Global Information Assurance Certification Paper

Using GUPI to Create A Null Box

GIAC (GCIH) Gold Certification

Author: Robert "Gremlin" Comella, Gremlinscs@aol.com
Advisor: Rick Wanner

Accepted: July 24th 2010

Abstract

This whitepaper introduces GUPI, a tool that helps administrators recognize removable packages. They can use it to create the Null Box, a Linux server administrators can use as a base for secure servers.
The most efficient and secure Linux machines only run the software necessary to fulfill their assigned task, no more. Therefore, the goal of a security-minded administrator is to build machines to that standard. While it is easy to understand this concept, it is difficult to realize it. Unfortunately, the normal Linux distribution is a "rat's nest" of interdependencies, making it difficult to determine which packages to remove. This paper introduces "Gremlin's Unnecessary Package Identifier", or GUPI, to solve this problem. GUPI identifies packages upon which no other packages depend, and designates these "removable". GUPI then allows the user to mark these packages "Remove" or "Keep". Given the user's input, it recalculates the new list of removable packages and presents it to the user. This tool allows the administrator to create the "Null Box", a Linux-based server that has only enough software to boot and install additional packages as necessary. This machine provides a base from which administrators can build secure machines for their networks.

1. Introduction

1.1. Servers with less do more

When an administrator builds a Linux server, they make many decisions. One of the most difficult is deciding which packages to install. Linux distributions, upon installation, try to pass package selection off as an easy choice. The administrator must simply choose a function from the list and the installation program will automatically install all the necessary software to provide that service. The installation usually works and the resulting machine performs the desired task. Administrators focused only on functionality consider themselves finished and move on to the next task.

While it is impressive that Linux distribution masters can make this process as easy as it is, it is not possible for them to customize the system to the exact needs of every organization. The systems that result from these default installations, while completely functional, usually contain more software than is necessary for the assigned task to be completed. Since each software package is a potential source of vulnerabilities, the more installed packages there are the more vulnerable the machine is. Secondly, unnecessary services and software take up precious resources, which leave fewer resources for the server's assigned tasks. Finally, servers with more packages than necessary are more complex, which makes them more difficult to maintain. (Pomeranz, 2009)

Clearly, then, it is desirable to create servers with only the software necessary to perform their assigned task. Unfortunately, that is difficult to accomplish. Linux packages all are interrelated in different ways, which makes it difficult to figure out which packages administrators can remove without causing damage to the system. If an administrator is to build a server containing only task-specific software, they must start from a machine that has only the minimum of what is essential for the server to function.

The focus of this paper is to create a system that has only what is required to boot and install more packages. An administrator can use this "Null Box" as a base from which he may build other machines. Please note the Null Box is not itself secure. The

system architect must add security packages to secure the server properly but the specific packages chosen will depend on the environment in which the server resides.

To aid in the construction of such a lean system this paper introduces a tool called GUPI (Gremlin's Unnecessary Package Identifier). GUPI scans the computer's installed packages, identifies which are removable, and displays them in a list. The user can then decide, for each package, whether to keep or remove it. The program takes the user-supplied information and uses it to identify the next set of removable packages that the user may investigate. This iterative process continues until there are no removable packages remaining.

2. Where to Begin…

2.1. About the test machine

GUPI should work on any Linux-based system that uses the `apt` package manager, but its author built it using Ubuntu for Ubuntu–based systems. This paper provides an example based on Ubuntu 10.04 Server to show how to use GUPI and create a null box.

2.2. Getting the Installation Media

The installation media is available to the public in several different ways. Administrators can request free CDs from Ubuntu by going to https://shipit.ubuntu.com/ and filling out the appropriate web forms. It is clear from the website that this method is somewhat discouraged and it does take up to ten weeks to get the CD to the one who requests it. The second option is to purchase the media from Ubuntu themselves or from one of their worldwide distributors. The cost is between five and ten dollars (depending on the conversion rate) and shipping is far quicker. The website http://www.ubuntu.com/desktop/get-ubuntu/cds has more details.

Most administrators, however, have the ability to download all of the different Ubuntu versions free from the internet. Each one is a little less than 700MB. Download the .iso file, calculate its MD5 and SHA1 hashes, and compare them to the hashes calculated by Ubuntu at http://releases.ubuntu.com. If they match, the administrator can

be reasonably certain that they obtained the .iso file free of both intentional and unintentional corruption. Those using Linux to download the file can use *md5sum* and *sha1sum* programs provided by most Linux distributions. Windows users must download and install another package in order to check hashes. One decent one called hashcalc is available free. Once the administrator had obtained good images, they can burn them to CD. Ubuntu CDs offer the opportunity to check the cd for defects upon boot-up. This check is highly recommended as it can save administrators a great deal of time troubleshooting errors caused by a faulty CD.

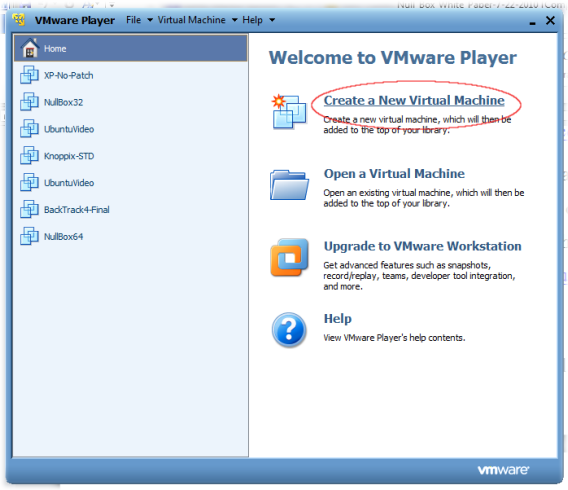
VMware can be an excellent tool for experimenting with which packages to keep and remove. It allows a user to create "snapshots" of the system by copying the directory where the virtual machine resides. When something occurs that puts the system in an unusable state, the administrator can copy the files back to the working directory from the backup and keep going. This takes far less time than re-installing a system. VMware Player® is freely available at www.vmware.com.

VMware Player® has a feature that is detrimental to this experiment. If the user chooses to use an .iso image to install the operating system and they select it when VMware player asks for details of the CD-ROM, VMware will "help" the user by implementing a tool called "Easy Install". This tool will automatically install the operation system, choosing options it thinks are best. Easy install prevents administrators from customizing the installation to their needs. To work around this, users must choose "I will install the operating system later" when creating a virtual machine.

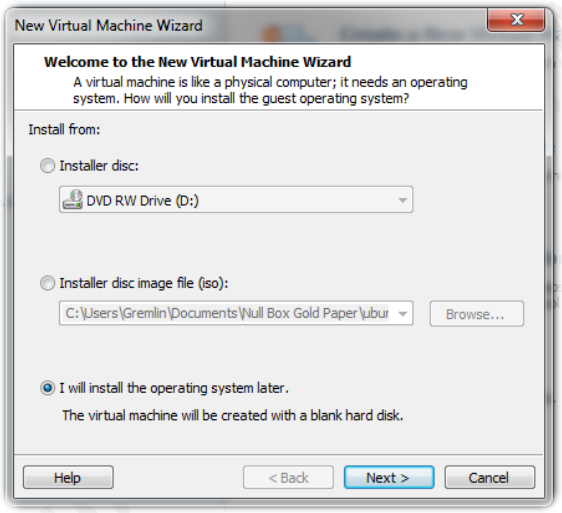
2.3. Exact choices used for the example

When creating the VMware Player test servers choose the following settings:

1. Click "Create a New Virtual Machine"

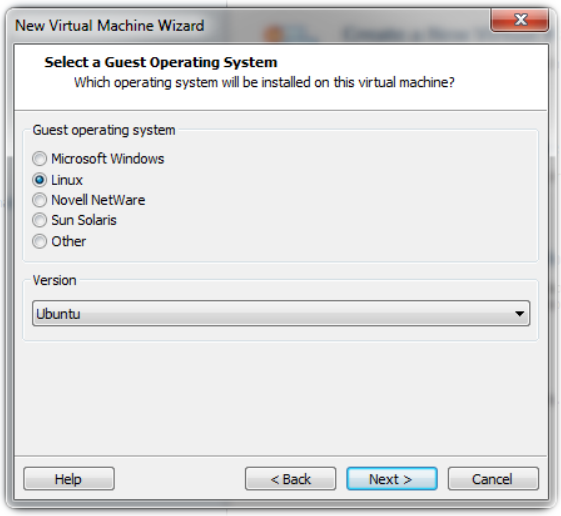


2. Choose "I will install the operation system later."

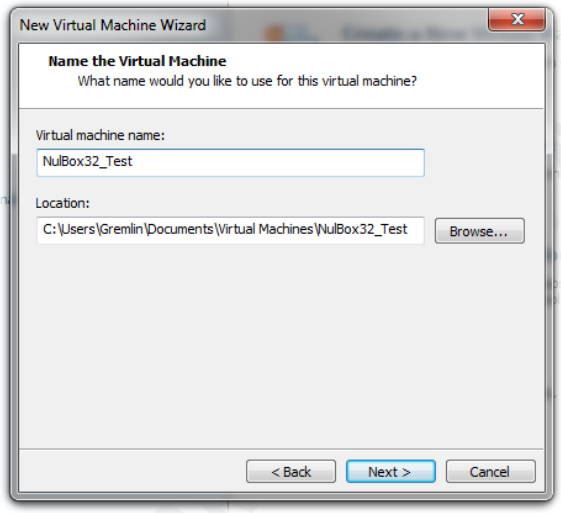


Author Name, email@address

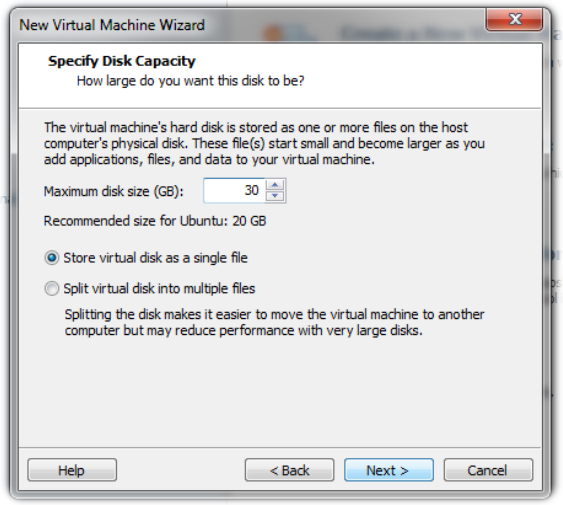
3. Choose an operating system of "Linux" with the version "Ubuntu"

