RCOS Project Proposal: big

Ian Ooi

Summer 2012

NOTE: Suggestions, modifications (in both function and implementation) are welcome and encouraged. The name is also open for change (as well as the syntax for the commands of course). I originally came up with big simply for the novelty of being able to run `big bang` and `big crunch`.

1 Overview

Big is a proposed "setup storing" tool, similar to System Restore for Windows, which at its base will keep a list of installed packages which it can re-download from specified repositories and install with the package manager of your choosing or reinstall from a specified local directory. It will also have the option of storing configuration files, attempting to detect them automatically and allowing the user to specify them in case of failure or additional files (vimrc, bashrc etc.).

Unlike a system image, or simply running a backup on `/` after you set up your system, big allows users to save their setup, modify a saved setup, recover an old setup, or use their setup on another machine *at any time*. As opposed to a simple backup, if a user decides they want to include another package, or not include a package in their stored setup, they can do so even if their current setup is completely different from the stored one. As a note, big will *not* store a users' files (besides conf. files), allowing for clean installs of the setup onto the same system or different systems.

The purpose is to allow a user who has a specific setup that they prefer for productivity or personal happiness to:

1. recover after accidental unwanted software changes such as deletion or changes to the system which cause unwanted changes

2. recover their setup after hardware damage, unwanted software changes, or to transfer their setup to new systems, such as additional computers or additional OS's (have the same packages on both Debian and ArchLinux)

3. otherwise backup their programs and configuration without storing their files to allow for expedited "clean" installs

2 Development Plan

The project will likely be developed in C/C++, though other languages could be considered (Python, Perl, or Haskell perhaps). Since these languages are also extensible through C/C++, it is plausible that there would be a combination of languages used for development.

The initial platform to develop for will be Debian/Ubuntu, targeting `.deb` packages and aptitude/synaptic/dpkg first, before extending to additional platforms.

Tentative development plan:

1. read info from a .deb package and stores it in a manageable format in a manifest file
2. step through directory, check settings from configuration file, either store/don't store packages appropriately. options are:
    - store everything
    - prompt user
    - check a list of "store only these"
    - "store these, ignore others"
    - "store these and ask about others"
    - "ignore these, store others"
    - "ignore these and ask about others"
    - "ignore all"
3. run through file system and grab configuration files associated with packages (also grabbing associated directories from packages)
4. find a way to latch onto package manager and grab the downloaded .deb the user is unpacking
5. grab stored repositories (Ubuntu's ppas)
6. run through file system and grab installed programs
7. create an archive and/or disk image which can be backed up
8. in cases where the user wants to burn the archive to a disk, split into multiple archives
9. extend to other Unix/Linux/BSD platforms
10. build into packages/install script, at least have src and Makefile to build with and README with instructions
11. extend to Mac
12. extend to Windows (essentially a whole new program, probably not worth building and/or maintaining)