you want it.
- Take an "image" of it. (That's where `tbku` is helpful.)
- Save the image somewhere (DVD, USB drive, network drive ...) you can get at when you need it to (re)install a system.

B. Use the master image to (re)provision a machine:

- Prepare the target hard disk to receive the image.
- Dump the image onto the hard disk.
- Adjust the configuration if/as needed for the new hardware.

Creating The Master Image

Unlike other approaches that make an image of *the disk*, `tbku` creates an image of *files* on the disk. This means that your new target disk does not have to be physically the same as the one on which the master image (sometimes called a "snapshot") was made. You can clone systems back and forth between SCSI, IDE, and SATA. You can clone from smaller disks to larger ones or go the other way.

> **Note**
>
> The whole point of imaging is to avoid having to do custom configuration for each new installation. However, some configuration changes may be necessary when the target environment or hardware is different than the system on which the master image was created. This is discussed a bit more below in the Gotchas section.

Creating The Master Image

1. Select the machine whose existing FreeBSD installation you want preserved or used as a standard installation image.
2. Image that system with `tbku` using the following fileset:

   ```
   /.cshrc
   /.profile
   /COPYRIGHT
   /bin
   /boot
   /compat
   /dist
   /etc
   /home
   /lib
   /libexec
   /rescue
   /root
   /sbin
   /sys
   /usr
   /var
   /www
   ```

   Notice that we do *not* backup the dynamic kernel-created filesystems like `/dev` or `/proc`, nor do we backup utility mountpoints like `/mnt` or `/tmp`.

   Also, if you have `tbku` writing your backup to the local disk, make sure that directory is *not* included in the fileset. Doing so would create a recursive backup wherein the backup would be copied to itself.

The exact fileset you use will vary somewhat depending on how you've laid out your directory tree and just what you want included in your image. Use the fileset above as a point of departure, and tune it for your exact needs.

3. Save the resulting `.tar.gz` (tarball) file somewhere it can be retrieved later when you want to image another machine. This can be a network server, a USB drive, a DVD or whatever makes sense in your environment. As with all backup systems, it's pretty important to make multiple copies of the backup image, and keep a couple of them off-site.

Provisioning With The Master Image

Now that we have a "snapshot" or master image, we can use it to (re)provision machines. The general idea here is to take advantage of the tools already present on the FreeBSD installation CD. However, instead of actually installing an operating system, we'll just use the paritioning and disk labeling tools to prepare the target disk to receive our FreeBSD image. Then, we'll jump into the `Fixit` shell and actually do the restore from there.

Provisioning Machines With A Master Image

1. Boot the FreeBSD installation disk.
2. Prepare the disk to receive a FreeBSD filesystem:

```
Custom

  Partition

    Select the target drive

    Partition as desired

    Select the partition that will boot
      S  - To make it bootable

    Quit

    Select the boot manager you want

Label

  Layout your partition(s) as desired
    The Automatic option is a good choice
    **WRITE DOWN THE DEVICE/MOUNT ASSIGNMENTS!**
      You'll need them later
```

```
W - Write the changes out (Answer "Yes" at the prompt)
```

```
Exit back to the main menu
```

At this point, your target disk has been partitioned, labeled, and had the Master Boot Record installed. The copy of FreeBSD you booted from CD is pretty smart about this. It has already mounted your mountpoints (the ones you wrote down above, right?) under its own `/mnt` directory. We'll take advantage of this in the next step.

3. Now we're ready to actually dump the image onto our newly prepared disk. The FreeBSD team helpfully provides a fairly complete shell environment where we can do what is needed. Simply select the `Fixit` option from the main menu, and the `CDROM/DVD` suboption, and you'll find yourself in a shell. You can prove that your new disk mountpoints are ready to be loaded, by doing this:

   ```
   mount
   ```

   You should see your newly created filesystems mounted under `/mnt`. Now, we need to create the top level directories that are typically not backed up in an image (some of these may already be present):

   ```
   cd /mnt
   mkdir cdrom dev dist proc tmp
   ```

   At this point, we need the image tarball file to go ahead and do the image restore. But ... we need to take a small detour here. You may need to do a few things to be able to *get* to your image. The `Fixit` environment is purposely fairly small (so it can run in a logical memory disk). As such, it does not have a lot of the utilties, kernel modules, and/or libraries you may need to get to your backup medium.

   Say, for example, your image is on as USB drive. You plug the drive in and take a look at `/var/log/messages` where you are informed that the drive has been recognized as `/dev/da1`. So, you try this:

   ```
   mount -t msdosfs /dev/da1s1 /mnt/mnt
   ```

   Oops ... the `Fixit` shell complains - it doesn't know how to mount filesystems of type `msdosfs` because the necessary file `/sbin/mount_msdosfs` file is not present in the `Fixit` operating environment.

   Fortunately, there's a very simple way to work around this. The CD from which you booted is itself mounted

under `/dist`. That CD has a more-or-less full "live" FreeBSD system on it, that *does* have the files you need there. In this case, the "fix" is to do something like this:

```
mkdir -p /sbin  # Make sure the directory exists in Fixit
cp -pv /dist/sbin/mount_msdosfs /sbin
```

Now the mount command above will work fine, and you can get to your backup tarball.