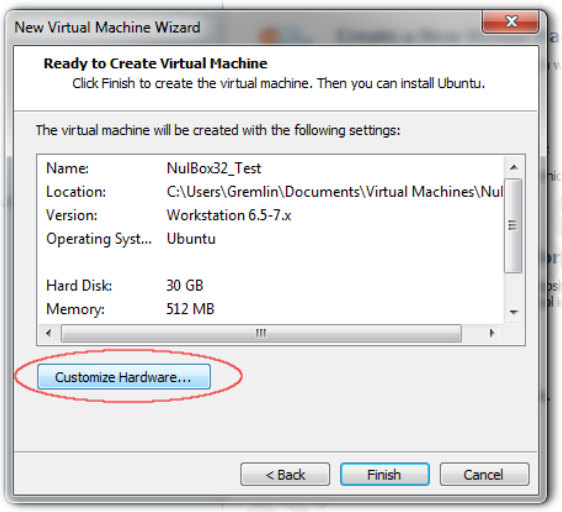


4. Name the virtual machine



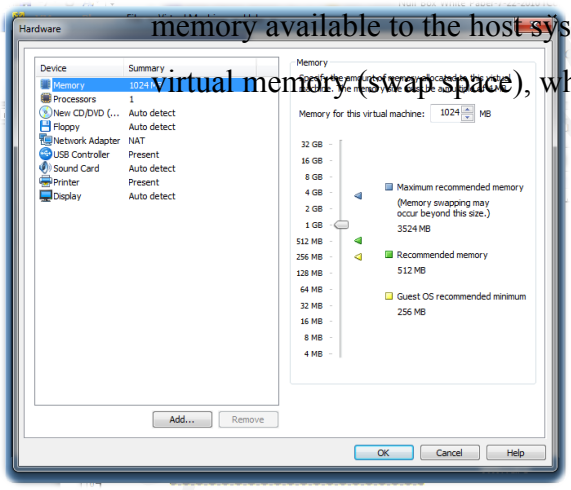
5. Choose a hard drive size (30 gig was used)





6. On the "Customize Hardware" screen:

   a. Set the memory to 1024MB if possible, but do not exceed the amount of memory available to the host system. The machine will be forced to use virtual memory (swap space), which is much slower than physical RAM.



   b. Set the CD/DVD to use the created CD or the downloaded .iso file.

   c. Remove the Floppy Drive, Sound Card, and printer.

   d. Set the network adapter to NAT but be sure to uncheck "Attached" and "Attach at power on".

7. Start the machine.

2.4. Preparing the Server

When repurposing a server it is possible to use the desktop version of Ubuntu to wipe the data from the hard drives. It is easy to do but it takes a long time for large hard

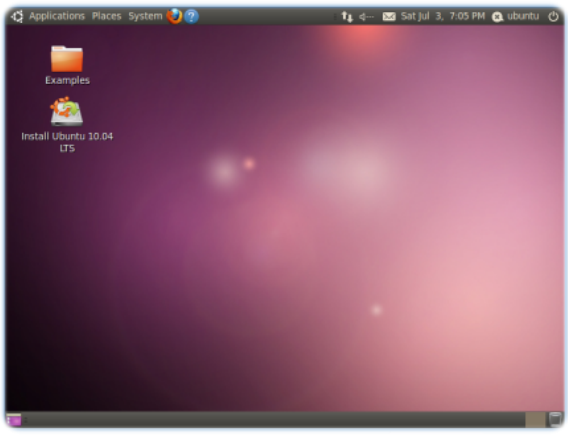
drives. Before starting, be certain to unplug all network cables from the machine. Insert the Ubuntu Desktop liveCD and boot from the CD-ROM. If everything is successful, the computer displays the screen below.

Choose the desired language and then choose to "Try Ubuntu 10.04". The computer will continue to load from this point until you have the normal Ubuntu desktop as seen again below.

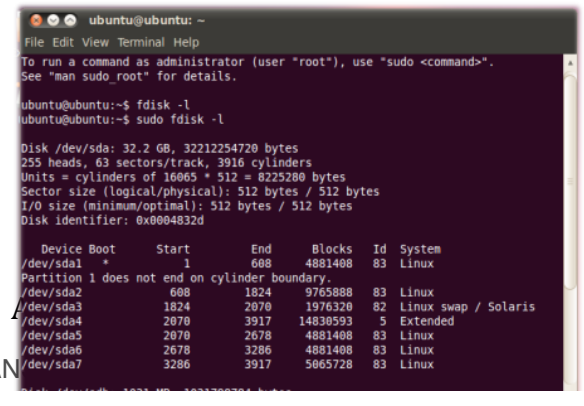
Now choose Applications → Accessories → Terminal to open a terminal window. Type the following command:

```
sudo fdisk -l
```

`sudo`	**Program to give temporary root privileges to a user**
`fdisk`	**Partition table manipulator for Linux**



`-l`	**Lists the partition tables for the devices on the system**



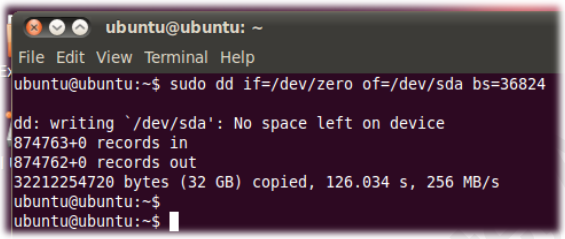
A list for each device on the system will appear as in the screen shown on the left. For each physical drive that needs to

be blanked, enter the following command, substituting the name of the drive in the "`of=`" option.

```
sudo dd if=/dev/zero of=/dev/sda bs=36824
```

`sudo`	**Program to give temporary root privileges to a user**
`dd`	**Program that copies data at a bit level**
`if=/dev/zero`	**dd parameter "input file" being set to a special device that give a never ending line of zeros (can also use /dev/random but it is much slower)**
`of=/dev/sda`	**dd parameter "output file" set to an example of a hard disk device**
`bs=36824`	**dd parameter "block size" set to an even number of sectors to help speed the process.**

You can repeat this for each open command window. Unfortunately, the version of dd that comes on the Desktop disk is an older version that has no progress reports.



The program gives no indication that anything is happening, however, the hard drive access light on the computer will be on steadily. This process may take many hours to complete, so for now find something else to do for a while. When it is finished, the screen should look like the picture. Close the window and use the circle button in the upper right part of the screen to shut the machine down.

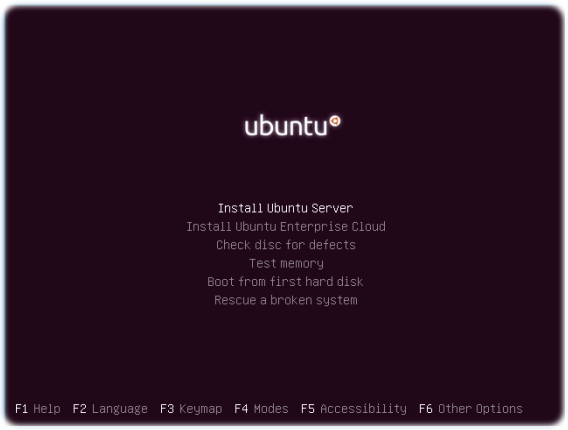
3. Building the Null Box

3.1. First Steps

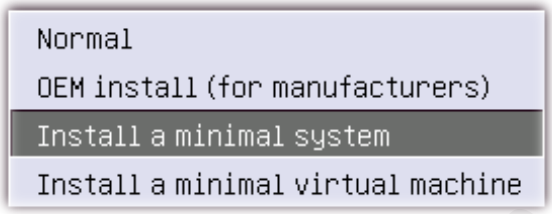
With a clean server, it is time to install the base operating system. This will be a working server. It may take some time and trial and error before an administrator is satisfied with the particular set of packages on a Null Box.

Author Name, email@address

To start the process the administrator needs to place the Ubuntu Server disk in the CD-ROM and again set the computer to boot from the CD-ROM. The network cables should remain unplugged. If the system boots successfully from the CD it will display the screen on the left where the user may select their language.



The welcome screen gives the administrator many options. This is where he may choose to check the disk for defects as mentioned above. If the user presses the F4 button, a few more options appear. To minimize the normal install as much as possible the user should select "Install a minimal system" from this list and then choose the "Install Ubuntu Server" option.



The installation process begins. The first choice is the language then the country in which the user is located. After asking about the user's location, it asks if it should detect the keyboard layout. English speaking persons with no special needs can select no, as the default is a North American keyboard. If the administrator answers yes, the installer package will ask a series of questions to try to determine the layout of the keyboard, which takes longer than just selecting the correct choice.

After the system loads some common files, it attempts to detect the network using DHCP. This will fail, because the network cables are unplugged. When the automatic detection fails, the installer may choose to configure the network automatically. Individual screens will appear asking for the IP address, net mask, gateway, and name server. The administrator can set the values as necessary for the network. (An aside for those using VMware player on Windows: During the installation of VMware, it will automatically select IP address ranges. Administrators can use the command ipconfig to ascertain what these settings are. The entry for VMnet8 has the appropriate settings for the NAT network. Also note that the gateway and DNS is 192.168.XXX.2)

Author Name, email@address

The next screen asks the user to enter the host name for the system. The following one asks for the domain name. Administrators should enter the values appropriate to the network. Next, the system will try to get time from a timeserver. This action, like the DHCP attempt, will also fail, but it will not display any sort of message. It will then ask for the time zone of the user.

3.2. Hard Disk Partitioning

The administrator will need to make some decisions here based on the function of the machine. To facilitate a more secure file system, it is important that at least partitions for "/", "/usr", "/var" are created. (Pomeranz, 2009) You will also need a partition for swap, which in general should be two times the size of the installed RAM. (Koconis, Murry, Purvis, & Wassom, 2004) Depending on the server, other partitions may be necessary. A "/home" directory will always be created when installing Ubuntu 10.04. Placing it on a separate partition will prevent users from filling the file system to cause a DOS. The /tmp directory can have its own partition or it can be mounted in memory on /dev/shm. Doing so may be faster on machines with enough memory to handle it. It is also possible to partition off subdirectories. For example, administrators can create a "/usr/local" partition to store third party applications.

Size of each of these will vary greatly depending on need. Hard drives are generally rather large and the installation on most servers does not take much space. Remember that logs will generally go to /var, the operating system will be placed mostly in /usr, and everything not given a specific partition will end up in the "/" partition.

/	5gig	Ext4	Primary
/usr	10gig	Ext4	Primary
Swap	2048Mb	-	Primary
/var	5gig	Ext4	Extended
/tmp	2 gig	Ext4	Extended
/home	10.2gig	Ext4	Extended

No matter the partition scheme chosen, the administrator must choose the Manual option in the partition disks dialog box. Use the on-screen prompts to set up the necessary partitions. In this example, the partitions were set up according to the table at the left. Be sure to set the bootable flag on the "/" partition or the system will be unable to boot.

