Thinking About Installers: Discord and Happiness

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
	2015-07-18		WB

Contents

Last updated 2015-07-18

Available in HTML or PDF. Links to all my articles here. Created with AsciiDoc.

This article was presented as a talk at BSDCan 2015. The LibreOffice Impress slides are available.

1 Who Cares?

Why should we care about an installer? It is just to get the basic system installed, after all.

It is a mistake to under-rate the importance of an installer, though. The installer is a user's first exposure to the system. The impression it makes will color the user's perception of the system itself. Making a good first impression is vitally important. A bad first impression might be the **only** impression.

2 What is an Installer?

Some would say an installer is just to get the basic system installed, but that is a mistake. An installer's job is really to install the system like the user wants.

A perfect installer would install exactly what the user wants. Actual, existing installers have some defaults that are correct, but some will always be wrong for any particular user.

3 Profiles

Let's call the set of installation options a "profile". A profile contains settings for everything an installer can do. A profile for a desktop machine would partition the hard drive for one big filesystem partition, install X, a desktop manager, and Firefox. A server profile might set up a dozen partitions with different filesystems, and install database and webserver packages.

4 Discord

The discord generated by an installer is proportional to the built-in profile, usually just defaults, and what the user really wants to do. If the defaults are very different from what the user wants, or the user has to install a lot of systems, the irritation caused by the difference grows. It wastes the user's time and energy, and makes them unhappy.

5 The Key Insight

An installer is really **two** things:

First, it is a profile editor.

Second, it is a profile installer.

To date, most installers do the second part well enough. *sysinstall(8)*, *bsdinstall(8)*, and *pc-sysinstall(8)* can all be scripted and offer varying degrees of low-level installation operations.

What most people miss is the first part, and the key to why so many people hate installers. If your needs are close to those envisioned when the installer was created, you'll love it. It may need a few additional features, but it mostly agrees with your use. If it does not agree with your use, it will be an endless source of frustration.

So what is being missed when installers are created?

Let's back up and ask this question: what would the perfect installer do? The answer is pretty easy: it would do exactly what you needed it to do, by default. Few or even no questions to answer, it would all be preprogrammed.

Let's go back to the idea of a profile, and take it one step farther: a profile can be composed of smaller profiles for individual tasks. Examples of these profiles are easy to imagine:

PROFILES

disk
: define disks, including plain disks or gmirror, motherboard RAID

partition
: MBR or GPT, the size and number of partitions

filesystem
: configures UFS or ZFS on the existing partitions

packages
: which packages are installed

user
: create users, initial passwords, home directories, default shells

At each step, the user is allowed to choose an existing profile or accept the current settings and continue. As they scroll through the profile names, the configuration of each is shown. A profile can be loaded for modification or used as-is. As profiles are defined, or after an install, they are saved, either to local media like a USB stick,or to a network drive, or to a remote site via SSH or HTTP.

When ready to install, a combination of sub-profiles for partition, filesystem, packages, users, and so on are saved as a "system" profile with a unique name.

After going through one installation, the user will have a small set of sub-profiles and one system profile. After several installations, the user might have a few system profiles that use different partition or filesystem profiles.

Now we make it great: when the installer starts, it looks for a set of profiles, both on local media and available network connections, and offers the user a choice of those profiles.

After a few installations, the predefined profiles are already configured. The user only has to pick one from the list.

```
Embedded: single disk, GPT, one partition, no swap, UFS, minimal packages
Desktop: single disk, GPT, 4G swap, ZFS single partition, KDE
Server:  two disks, gmirror, MBR, 8G swap, split partitions, UFS, Postgres, nginx
```

If a profile that does what they need is not present, they pick one that is close and change only the steps that are different.

6 Potential

There is a lot of automation potential with profiles. A profile could be automatically selected based on some hardware characteristic, like a MAC address or processor type.

Custom profiles could show only the fields that need user modification, so an install might need only a single user interaction, like setting a hostname. Or a profile could be automated so that it automatically generates everything, and the user does not have to interact with it at all.

Profiles could be shared with other users. Consider the example of a demo version of FreeBSD with a complete set of packages installed and configured for desktop use. A potential user would only need the installer and the URL of that profile to have a complete working machine. Rather than forcing them to download a single 2G or larger disk image, the target machine can fetch the small profile and download anything else it needs during installation.

The end effect is that the installer becomes the perfect installer for each user, doing exactly what they need.

7 Other Design Features

There are some additional desirable features and goals that a user-oriented installer should implement. Many of these are variations of "value the user's time".

7.1 Walk-Away Design

Have the user answer all the questions up front. Do not wait until the target system is formatted and installed to ask for the root password, and do not waste the user's time. Get all the information so the user can go do something else while the system is installed, and return to a completed installation.

7.2 No Entrapment

If a choice is not valid... do not show it! Do not show invalid options, let the user select them, and only then give an error message. Do not let the user even type letters into numeric-only fields, filter them out when typed. Wherever possible, prevent user errors.

Along the same lines, do not ask the user a question that they probably cannot answer:

```
Is this machine's CMOS clock set to UTC?  If it is set to local time
or you don't know, please choose NO here!
```

What percentage of people know what UTC is, and what fraction of those know whether their CMOS clock is set to it? What fraction of the potential user base knows what a CMOS clock is? Even if the user knows what UTC time and a CMOS clock are, there is no way to tell at that prompt what the time is set to in the target machine! Really, the only people who can answer this question are those who knew it was going to be asked and checked before even starting the installer.

Most people are going to select the default regardless of whether it is correct. This question is pointless. Set the time to whatever the default is. Power users will be able to change it, non-power users do not care, and asking only assures incorrect answers.

7.3 *dialog(1)* Makes You Stupid

For decades, FreeBSD installers have used *dialog(1)* for much of the user interface. The problem with *dialog(1)* is that it can only ask simple questions. It does not have facilities to make screens composed of multiple user input elements that would allow grouping of related functions together and make questions quicker and easier for the user to understand and answer. For example, *dialog(1)* cannot display a checkbox that allows the user to choose DHCP or a static IP address on a single screen. Instead, they must answer multiple questions on multiple screens. While the screens ask different questions, they look remarkably and confusingly alike.

Instead, we can use ncurses libraries to build logically-grouped custom input screens.

7.4 It Does Not Have to be in Base

Historically, FreeBSD installers have been written in C, *sh(1)*, or both. Neither is well-suited to building installers. A more powerful language can express the same functions in less code. This makes it easier to write, and more importantly, easier to maintain.

Fortunately, there are numerous languages which are more productive, more maintainable, and have an adequate license. Any of the Perluabython family will do.

Many will point out that these languages are not present in the base install of FreeBSD. My reaction to that is simple: so what? An installer disk does not need to be a bare FreeBSD install. It can and should include anything needed to make the job of installation easier.

7.5 Keep It Together

Why are there different installers for i386 and amd64? A 32-bit version runs on both and can install either. Why have separate install disks for -release and -stable? Why have a separate install disk for bootonly? The user can download bootonly, put it on a writable USB stick, and have the installer download and save additional compatible architectures, distributions, and packages (and profiles!) until the media is full.

7.6 Summary

The installer is the first exposure a new user has to FreeBSD. Experienced users have an ongoing relationship with the installer. With the concepts described here, we can make it an easy and rewarding experience that users remember and love.

7.7 Thanks

The concept of an installer saving a log of its actions that could document or be fed back in to recreate the install was suggested to me by Paul Schenkeveld when we talked in 2012. Thanks are also deserved by all the others who have maintained and written FreeBSD installers.