You can use a similar approach to get the necessary files for accessing your image via sftp, nfs, and other filesystem types. Just remember that, if you do need to use a network, you need to initialize it first with something like:

```
ifconfig <NIC device name> address mask
```

OK, so now we've mounted the medium with our image on it under `/mnt/mnt`. To actually image the new disk, all we have to do is this:

```
cd /mnt           # Make sure we're at the
                  # logical root of our disk

tar -xzvf mnt/my-fine-image.tar.gz
```

4. Finally, we need to make sure that our newly imaged filesystems will be mounted properly at boot time. This is controlled by the contents of: `/mnt/etc/fstab` Suppose, after we image the drive, that file looks like this:

```
/dev/ad4s1b     none      swap    sw         0   0
/dev/ad4s1a     /         ufs     rw         1   1
/dev/ad4s1d     /var      ufs     rw         2   2
/dev/ad4s1e     /usr      ufs     rw         2   2
/dev/acd0       /cdrom    cd9660  ro,noauto  0   0
```

This would indicate the image was taken from a system with FreeBSD installed on the first SATA drive. Now, lets assume we're going to use the same slice layout, but our newly imaged drive is the first *SCSI* drive on the system. `/mnt/etc/fstab` needs to be edited to look like this:

```
/dev/da0s1b     none      swap    sw         0   0
/dev/da0s1a     /         ufs     rw         1   1
/dev/da0s1d     /var      ufs     rw         2   2
/dev/da0s1e     /usr      ufs     rw         2   2
```

```
/dev/acd0       /cdrom    cd9660  ro,noauto 0    0
```

It may also be necessary to edit the `/mnt/etc/rc.conf` file to adjust IP address assignments or other system configuration parameters.

We're DONE! Well ... maybe. If the environment or hardware of your target machine is similar/same as the machine from which you took the original image OR if the kernel you plan to boot has support for your new target hardware, you should just be able to boot and run at this point. If not, read the following Gotchas section for further explanation.

This may all seem complex the first time you do it, but after a couple of times, you'll be able to do this in your sleep. This is one of those things where describing it is more complicated than just doing it!

Depending on how large your backup image is, a complete system restore can typically be done in less than an hour. That's less than an hour to a *completely configured system* with all your applications, custom configuration, and so on as you last left them.

Gotchas

If you use the approach described above to reprovision the same machine - say after a disk failure or disk upgrade - then that's all you have to do. Your "target" machine is essentially identical to the one from which you got the backup image ... the same machine.

However, there are circumstances where you cannot avoid doing some configuration on the newly provisioned machine. This is the case where there is a significant difference between the machine that took the snapshot and the machine receiving it. This might be because the target machine has different hardware, needs a different IP address, uses a different chipset, and so on.

What Problems Can I Expect?

So, you've decided to image a machine that is somehow different than the original source of the image. Here's what you'll possibly encounter:

A. Environmental Differences

   Your newly imaged machine may work fine except that its environment needs to change. The most common thing here is the need to reconfigure the NIC with new network parameters like IP address, netmask, DNS server, default route, and so on. Similarly, you may want to change the machine name or domain name. This is why you need to edit `/mnt/etc/rc.conf` before booting your newly imaged system.

   Keep in mind that changing the OS environment may also require changes in your applications' configuration. For instance, changing your machine name, IP, and so forth can break Apache. You may need to edit `/mnt/etc/rc.conf`

to temporarily prevent these applications from starting so that you can successfully boot boot the newly imaged system. Once the system is running, you can correct any applications' configuration that need to be changed.

B. Different Hardware

This is the tougher situation to handle after a machine has been newly imaged. Modern FreeBSD kernels come with enough standard driver support built-in that they should boot on most standard hardware ... unless you've hand tuned the kernel on the machine where the image was taken. You should therefore always build an image with a system that has the option to boot a GENERIC kernel. This kernel is likely to boot on almost all but the strangest hardware configurations.

However "booting" and "running properly" are two different things. If the hardware on your target machine is considerably different thatn the original machine on which the image was produced, you may need to do some further systems and/or kernel configuration.

Hardware differences show up in a number of places:

1. CPU Architecture

   If you built your image on a machine that is configured exclusively to run, say, on Pentium 4 chipsets, and then try to image another machine with an 80386, um ... it's not going to work. The kernels in your image have to be compatible with the CPU architecture on your target machine.

2. Motherboard Chipset

   Motherboards have so-called "Northbridge" and "Southbridge" chipsets. The Northbridge chip(s) control memory and high speed graphics (like AGP). The Southbridge chip(s) control the slower I/O functions and peripherals of the motherboard. If the machine you're imaging uses wildly different chipsets than the machine where the image was taken, you may have problems.

   If you have different Southbridges, you'll run into this with any of the on-board controllers:

   - Audio
   - Buses
   - Disk
   - Joystick
   - Network
   - Video

3. Peripheral Cards

   If your newly imaged machine has different PCI and/or video cards than the machine that produced the image, you may, again, have to install additional or different drivers.

The good news is that FreeBSD is much more forgiving than Linux or Windows are in this regard *so long as you can boot a GENERIC kernel*. The whole point of the GENERIC kernel is to be able to get the machine to boot. Once you're able to boot, it's a fairly straightforward matter to build a custom kernel or have the boot loader dynamically load the additional necessary kernel modules.

> **Tip**
>
> *Always* build your image on a machine that has a GENERIC kernel on it even if you boot a different or custom kernel by default. This will save your bacon later when you are imaging to other hardware configurations.

Author

Tim Daneliuk - tbku@tundraware.com

Comments and/or improvements welcome!

Document Information

This document was produced using the very useful `reStructuredText` tools in the `docutils` package. For more information, see:

http://docutils.sourceforge.net/rst.html

`$Id: Imaging-FreeBSD-With-tbku.rst,v 1.1 2012/06/09 18:07:30 tundra Exp $`