3.3. Finishing the install process

After creating the partitions, the installer program will load most of the files to the hard disk. After a few moments, it will ask for the name of the main user. Ubuntu, by default, sets the root password to make it impossible to log on a root. This user is set to be "administrator" of the machine. This user does not have root privileges but can use sudo to gain root status for any command. Choose carefully the name and password for this account over the next few dialog boxes. Once the installer creates the user, it asks if it should encrypt the user's directory. This is up to the administrator. The example computer did not encrypt the user's directory.

One of the final dialog boxes shown asks about proxy information. Fill in this dialog box appropriately for the network. Leave it blank if there is none. The computer will attempt to download several files from the internet. This, like the other network-related activities, will fail. The system will look as though it has hung for a little while but it will eventually move on.

Next, it will ask how to manage package updates on the system. The three options are "No automatic updates", "Install security updates automatically", or "Manage system with Landscape". Landscape is a paid tool that may be interesting but it is outside the scope of this document. Many administrators do not trust others to install updates on production machines without testing them first so they choose "no automatic updates".

Finally, the installer asks what software the administrator wishes to install on the system. Any option chosen here will direct the installer program to install all the software necessary for a generic but functional system. In order to obtain the most basic system, the administrator should skip this altogether by pressing tab without selecting any extra packages.

With that, the installer program installs several hundred files, does some configuration, and cleanup. When it is finished, it will ask if the user wishes to install grub-pc to the master boot record. Choosing no to this question will leave the computer in an unusable state. Finally, the installer program instructs the administrator to remove the disk and reboot the machine. Once the machine boots it is running a minimal version of Ubuntu.

Author Name, email@address

The Ubuntu server installer does rather well with the minimal install. For not knowing people's exact needs, the Ubuntu team produced a tight package. There are no listening ports (an nmap scan confirms this). Only seventeen processes run in memory. Finally, three hundred six installed packages take up about 1075 MB of disk space.

If the administrator watches closely, they may notice some boot messages flying by. To view them at a more leisurely pace, run:

```
cat /var/log/boot.log
```

If the installation completed successfully the administrator should see several messages about clean partitions, several init messages that terminated with status 4, and one message about AppArmor. While the status 4 messages look like errors, they are not. Here is a quote from the developer:

Status 4 (usually ureadahead-other exits with this) means that you had a mountpoint in your fstab that didn't have any files on it needed during boot. Probably that drive with all those MP3s and movie files on it.
It still reads everything needed during boot, the status is just there for me to debug other issues. (keybuk, 2010)

4. Removing Unnecessary Packages

4.1. Which Packages are Unnecessary?

A package whose removal does not prevent the system from booting nor prevent it from downloading and installing packages is unnecessary. The question becomes, "How does one identify packages that are not necessary?" Linux is like an onion. Most installed packages depend on others. Just as someone can peel the layers off an onion, revealing the layers below, administrators can peel the packages on the outside off, revealing other removable packages. So what programs are on the outside of the onion? They are the packages upon which no other packages rely. To find them, an administrator must take note of the dependencies of all the packages installed on the system then compare the list of dependencies to the list of the ones installed. Any installed packages not on the dependency list are not necessary, and therefore, removable. It is up to the administrator to decide if the package, when removed, breaks the ability of the machine to boot, download, and install new packages (Or any other criteria the

administrator may have in mind). If removing it does not interfere with the desired function of the machine then the administrator can remove it and its dependencies from the dependency list. If it does prevent the system from fulfilling its function, the administrator must keep the package. Once the user evaluates all the packages on the generated list of removable packages as legitimately removable or not, it is time to recalculate the dependency list and again compare it to the list of installed packages.

The administrator repeats the process of marking then recalculating until all removable are marked as must keep. At that time, the administrator cannot remove another package without breaking the criterion set. The system has only the packages necessary to fulfill its duty. The administrator can do this process manually (which takes forever) or they can use GUPI.

4.2. Installing and Running GUPI

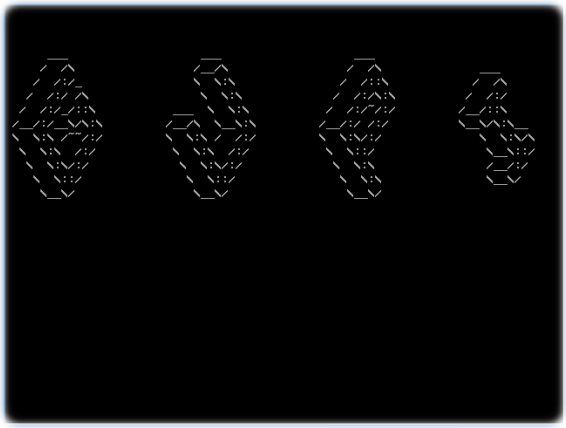
Robert "Gremlin" Comella wrote Gremlin's Unnecessary Package Identifier, or GUPI, for short, to automate the identification of nonessential packages discussed in the prior section. When run, GUPI creates a list of all installed software. Then it creates a list of all the packages upon which they depend. Finally, it compares the lists and displays the packages it finds on the installed packages list but not on the dependency list. The user can then mark the packages "remove" or "keep". When the user is finished with the list, GUPI will recalculate the package list and display a new list. Eventually, when nothing appears on the removable list the user can ask GUPI to produce a script that will remove all the packages indicated while running the software.

Installing GUPI is easy. GUPI consists of four script files. The user must create these four files on the machine or copy them to it. All the files must be in the same directory. The only prerequisites known are apt-cache, dpkg, and grep. Running the software is also simple. The user must cd to the directory where all the files reside and run the gupi.sh script.

```
./gupi.sh
```

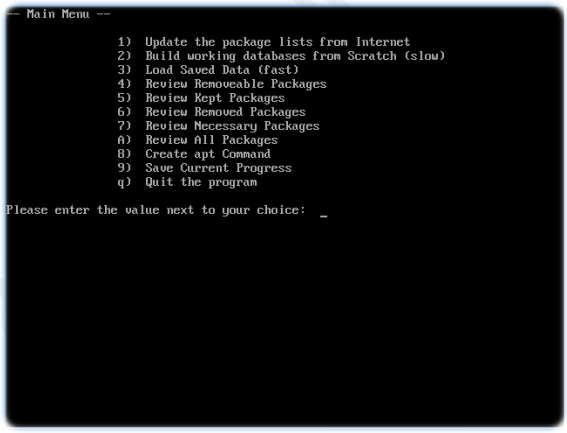
Or if the executable bit is not set for the package,

```
bash gupi.sh
```



For a moment, the program shows the title screen. Next, it displays the main menu. Before the program creates the package list, the main menu is very limited.

Option one will run `sudo apt-get update` to get the latest package information. Two will build the database of packages. This process takes longer, as the number of packages on the system increases. On slow computers with many packages, this process may take up to 15 minutes. In the test environment, however, it should only take a minute or two. Since it takes so long to create the list of packages, option three exists to load a previous database. The saved file must reside in the same directory as GUPI. This is in case the user needs to shut the machine down while in the middle of an investigation. They can save their work and begin again, where they left off. The final option will of course end the program.



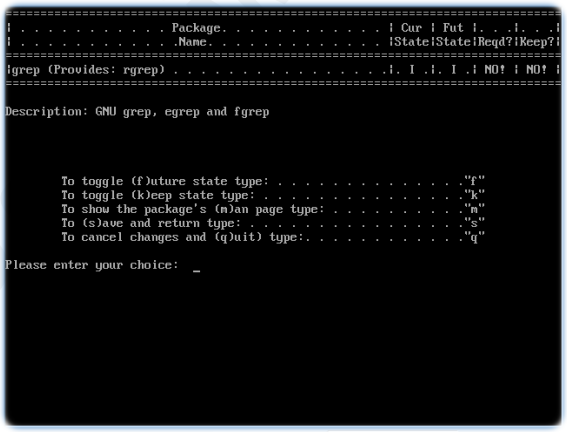
When the user selects option two, GUPI's current process and file will appear on the screen as it builds its database. When it completes, GUPI displays the entire main menu. "Review Removable Packages" shows the list of packages it considers removable. "Kept" packages are packages that the user marked as necessary even though they can

Author Name, email@address

remove them. "Removed Packages" are packages that the user tagged to remove. "Necessary" packages are those that the user cannot remove because another package depends on them. "All" displays all the packages.

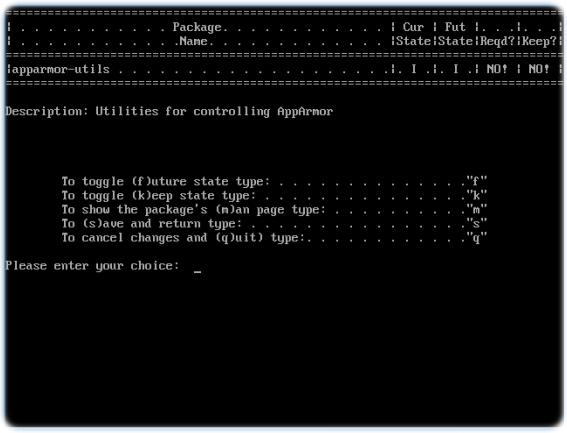
Option eight creates a script and places it in the directory with the apt files. The script contains an `apt-get` command that will remove all the packages in the "removed package" list. Option nine saves the current databases so that work may continue at a later point.

Choosing any of the options to review packages will display a package list screen. This package list screen displays information about the state of each package. At this time, "Current state" is "I", for "installed". Future state refers to the state of the package after running GUPI's generated script. "I" stands for "installed", and "U", for "uninstalled". The next column is "Required state" of the package. Its possible values are "YES" or "NO!" Finally, the "Keep" column tracks packages the user marked keep. Its possible values are also "YES" or "NO!". GUPI will show, at maximum, ten packages at a time. It is common, however, to have many more than ten packages in a particular list. The user can use "n" to see the next group of ten and "b" to go back to the previous list.



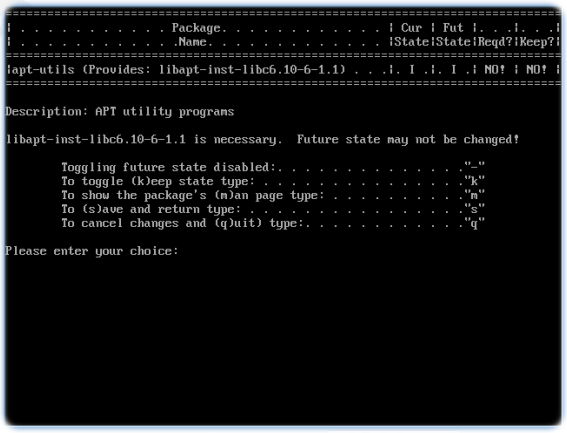
Enter the number beside the package name to look at that package individually. The program does not save changes immediately. As the user modifies package states, the program logs the changes. The functions "Save" and "Return" will write the changes to disk as well as calculate the new package lists. This avoids a lengthy pause between each change as the system writes

changes to the database. "Q" will return the user to the main menu, canceling any of the changes made since the last save.



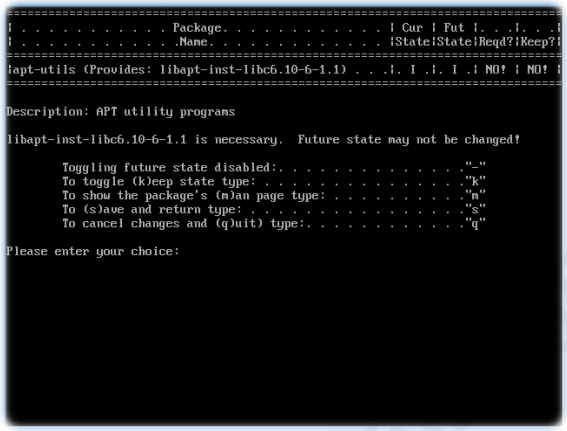
Choosing any of the numbers next to the listed packages will open the package modification screen. From this screen, the user is able to control the various states of the package if permitted. The screen shows the package's current states at the top. "f" will toggle the future state of the package and "k" will toggle the keep state of the package. "m" will attempt to display the man page for the package. If there is no man page, it will display a message telling the user. When the user is finished making changes, "s" will accept the changes and return the user to the package list screen while "q" will return the user to the package list without saving changes.

Two special cases of the package modification screen exist. The first occurs when a package provides another package. Currently there is no way for the program to discern, while it creates the list of "removable packages", that one package may not be removable because it provides another. The workaround for this issue for now is to check, on this screen, if the package displayed provides another necessary package. If it does, GUPI will disable the "f" option.



Author Name, email@address

The other special case is when a choice is available. In some cases, a single package may list several packages that will fulfill its dependency. The system still treats these packages as necessary, but the user may choose to use any of the packages shown to fulfill the dependency. Another option may be smaller or more specific to your environment.



An administrator can use GUPI on a system at any time to point out packages that he may remove. Allow it to create a list of packages to remove. Choose to remove or keep each then allow GUPI to recalculate. Look at the next set of packages decide to keep or remove. Rinse and repeat until finished. Once the user marks all removable packages as keep, they reached the end. He may not remove any other package without disturbing the ones that are necessary to fulfill the machine's function.

GUPI is not a perfect tool. There is one situation where GUPI fails. In some cases, two or more packages will rely on each other circuitously. For example, `tasksel` depends on `tasksel-data` and `tasksel-data` relies on `tasksel`. Even though both packages are removable, due to GUPI's algorithm, it always sees them as necessary. This does occur on a few occasions but the user must identify it.

4.3. Applying GUPI to the Null Box

The gory detail of which packages an administrator can remove and which ones he must keep is in appendix "B", but here are the highlights.

While some packages are technically removable, their absence will cause issues on the system. Removing `apt-utils` prevents `apt-get` from setting up packages. In some cases, this is not a problem because the packages are very simple. The `dhcp3-client` package obtains IP address information during the boot process. Assigning a static IP address before removing this package will avoid any problems. It is possible to remove `grub-pc`. Doing so prevents the system from updating the boot files. If during the

uninstallation the administrator chooses to remove the files from the /boot directory the computer will be unable to boot. Removing `iptables` will prevent the user from modifying the kernel side of the firewall. This action effectively disables the firewall. The program that builds the message of the day file (MODT) uses `lsb-release` package. The administrator can fix the non-critical error removing its removal creates by modifying `/etc/update-motd.d/00-header` file. Certain packages require `ncurses-base` to install correctly. Removing it prevents the installation of those packages.

Due to GUPI's current configuration, it cannot discover packages with circular dependencies. There are six packages with circular dependencies installed in the normal command line system install. The packages `tasksel` and `tasksel-data`, `perl` and `perl-modules`, and `kbd` and `console-setup` each have such a relationship. The administrator can remove all of these packages as long as they do so in pairs.

Of the original 306 packages installed by the Ubuntu install process, only 98 are necessary to boot the machine and allow an administrator to install other packages. Some packages must be marked as keep in order to allow the server to fulfill its function of booting and installing packages. Removing `apt`, `tar`, or `whiptail` prevents the installation of new packages. `apt` downloads new packages, `tar` opens the tarball (the form in which they are stored), and `whiptail` is a hidden dependency of `debconf`. The package `bash` provides the command shell. The user can replace it with alternative if they so desire. `debconf-i18n` is necessary but `debconf-english` can replace it. `debconf-english` is slightly smaller because it contains only support for English-speaking users. Administrators can remove `e2fsprogs` but when they do the boot scripts that check the drives fail. The package `gpgv` validates the packages in the Ubuntu package store. If the user removes it, the server complains that the package store is untrusted. `gzip` unzips packages retrieved form the package store without it most packages cannot be installed. The administrator can also remove `hostname` but doing so will cause errors to occur each time he runs a command in the shell. The command usually works but the server takes 10 to 15 seconds to time out in finding the name of the host before continuing. The `linux-image-2.6.XX-XX-generic-pae` is the kernel. Removing this package prevents the

Author Name, email@address

computer from functioning. `login` provides the ability for users to log into the computer, removing it stops users from logging on.

The `sudo` package is a little different. In the Ubuntu distribution, the creators have decided that no one should ever log on as root. The Ubuntu designers gave the root account a password that users cannot enter, effectively disabling it. It is possible to remove `sudo`, but the administrator must take two actions first. First, he must set the root account's password to something known. The second step is to export the SUDO_FORCE_REMOVE system variable by typing "`export SUDO_FORCE_REMOVE=yes`". Then it is possible to remove `sudo` and retain an operational system.

The final appendix of this paper contains a quick script that will remove all the possible packages from the experimental server built in this paper. It also performs minor changes to bolster the security of the machine.

5. Conclusions

If the administrator removes all the software possible, the null box is rather bare. It will serve no purpose except for eating electricity and holding down papers. The null box is secure only in the fact that there is very little to exploit. Further modification can improve its security.

Further actions that can be taken include adding a firewall, disabling unnecessary user accounts, setting password rules, setting the default umask, tuning the kernel, setting warning banners, and adjusting file system security. What actions the user takes and to what extent the user locks down the machine depends greatly on the environment in which a server operates as well as the server's final function.

Starting with a null box allows the administrator to install only what is necessary to make the server perform the duty assigned to it. Even if it is not possible to start with the null box and build up to the desired server, the GUPI program will help administrators identify removable packages.

Author Name, email@address

6. References

keybuk. (2010, March 20). *All about ureadahead*. Retrieved July 4, 2010, from Ubuntuforums.org: Http://ubuntuforums.org/showthread.php?t=1434502

Koconis, D., Murry, J., Purvis, J., & Wassom, D. (2004). *SANS Step-by-Step Series: Securing Linux A Survival Guied for Linux Security Version 2.0.* Washington DC: SANS.

Matthew, N., & Stones, R. (2008). *Beginning Linux Programming 4th Edition.* Indianapolis: Wiley Publishing, Inc.

Pomeranz, H. (2009). *Sans 506.2 Unix Hardening, Part 1.* Washington DC: SANS.

Williams, C. (1999). *Professional Visueal Basic 6 Databases With VM, ADO SQL and MTS.* Acock's Green: Wrox Press LTD.

Special thanks to Paul Kern and Rick Wanner who helped with the editing of this paper

7. Appendix A GUPI Source Code

7.1. Overview

The next four sections are the source code for the GUPI program. To create a useable version:

1. Create four text files, each named after the sections below.
2. Using a text editor (like notepad, notepad++, Kate, or gedit) cut and paste the code from each section into each of the files.
3. Save all the files into one directory.
4. Make them executable.

7.2. gupi.sh

```
#!/bin/bash
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
```

Author Name, email@address

```
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU Library General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor Boston, MA 02110-1301,  USA


#*****************************************************************************
#   Overall Description
#*****************************************************************************

#     This is the main controlling script for GUPI.  It will call the other
#             Scripts necessary to get work done.

#     GUPI or Gremlins' Unnecessary Package Identifier is a simple script that
#             will search through the packages installed on a computer and as the name
#             suggests identify the ones that can be removed.  I will continuously
#             update the list as packages are either added or removed virtually.
#             The final action the program will take is to create another script that
#             will both install and uninstall the packages requested during the
#             running of the program.

#     Note to all you decent hackers out there.  Please be kind to me this is
#             My first script of any consequence.  If you have any constructive
#             criticism you can contact me @ GremlinscsJunk@aol.com.  Please also note
#             that this is just a temporary program: I would like to release this as
#             open source tool but I think I will re-code it in C++ to speed it up and
#             to make it more extensable.

#     I know of no lions burned by this code.  Let me know if do.

#     Finally I wrote this in the Kate text editor.  If you open it in that
#             program it will look nice.

#*****************************************************************************
#   Included Script for the use of this script.
#*****************************************************************************

#     We need the database builder
. ./gupi_db_builder.sh
. ./gupi_packages.sh


#*****************************************************************************
#   Global Variables
#*****************************************************************************

# This variable adjusts menu choices based on actions
pkglstbuilt=0

#*****************************************************************************
#   Functions
#*****************************************************************************

get_main_menu_choice() {
      clear
      echo "-- Main Menu --"
      echo
      #  These Choices are always available
      echo "         1)  Update the package lists from Internet"
      echo "         2)  Build working databases from Scratch (slow)"
      echo "         3)  Load Saved Data (fast)"
      #  These will only appear after package lists are created
```

Author Name, email@address

```
      if [ "$pkglstbuilt" = "1" ]; then
             echo "         4)  Review Removeable Packages"
             echo "         5)  Review Kept Packages"
             echo "         6)  Review Removed Packages"
             echo "         7)  Review Necessary Packages"
             echo "         a)  Review All Packages"
             echo "         8)  Create apt Command"
             echo "         9)  Save Current Progress"
      fi
      echo "         q)  Quit the program"
      echo
      echo -e "Please enter the value next to your choice:  \c"
      read -n 1 main_menu_choice
      echo
      return
}

#*******************************************************************************
#   Main Script
#*******************************************************************************

# Display Main Menu

# Clear the screen  and display the title of the program

clear
echo
echo
echo
echo \
"      ___                    ___                    ___                       "
echo \
"     /  /\                  /__/\                  /  /\                ___     "
echo \
"    /  /:/_                 \  \:\                /  /::\              /  /\    "
echo \
"   /  /:/ /\                 \  \:\              /  /:/\:\            /  /:/    "
echo \
"  /  /:/_/::\            ___  \  \:\            /  /:/~/:/          /__/::\    "
echo \
" /__/:/__\/\:\          /__/\  \__\:\          /__/:/ /:/           \__\/\:\__  "
echo \
" \  \:\ /~~/:/          \  \:\ /  /:/          \  \:\/:/               \  \:\/\ "
echo \
"  \  \:\  /:/            \  \:\  /:/            \  \::/                 \__\::/ "
echo \
"   \  \:\/:/              \  \:\/:/              \  \:\                 /__/:/  "
echo \
"    \  \::/                \  \::/                \  \:\                \__\/   "
echo \
"     \__\/                  \__\/                  \__\/                       "
sleep 2


while [ "$quit" != "1" ];
do
      get_main_menu_choice
      case "$main_menu_choice" in
             1)
                    echo "Update Package lists from the Internet"
                    sudo apt-get update
                    sleep 1;;
             2)
                    echo "Creating Database..."
```

Author Name, email@address

```
        #       Get The list of all the packages
        #               get_pkgInfo
        BuildDatabase
        pkglstbuilt=1;;
3)
        if [ -e ./Depend_List ] || [ -e ./Package_List ]; then
                CopyDatabase
                pkglstbuilt=1
                echo "Save File Loaded."
                sleep 1
        else
                echo "No save data found.  Please build files."
                sleep 1
        fi;;
4)
        echo $pkglstbuilt
        if [ "$pkglstbuilt" = "1" ]; then
                DisplayRemovableList
        else
          echo "You must load the packages first!!"
          sleep 1
        fi;;
5)
        if [ "$pkglstbuilt" = "1" ]; then
                DisplayKeptList
        else
                echo "You must load the packages first!!"
                sleep 1
        fi;;
6)
        if [ "$pkglstbuilt" = "1" ]; then
                DisplayRemovedList
        else
                echo "You must load the packages first!!"
                sleep 1
        fi;;
7)
        if [ "$pkglstbuilt" = "1" ]; then
                DisplayNecessaryList
        else
                echo "You must load the packages first!!"
                sleep 1
        fi;;
a)
        if [ "$pkglstbuilt" = "1" ]; then
                DisplayAllList
        else
                echo "You must load the packages first!!"
                sleep 1
        fi;;
8)
        if [ "$pkglstbuilt" = "1" ]; then
                CreateCommand
        else
                echo "You must load the packages first!!"
                sleep 1
        fi;;
9)
        if [ "$pkglstbuilt" = "1" ]; then
                SaveDatabase
                echo "Database saved"
                sleep 1
        else
                echo "You must load the packages first!!"
```

Author Name, email@address

```
                        sleep 1
                fi;;

        q)
                Cleanup
                quit=1;;
        *)
                echo -e "That was not a valid choice.  Choose a value or press \n"
                echo "\"q\"to quit"
                sleep 2;;
    esac
done
```

7.3. gupi_db_builder.sh

```
#!/bin/bash
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU Library General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor Boston, MA 02110-1301,  USA


#*****************************************************************************
#   Overall Description
#*****************************************************************************

#     This script will handle all the database creation and upkeep issues
#     Related to gupi.  In the fastest manner possible it will create an update
#     two files.  The First file will be the package file.  It will contain:
#             Package Name:  [STR]   Self Explanatory
#             State:         [BOL]   Installed now or not
#             Future State:  [BOL]   To be installed or not installed
#             Necessary:     [BOL]   Necessary dependency for another package
#             Keep:          [BOL]   Not Specifically Necessary but User desires to
#                                    keep this package
#             Dirty:         [BOL]   If the package need to be checked for dependency

#     The Second file will be the relation file.  It will contain:
#             Package Name:                 Self Explanatory
#             Related package:       Self Explanatory
#             Relationship type:     Pre-Depends
#                                                Depends
#                                                Recommends
#                                                Suggests
#                                                Enhances
#                                                Breaks
#                                                Conflicts
#                                                Replaces
#                                                Provides

#     We can keep track of what is going on
#*****************************************************************************#
#*****************************************************************************#
#   Global Variables -- Not really necessary but I like to do this for clarity
#*****************************************************************************#
```

Author Name, email@address

```
#     Associate with the objects script
. ./objects.sh

#     Associate with the gupi common script
#. ./gupi_common.sh


#*****************************************************************************#
#*****************************************************************************#
#   Internal Functions
#*****************************************************************************#

IntPkgScrub()  #string_to_scrub relation_type
      #       This function will remove the extra stuff from entries in the table
{

#     #       Check if Internal go is set
#     if [ $bol_rel_int_go -eq "0" ]; then
#             echo "intCreatePath is an internal function only"
#             return $err_internal_go_not_set
#     fi

      #       Check to see if necessary values were passed
      if [ -z "$1" ] || [ -z "$2" ]; then
              #       Exit with an error
              return $err_argument_mismatch
      fi

      #       Set input to meaningful names for clarity
      local str_pkgname=$1
      local str_relation=$2

      #       Remove the relation label (e.g. Depends: or Pre-Depends:)
      local str_pkgname=${str_pkgname#$str_relation:}

      #       Remove the space before the package name
      local str_pkgname=${str_pkgname#" "}

      #       Remove the space after the package name
      local str_pkgname=${str_pkgname%" "}

      #       Remove any version information
      local str_pkgname=${str_pkgname% (*)}

      #       Return final string by reference
      str_pkgscrub_ret=$str_pkgname
}
#=============================================================================#
GetRelationships() #package_name relationship
      #       Send this internal function the package and the relationship and it will
      #               Create an array that has all the packages names are related to the
      #               target package in that way.
{
#     #       Check if Internal go is set
#     if [ $bol_rel_int_go -eq "0" ]; then
#             echo "intCreatePath is an internal function only"
#             return $err_internal_go_not_set
#     fi

      #       initialize the array
      unset array_related_pkgs_ret
      #       Check to see if necessary values were passed
      if [ -z "$1" ] || [ -z "$2" ]; then
```

Author Name, email@address

```
                #       Exit with an error
                return $err_argument_mismatch
        fi

        #       Set the inputs to names that have meaning
        local str_pkgname=$1
        local str_relationship=$2
        local str_relationship_path=$3

        #       Make note of the original IFS
        local ifsold="$IFS"

        #       Set the IFS to "," -- the separator in the file
        IFS=','

        #       Set the index number for the array
        local int_index=0

        #       If we loop through the apt-cache output with the current IFS we get each
        #               package in that line along with all the other information between
        #               the comas.
        int_index=0

        for str in $(apt-cache show $str_pkgname | grep -m 1 ^$str_relationship:)
        do
                #       If the file contains a | symbol then we need to split the value
                IFS='|'
                set $str
                local str_tempstring1="$1"

                #       Reset the IFS for next round
                IFS=','

                #       We can start the output with the first value
                IntPkgScrub $str_tempstring1 $str_relationship
                local str_final_value="$str_pkgscrub_ret"

                #       If we came across a more than one part this loop will take care of
                #       it and any others
                while [ "$2" != "" ]; do

                        #       Scrub the second part and concatenate it with the first
                        IntPkgScrub $2 $str_relationship
                        local str_final_value="$str_final_value"'|'"$str_pkgscrub_ret"

                        shift
                 done

                #       place the final value into the array
                echo "$str_pkgname•$str_final_value•$str_relationship" >> \
                                $str_relationship_path
                #array_related_pkgs_ret[$int_index]=$str_final_value
                let local "int_index += 1"
        done

        #       Set IFS back to its original value.
        IFS=$ifsold
}


#***************************************************************************
#***************************************************************************
#   Functions that are public -- These should be accessed by external users
#***************************************************************************
```

Author Name, email@address

```
PrePkgBuild()

#      This function will take no arguments.  It will build the list of all
#             packages available in the repository.  It sets preliminary values for
#             state, future state, keep, and necessary fields
{

       #       Create a package object
       ObjCreate Package_List

       #       apt-cache pkgnames will give a list of all the packages available on the
       #       system.  sort and uniq will make sure they are in order and there is
       #       only one of each.

#      for item in $(apt-cache pkgnames | sort | uniq)
#      do
#             echo "$item•0•0•0•0•0" >> $obj_str_path
#      done

       #       This is a truncated version I can use to keep testing times reasonable
       for item in $(dpkg --get-selections | awk '{print $1}')
       do
              echo "$item•0•0•0•0•0" >> $obj_str_path
       done


       #       Make 1 write

}
#============================================================================#
PkgRelBuild()

#      This function will take no arguments.  It will build a table that will
#             contain all the relationships between all the known packages.
{

       #       Get path to Package_List.  If it has not been created yet run
       #              PrePkgBuild and get the path.
       ObjExist Package_list
       local int_ret_val=$?

       #       If int_ret_val shows 32 create the object and continue otherwise
       #              continue
       if [ $int_ret_val -eq 32 ]; then
              PrePkgBuild
       fi

       #       Either the ObjExist or the PrePgkBuild functions will set the path
       local pkg_lst_file=$obj_str_path

       #       Create an object to hold the relationships
       ObjCreate Relation_List
       local rel_lst_path=$obj_str_path
       echo $rel_lst_path

       #       Calculate the relationships for each item in the list by iterating over
       #              The entire list
       itemindex=0
       for item in $(cat $pkg_lst_file)
       do
              GetRelationships ${item%%•*} Pre-Depends $rel_lst_path
              GetRelationships ${item%%•*} Depends $rel_lst_path
              GetRelationships ${item%%•*} Recommends $rel_lst_path
              GetRelationships ${item%%•*} Suggests $rel_lst_path
```

Author Name, email@address

```
                GetRelationships ${item%%•*} Enhances $rel_lst_path
                GetRelationships ${item%%•*} Breaks $rel_lst_path
                GetRelationships ${item%%•*} Conflicts $rel_lst_path
                GetRelationships ${item%%•*} Replaces $rel_lst_path
                GetRelationships ${item%%•*} Provides $rel_lst_path
                #       Set a local index
#               local int_end=$((${#array_related_pkgs_ret[@]}-1))
#               for i in $(seq 0 $int_end)
#               do
#                       echo "${item%%•*}•${array_related_pkgs_ret[$i]}•Depends" \
#                               >>$rel_lst_path
#               done
                let "itemindex += 1"
                if [ $(($itemindex % 100)) -eq 0 ]; then
                        echo $itemindex
                fi
        done

}
#===============================================================================#
MarkInstalled()

#       This function will take no inputs and it will go through and mark all the
#       packages that are installed in the list
{
        #       Get path to Package_List.  If it has not been created yet run
        #               PrePkgBuild and get the path.
        ObjExist Package_List
        local int_ret_val=$?

        #       If int_ret_val shows 32 create the object and continue otherwise
        #               continue
        if [ $int_ret_val -eq 32 ]; then
                PrePkgBuild
        fi

        #       Either the ObjExist or the PrePgkBuild functions will set the path
        local pkg_lst_file=$obj_str_path

        #       Create a file for the patterns
        #local path_pattern_file="$obj_str_path.pattern"
        #touch $path_pattern_file

        for item in $(dpkg --get-selections | awk '{print $1}')
        do
                #The periods that appear in search fields mess up grep so escape them
                local fixeditem=${item//\./\\\.}
                clear
                echo -e "Setting installed bit for: $item"
                #echo "^$fixeditem•[0-9]•[0-9]•[0-9]•[0-9]•[0-9]" >>"$path_pattern_file"
                grep -v "^$fixeditem•[0-9]•[0-9]•[0-9]•[0-9]•[0-9]" $obj_str_path >\
                        "$obj_str_path.tmp"
                echo "$item•1•1•0•0•0" >> "$obj_str_path.tmp"
                cat "$obj_str_path.tmp" | sort > $obj_str_path

        done

        #rm "$obj_str_path.pattern"
        rm "$obj_str_path.tmp"
}

#===============================================================================#
DepFileBuild()
```

Author Name, email@address

```
#	This function will take no arguments.  It will build a table that will
#		contain the names of all the files upon which others depend
{

	#	Get path to Package_List.  If it has not been created yet run
	#		PrePkgBuild and get the path.
	ObjExist Package_List
	local int_ret_val=$?

	#	If int_ret_val shows 32 create the object and continue otherwise
	#		continue
	if [ $int_ret_val -eq 32 ]; then
		PrePkgBuild
	fi

	#	Either the ObjExist or the PrePgkBuild functions will set the path
	local pkg_lst_file=$obj_str_path

	#	Create an object to hold the relationships

	ObjExist Depend_List
	local int_ret_val=$?

	if [ $int_ret_val -eq 32 ]; then
		ObjCreate Depend_List
	else
		ObjDestroy Depend_List
		ObjCreate Depend_List
	fi
	local rel_lst_path=$obj_str_path


	echo $rel_lst_path

	#	Calculate the relationships for each item in the list by iterating over
	#		the list of changed items that are to be installed in the future
	itemindex=0
	#for item in $(grep ".•[0-9]•[0-9]•[0-9]•[0-9]•1" $pkg_lst_file)
	for item in $(grep "•[0-9]•1•[0-9]•[0-9]•[0-9]" $pkg_lst_file)
	do
		GetRelationships ${item%%•*} Pre-Depends $rel_lst_path
		GetRelationships ${item%%•*} Depends $rel_lst_path

		#	Set a local index
#		local int_end=$((${#array_related_pkgs_ret[@]}-1))
#		for i in $(seq 0 $int_end)
#		do
#			echo "${item%%•*}•${array_related_pkgs_ret[$i]}•Depends" \
#				>>$rel_lst_path
#		done
		clear
		echo "Finding Dependencies for:  ${item%%•*}"
#sleep 1
#		let "itemindex += 1"
#		if [ $(($itemindex % 100)) -eq 0 ]; then
#			echo $itemindex
#		fi
	done
}

#============================================================================#
MarkNecessary()

#	This function will take no inputs and it will go through and mark all the
```

Author Name, email@address

```
#     packages that are necessary
{
      #       Get path to Package_List.  If it has not been created yet run
      #               PrePkgBuild and get the path.
      ObjExist Package_List
      local int_ret_val=$?

      #       If int_ret_val shows 32 create the object and continue otherwise
      #               continue
      if [ $int_ret_val -eq 32 ]; then
              PrePkgBuild
      fi

      #       Either the ObjExist or the PrePgkBuild functions will set the path
      local pkg_lst_file=$obj_str_path

      #       Make sure the Depend_List exists
      ObjExist Depend_List
      local int_ret_val=$?

      #       If int_ret_val shows 32 create the object and continue otherwise
      #               continue
      if [ $int_ret_val -eq 32 ]; then
              DepFileBuild
      fi

      #       Either the ObjExists or the DepFileBuild functions will set the path
      local dep_lst_file=$obj_str_path

      #       What needs to happen.  I need to go through all the installed packages
      #               and see if they are in the dependency list in the second column
      #for item in $(grep ".•1•[0-9]•[0-9]•[0-9]•1" $pkg_lst_file)
      for item in $(grep ".•[0-9]•1•[0-9]•[0-9]•[0-9]" $pkg_lst_file)
      do
              ifs_old=$IFS
              IFS="•"
              set $item
              IFS=$ifs_old
#echo $1
              #The periods that appear in search fields mess up grep so escape them
              #local fixeditem=${item%%•*}
              local fixeditem=$1
              local fixeditem=${fixeditem//\./\\\.}
              clear
              echo -e "Checking Necessity for: $1"
              if [ $(grep -c ".•$fixeditem•." $dep_lst_file) -gt 0 ]; then
                  grep -v "^$fixeditem•[0-9]•[0-9]•[0-9]•[0-9]•[0-9]" $pkg_lst_file\
                      >"$pkg_lst_file.tmp"
                  echo "$1•$2•$3•1•$5•$6" >> "$pkg_lst_file.tmp"
                  cat "$pkg_lst_file.tmp" | sort > $pkg_lst_file
              elif [ $(grep -c ".•.*|$fixeditem•." $dep_lst_file) -gt 0 ]; then
                  grep -v "^$fixeditem•[0-9]•[0-9]•[0-9]•[0-9]•[0-9]" $pkg_lst_file\
                      >"$pkg_lst_file.tmp"
                  echo "$1•$2•$3•1•$5•$6" >> "$pkg_lst_file.tmp"
                  cat "$pkg_lst_file.tmp" | sort > $pkg_lst_file
              elif [ $(grep -c ".•$fixeditem|.*•." $dep_lst_file) -gt 0 ]; then
                  grep -v "^$fixeditem•[0-9]•[0-9]•[0-9]•[0-9]•[0-9]" $pkg_lst_file\
                      >"$pkg_lst_file.tmp"
                  echo "$1•$2•$3•1•$5•$6" >> "$pkg_lst_file.tmp"
                  cat "$pkg_lst_file.tmp" | sort > $pkg_lst_file
              else
                  grep -v "^$fixeditem•[0-9]•[0-9]•[0-9]•[0-9]•[0-9]" $pkg_lst_file\
                      >"$pkg_lst_file.tmp"
                  echo "$1•$2•$3•0•$5•$6" >> "$pkg_lst_file.tmp"
```

Author Name, email@address

```
                cat "$pkg_lst_file.tmp" | sort > $pkg_lst_file
            fi
            #echo "$item•1•0•0•0•1" >> "$obj_str_path.tmp"
            #cat "$obj_str_path.tmp" | sort > $obj_str_path

      done
      rm $pkg_lst_file.tmp
}

BuildDatabase()

#     This function will just call the appropriate functions to build the database

{

      PrePkgBuild
      MarkInstalled
      DepFileBuild
      MarkNecessary

}

CopyDatabase()

#     This function is for testing purposes only since it takes so long to load
#             All the package information and update the tables correctly this
#             function will shortcut that by creating the home for the package info
#             then copying it from the directory where the program has been run

{
      #       This is a rather quick program that will create the appropriate
      #               Directory
      PrePkgBuild

      #       Now just copy the two file to that directory and exit.
      cp ./Depend_List $obj_dir_name/Depend_List
      cp ./Package_List $obj_dir_name/Package_List


}

SaveDatabase()

#     This function assumes that we have created the database already and copies
#             the files to the local directory so they can be used again in the
#             future
{
      #       Now just copy the two file to that directory and exit.
      cp  $obj_dir_name/Depend_List ./Depend_List
      cp  $obj_dir_name/Package_List ./Package_List
}

Cleanup()

{
      RemoveAllObjects
}

#     This function will take no arguments and simply clean up after the program
#             just before it exits.
#*****************************************************************************
#*****************************************************************************
#   Main Program -- mostly to test
#*****************************************************************************
```

Author Name, email@address

```
#BuildDatabase
#grep .•awk•. Depend_List
#bol_rel_int_go=1
#echo "Start" > ./test.txt
#ObjCreate Relation_List
#rel_lst_path=$obj_str_path
#echo $rel_lst_path
#GetRelationships zope-ploneformgen Depends $rel_lst_path
#MarkInstalled
#PkgRelBuild
#array_related_pkgs_ret[$int_index]
```

7.4. gupi_packages.sh

```
#!/bin/bash
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU Library General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor Boston, MA 02110-1301,  USA

#*****************************************************************************
#   Public Functions
#*****************************************************************************

UpdateDatabase()

#     This function will take no input.  It will take the contents of the array
#             and update the contents of the database with the changes.  IT will then
#             call the appropriate functions to update the dependencies
{
      #       Get path to Package_List.  If it has not been created yet run
      #               PrePkgBuild and get the path.
      ObjExist Package_List
      local int_ret_val=$?

      #       If int_ret_val shows 32 create the object and continue otherwise
      #               continue
      if [ $int_ret_val -eq 32 ]; then
            PrePkgBuild
      fi

      #       Either the ObjExist or the PrePgkBuild functions will set the path
      local pkg_lst_file=$obj_str_path

      #       Create a file for the patterns
      #local path_pattern_file="$obj_str_path.pattern"
      #touch $path_pattern_file


      #       Set the highest index number
      let local int_max=${#ary_item_list[@]}

      #       Set the starting value
      local int_index=0
```

```
	while [ $int_index -lt $int_max  ];
	do
		#	Tear list item apart
		ifs_old=$IFS
		IFS="•"
		set ${ary_item_list[$int_index]}
		IFS=$ifs_old

		#The periods that appear in search fields mess up grep so escape them
		local fixeditem=${1//\./\\\.}

		#	Let user know something is happening
		clear
		echo -e "Updating database with: $1"
#echo "grep -v \"^$fixeditem•[0-9]•[0-9]•[0-9]•[0-9]•[0-9]\" $obj_str_path \\" \
#>command.txt
#echo "\"$obj_str_path.tmp\"" >> command.txt
		grep -v "^$fixeditem•[0-9]•[0-9]•[0-9]•[0-9]•[0-9]" $obj_str_path >\
			"$obj_str_path.tmp"
		#read
		echo "$1•$2•$3•$4•$5•$6"  >> "$obj_str_path.tmp"
		cat "$obj_str_path.tmp" | sort > $obj_str_path
		let local int_index=$int_index+1
	done

	rm "$obj_str_path.tmp"
	DepFileBuild
	MarkNecessary
}

ToggleFutureState()  #	str_package_string

#	This function will simply swap the value of the future state variable
#		between 0 and 1
{

	if [ -z "$1" ]; then
		return
	fi

	str_package_string=$1

	#	Tear apart the entry
	ifs_old=$IFS
	IFS="•"
	set $str_package_string
	IFS=$ifs_old

	#	swap the state
	if [ $3 -eq 0 ]; then
		local str_fut_state="1"
	else
		local str_fut_state="0"
	fi

	#	A package can't have a future state of 0 and keep set to 1 so if future
	#		state is 0 change keep to 0
	if [ $str_fut_state -eq 0 ]; then
		local str_keep_state="0"
	else
		local str_keep_state=$5
	fi
```

Author Name, email@address

```
	#	rebuild string
	str_updated_value="$1•$2•$str_fut_state•$4•$str_keep_state•$6"

}

ToggleKeepState()	#	str_package_string

#	This function will simply swap the value of the future state variable
#		between 0 and 1
{

	if [ -z "$1" ]; then
		return
	fi
	str_package_string=$1

	#	Tear apart the entry
	ifs_old=$IFS
	IFS="•"
	set $str_package_string
	IFS=$ifs_old

	#	swap the state
	if [ $5 -eq 0 ]; then
		local str_keep_state="1"
	else
		local str_keep_state="0"
	fi

	#	If Keep is 1 future state can't be 0 so if keep is 1 change future to 1
	if [ $str_keep_state -eq 1 ]; then
		local str_fut_state="1"
	else
		local str_fut_state=$3
	fi

	#	rebuild string
	str_updated_value="$1•$2•$str_fut_state•$4•$str_keep_state•$6"

}

ShowManPage() #	str_package_string

#	This will simply show the man page of a package
{
		if [ -z "$1" ]; then
		return
	fi
	str_package_string=$1

	#	Tear apart the entry
	ifs_old=$IFS
	IFS="•"
	set $str_package_string
	IFS=$ifs_old

	#Show manpage
	man $1
}
DisplayListItem()	#item_to_display

#	This function will display a single item from the list and allow for
#		changes.
{
```

Author Name, email@address

```
if [ -z "$1" ]; then
       return
fi

local str_orig_string=$1

#      Display the normal header
local str_filler=". . . . . . . . . . . . . . . . . . . . . . . . . . . ."

#      Build the form header
clear
local str_line='========================================'
#local str_head1='|                        Package'
local str_head1="| . . . . . . . . . . . Package. . . . . . . . . . . . "
local str_head1="$str_head1| Cur | Fut |. . .|. . .|"
#local str_head2="|                         Name"
local str_head2="| . . . . . . . . . . . .Name. . . . . . . . . . . . . "
local str_head2="$str_head2|State|State|Reqd?|Keep?|"

echo -e $str_line$str_line
echo -e $str_head1
echo -e $str_head2
echo -e $str_line$str_line

#      Break up the input
ifs_old=$IFS
IFS="•"
set $str_orig_string
IFS=$ifs_old
local str_name=$1
local str_decrip=$(apt-cache show $str_name | grep -m 1 Description:)
if [ $2 -eq 0 ]; then
       local str_cur_state=". U ."
else
       local str_cur_state=". I ."
fi
#echo $str_cur_state
if [ $3 -eq 0 ]; then
       local str_fut_state=". U ."
else
       local str_fut_state=". I ."
fi
#echo $str_fut_state
if [ $4 -eq 0 ]; then
       local str_necessary=" NO! "
else
       local str_necessary=" YES "
fi
#echo $str_necessary
if [ $5 -eq 0 ]; then
       local str_keep=" NO! "
else
       local str_keep=" YES "
fi

local str_output="|$str_name $str_filler"

#      Create an object to hold the relationships

ObjExist Provide_List
local int_ret_val=$?

if [ $int_ret_val -eq 32 ]; then
       ObjCreate Provide_List
```

Author Name, email@address

```
else
        ObjDestroy Provide_List
        ObjCreate Provide_List
fi

local rel_provides_path=$obj_str_path

#       Make sure the Depend_List exists
ObjExist Depend_List
local int_ret_val=$?

#       If int_ret_val shows 32 create the object and continue otherwise
#               continue
if [ $int_ret_val -eq 32 ]; then
        DepFileBuild
fi

#       Either the ObjExists or the DepFileBuild functions will set the path
local dep_lst_file=$obj_str_path
local str_name_orig=$str_name

bol_Dependant='0'

#       Check to see if this package provides any other package it it
#               does then we need to check to see if the provided
#               package is considered necessary.  If so we need to
#               inform the user and prevent them from removing the
#               package.
GetRelationships $str_name Provides $rel_provides_path
if [ $(grep -c . $rel_provides_path) -gt 0 ]; then
    ifs_old=$IFS
    IFS="•"
    set $(grep -m 1 . $rel_provides_path)
    IFS=$ifs_old
    local fixeditem=$2
    local fixeditem=${fixeditem//\./\\\.}
    local "str_name=$str_name (Provides: $2)"
    if [ $(grep -c ".•$fixeditem•." $dep_lst_file) -gt 0 ]; then
       bol_Dependant='1'
    else
       bol_Dependant='0'
    fi
fi

#       Next we wish to find if there are multiple packages that may
#               fulfill a specific dependency.  In this case the user
#               Has a choice which package to use but this program
#               will only identify the issue and prevent the user
#               from uninstalling the package.
local fixeditem=$str_name_orig
local fixeditem=${fixeditem//\./\\\.}
if  [ $(grep -c ".•$fixeditem|.*•." $dep_lst_file) -gt 0 ]; then
    ifs_old=$IFS
    IFS="•"
    set $(grep -m 1 ".•$fixeditem|.*•." $dep_lst_file)
    IFS=$ifs_old
    local "str_name=$str_name (Choice: $2)"
fi

local str_output="|$str_name $str_filler"
local str_output="${str_output:0:55}"
local str_output="$str_output|$str_cur_state|$str_fut_state"
local str_output="$str_output|$str_necessary|$str_keep|"
echo $str_output
```

Author Name, email@address

```
        echo -e $str_line$str_line
        echo
        echo $str_decrip
        echo
        if [ $bol_Dependant = '1' ]; then
            echo "$2 is necessary.  Future state may not be changed!"
            echo
            echo "     Toggling future state disabled:. . . . . . . . . . . . . .\"-\""
        else
            echo
            echo
            echo
            echo "     To toggle (f)uture state type: . . . . . . . . . . . . . .\"f\""
        fi

        echo "  To toggle (k)eep state type: . . . . . . . . . . . . . . .\"k\""
        echo "  To show the package's (m)an page type: . . . . . . . . . .\"m\""
        echo "  To (s)ave and return type: . . . . . . . . . . . . . . . .\"s\""
        echo "  To cancel changes and (q)uit) type:. . . . . . . . . . . .\"q\""
        echo
        echo -e "Please enter your choice:  \c"
        read -n 1 item_choice
}

DisplayListMenu()    #int_offset

#       This function will display the list that was just created so the user may
#               make use of it.
{
        #       Set offset index:  The number we need to skip from the beginning
        if [ -z "$1" ]; then
                local int_start=0
        else
                local int_start=$1
        fi

        #       Set the highest index number
        #let local int_max=${#ary_item_list[@]}-1
        let local int_max=${#ary_item_list[@]}
        let local int_end=$int_start+10


        if [ $int_max -lt $int_end ]; then
                int_end=$int_max
        fi

        local int_index=$int_start
        local int_choice_val=0
        local str_filler=". . . . . . . . . . . . . . . . . . . . . . . . . . . ."

        #       Build the form header
        clear
        local str_line='========================================'
        #local str_head1='|                         Package                       '
        local str_head1="| . . . . . . . . . . . Package. . . . . . . . . . . . "
        local str_head1="$str_head1| Cur | Fut |. . .|. . .|"
        #local str_head2="|                          Name                         "
        local str_head2="| . . . . . . . . . . . .Name. . . . . . . . . . . . . "
        local str_head2="$str_head2|State|State|Reqd?|Keep?|"

        echo -e $str_line$str_line
        echo -e $str_head1
        echo -e $str_head2
        echo -e $str_line$str_line
```

Author Name, email@address

```
	while [ $int_index -lt $int_end  ];
	do
		ifs_old=$IFS
		IFS="•"
		set ${ary_item_list[$int_index]}
		IFS=$ifs_old

		local str_name=$1


		if [ $2 -eq 0 ]; then
			local str_cur_state=". U ."
		else
			local str_cur_state=". I ."
		fi

		if [ $3 -eq 0 ]; then
			local str_fut_state=". U ."
		else
			local str_fut_state=". I ."
		fi

		if [ $4 -eq 0 ]; then
			local str_necessary=" NO! "
		else
			local str_necessary=" YES "
		fi

		if [ $5 -eq 0 ]; then
			local str_keep=" NO! "
		else
			local str_keep=" YES "
		fi

		local str_output="|$int_choice_val."
		local str_output="$str_output $str_name $str_filler"
		local str_output="${str_output:0:55}"
		local str_output="$str_output|$str_cur_state|$str_fut_state"
		local str_output="$str_output|$str_necessary|$str_keep|"
		echo $str_output

		let local int_index=$int_index+1
		let local int_choice_val=$int_choice_val+1
	done
	#	Now I want to list 10 items from the list we created this list should
	#		be from the offset plus 10 up to the highest number
	echo -e $str_line$str_line
	echo
	echo "  To modify a row please enter the number to its left: . . [0-9]"
	echo "  To move forward to the (n)ext list type: . . . . . . . . \"n\""
	echo "  To move (b)ack to the previous list type:. . . . . . . . \"b\""
	echo "  To (s)ave your changes type: . . . . . . . . . . . . . . \"s\""
	echo "  To save and (r)eturn type: . . . . . . . . . . . . . . . \"r\""
	echo "  To cancel changes (q)uit) type:. . . . . . . . . . . . . \"q\""
	echo
	echo -e "Please enter your choice:  \c"
	read -n 1 display_choice
}

ListItemMenu()	#	str_package_entry
#	This function will get and act on the choices for the List Item

{
```

Author Name, email@address

```
	itemmenuquit=0
	str_list_entry=$1
	str_list_backup=$1
	while [ "$itemmenuquit" != "1" ];
	do
		DisplayListItem $str_list_entry
		case "$item_choice" in
			f)

				#	Tear apart the entry
				ifs_old=$IFS
				IFS="•"
				set $str_list_entry
				IFS=$ifs_old
				if [ $4 = '1' ]; then
				    clear
				  echo "Cannot change future state of necessary packages."
				    sleep 2
				elif [ $bol_Dependant = '1' ]; then
				    clear
				    echo "Adjustment of future state not permitted."
				    sleep 2
				else
				    ToggleFutureState $str_list_entry
				    str_list_entry=$str_updated_value
				fi;;

			k)
				ToggleKeepState $str_list_entry
				str_list_entry=$str_updated_value;;
			m)
				clear
				ShowManPage $str_list_entry
				sleep 1;;
			s)
				str_updated_value=$str_list_entry
				itemmenuquit=1;;
			q)
				str_updated_value=$str_list_backup
				itemmenuquit=1;;
			*)
			echo -e "That was not a valid choice.  Choose a value or press \n"
				echo "\"q\"to quit"
				sleep 2;;
		esac
	done

}

DisplayRemovableList()
{

	#	Get path to Package_List.  If it has not been created yet run
	#		PrePkgBuild and get the path.
	ObjExist Package_List
	local int_ret_val=$?

	#	If int_ret_val shows 32 create the object and continue otherwise
	#		continue
	if [ $int_ret_val -eq 32 ]; then
		PrePkgBuild
	fi

	#	Either the ObjExist or the PrePgkBuild functions will set the path
```

```
	pkg_lst_file=$obj_str_path

	#	Assign each returned string to a field in an array.  This way I can
	#		easily find which is going to be modified.
	unset ary_item_list
	local int_index=0
	for item in $(grep "•[0-9]•1•0•0•[0-9]" $pkg_lst_file)
	do
		ary_item_list[$int_index]=$item
		let "int_index += 1"
	done

	local start_val=0

	menuquit=0
	while [ "$menuquit" != "1" ];
	do
		DisplayListMenu $start_val
		case "$display_choice" in
			[0-9]*)
				let local showitem=$display_choice+$start_val
				if [ $showitem -ge $int_index ]; then
				    showitem=$int_index
				    let "showitem -= 1"
				fi
				ListItemMenu ${ary_item_list[$showitem]}
				ary_item_list[$showitem]=$str_updated_value;;
			n)
				let start_val=$start_val+10
				if [ $start_val -gt $int_index ]; then
					let start_val=$start_val-10
				fi;;
			b)
				let start_val=$start_val-10
				if [ $start_val -lt 0 ]; then
					let start_val=0
				fi;;
			s)
				UpdateDatabase
				local int_index=0
				unset ary_item_list
				for item in $(grep "•[0-9]•1•0•0•[0-9]" $pkg_lst_file)
				do
					ary_item_list[$int_index]=$item
					let "int_index += 1"
				done;;
			r)
				UpdateDatabase
				menuquit=1;;
			q)
				menuquit=1;;
			*)
			echo -e "That was not a valid choice.  Choose a value or press \n"
				echo "\"q\"to quit"
				sleep 2;;
		esac
	done
}

DisplayKeptList()
{

	#	Get path to Package_List.  If it has not been created yet run
	#		PrePkgBuild and get the path.
```

Author Name, email@address

```
ObjExist Package_List
local int_ret_val=$?

#       If int_ret_val shows 32 create the object and continue otherwise
#               continue
if [ $int_ret_val -eq 32 ]; then
        PrePkgBuild
fi

#       Either the ObjExist or the PrePgkBuild functions will set the path
pkg_lst_file=$obj_str_path

#       Assign each returned string to a field in an array.  This way I can
#               easily find which is going to be modified.
unset ary_item_list
local int_index=0
for item in $(grep "•[0-9]•1•0•1•[0-9]" $pkg_lst_file)
do
        ary_item_list[$int_index]=$item
        let "int_index += 1"
done

local start_val=0

menuquit=0
while [ "$menuquit" != "1" ];
do
        DisplayListMenu $start_val
        case "$display_choice" in
                [0-9]*)
                        let local showitem=$display_choice+$start_val
                        if [ $showitem -ge $int_index ]; then
                            showitem=$int_index
                            let "showitem -= 1"
                        fi
                        ListItemMenu ${ary_item_list[$showitem]}
                        ary_item_list[$showitem]=$str_updated_value;;
                n)
                        let start_val=$start_val+10
                        if [ $start_val -gt $int_index ]; then
                                let start_val=$start_val-10
                        fi;;
                b)
                        let start_val=$start_val-10
                        if [ $start_val -lt 0 ]; then
                                let start_val=0
                        fi;;
                s)
                        UpdateDatabase
                        local int_index=0
                        unset ary_item_list
                        for item in $(grep "•[0-9]•1•0•1•[0-9]" $pkg_lst_file)
                        do
                                ary_item_list[$int_index]=$item
                                let "int_index += 1"
                        done;;
                r)
                        UpdateDatabase
                        menuquit=1;;
                q)
                        menuquit=1;;
                *)
                echo -e "That was not a valid choice.  Choose a value or press \n"
                        echo "\"q\"to quit"
```

Author Name, email@address

```
                        sleep 2;;
                esac
        done
}

DisplayRemovedList()
{

        #       Get path to Package_List.  If it has not been created yet run
        #               PrePkgBuild and get the path.
        ObjExist Package_List
        local int_ret_val=$?

        #       If int_ret_val shows 32 create the object and continue otherwise
        #               continue
        if [ $int_ret_val -eq 32 ]; then
                PrePkgBuild
        fi

        #       Either the ObjExist or the PrePgkBuild functions will set the path
        pkg_lst_file=$obj_str_path

        #       Assign each returned string to a field in an array.  This way I can
        #               easily find which is going to be modified.
        unset ary_item_list
        local int_index=0
        for item in $(grep "•[0-9]•0•0•0•[0-9]" $pkg_lst_file)
        do
                ary_item_list[$int_index]=$item
                let "int_index += 1"
        done

        local start_val=0

        menuquit=0
        while [ "$menuquit" != "1" ];
        do
                DisplayListMenu $start_val
                case "$display_choice" in
                        [0-9]*)
                                let local showitem=$display_choice+$start_val
                                if [ $showitem -ge $int_index ]; then
                                    showitem=$int_index
                                    let "showitem -= 1"
                                fi
                                ListItemMenu ${ary_item_list[$showitem]}
                                ary_item_list[$showitem]=$str_updated_value;;
                        n)
                                let start_val=$start_val+10
                                if [ $start_val -gt $int_index ]; then
                                        let start_val=$start_val-10
                                fi;;
                        b)
                                let start_val=$start_val-10
                                if [ $start_val -lt 0 ]; then
                                        let start_val=0
                                fi;;
                        s)
                                UpdateDatabase
                                local int_index=0
                                unset ary_item_list
                                for item in $(grep "•[0-9]•0•0•0•[0-9]" $pkg_lst_file)
                                do
                                        ary_item_list[$int_index]=$item
```

Author Name, email@address

```
                                        let "int_index += 1"
                                done;;
                        r)
                                UpdateDatabase
                                menuquit=1;;
                        q)
                                menuquit=1;;
                        *)
                        echo -e "That was not a valid choice.  Choose a value or press \n"
                                echo "\"q\"to quit"
                                sleep 2;;
                esac
        done
}

DisplayNecessaryList()
{

        #       Get path to Package_List.  If it has not been created yet run
        #               PrePkgBuild and get the path.
        ObjExist Package_List
        local int_ret_val=$?

        #       If int_ret_val shows 32 create the object and continue otherwise
        #               continue
        if [ $int_ret_val -eq 32 ]; then
                PrePkgBuild
        fi

        #       Either the ObjExist or the PrePgkBuild functions will set the path
        pkg_lst_file=$obj_str_path

        #       Assign each returned string to a field in an array.  This way I can
        #               easily find which is going to be modified.
        unset ary_item_list
        local int_index=0
        for item in $(grep "•[0-9]•[0-9]•1•[0-9]•[0-9]" $pkg_lst_file)
        do
                ary_item_list[$int_index]=$item
                let "int_index += 1"
        done

        local start_val=0

        menuquit=0
        while [ "$menuquit" != "1" ];
        do
                DisplayListMenu $start_val
                case "$display_choice" in
                        [0-9]*)
                                let local showitem=$display_choice+$start_val
                                if [ $showitem -ge $int_index ]; then
                                    showitem=$int_index
                                    let "showitem -= 1"
                                fi
                                ListItemMenu ${ary_item_list[$showitem]}
                                ary_item_list[$showitem]=$str_updated_value;;
                        n)
                                let start_val=$start_val+10
                                if [ $start_val -gt $int_index ]; then
                                        let start_val=$start_val-10
                                fi;;
                        b)
                                let start_val=$start_val-10
```

Author Name, email@address

```
                        if [ $start_val -lt 0 ]; then
                                let start_val=0
                        fi;;
                s)
                        UpdateDatabase
                        local int_index=0
                        unset ary_item_list
                        for item in $(grep "•[0-9]•1•0•1•[0-9]" $pkg_lst_file)
                        do
                                ary_item_list[$int_index]=$item
                                let "int_index += 1"
                        done;;
                r)
                        UpdateDatabase
                        menuquit=1;;
                q)
                        menuquit=1;;
                *)
                echo -e "That was not a valid choice.  Choose a value or press \n"
                        echo "\"q\"to quit"
                        sleep 2;;
        esac
    done
}

DisplayAllList()
{

    #       Get path to Package_List.  If it has not been created yet run
    #               PrePkgBuild and get the path.
    ObjExist Package_List
    local int_ret_val=$?

    #       If int_ret_val shows 32 create the object and continue otherwise
    #               continue
    if [ $int_ret_val -eq 32 ]; then
            PrePkgBuild
    fi

    #       Either the ObjExist or the PrePgkBuild functions will set the path
    pkg_lst_file=$obj_str_path

    #       Assign each returned string to a field in an array.  This way I can
    #               easily find which is going to be modified.

    unset ary_item_list
    local int_index=0
    for item in $(grep "•[0-9]•[0-9]•[0-9]•[0-9]•[0-9]" $pkg_lst_file)
    do
            ary_item_list[$int_index]=$item
            let "int_index += 1"
    done

    local start_val=0

    menuquit=0
    while [ "$menuquit" != "1" ];
    do
            DisplayListMenu $start_val
            case "$display_choice" in
                    [0-9]*)
                            let local showitem=$display_choice+$start_val

                            if [ $showitem -ge $int_index ]; then
```

```
                                showitem=$int_index
                                let "showitem -= 1"
                        fi
                        ListItemMenu ${ary_item_list[$showitem]}
                        ary_item_list[$showitem]=$str_updated_value;;
                n)
                        let start_val=$start_val+10
                        if [ $start_val -gt $int_index ]; then
                                let start_val=$start_val-10
                        fi;;
                b)
                        let start_val=$start_val-10
                        if [ $start_val -lt 0 ]; then
                                let start_val=0
                        fi;;
                s)
                        UpdateDatabase
                        local int_index=0
                        unset ary_item_list
                        for item in $(grep "•[0-9]•[0-9]•[0-9]•[0-9]•[0-9]" \
                                $pkg_lst_file)
                        do
                                ary_item_list[$int_index]=$item
                                let "int_index += 1"
                        done;;
                r)
                        UpdateDatabase
                        menuquit=1;;
                q)
                        menuquit=1;;
                *)
                echo -e "That was not a valid choice.  Choose a value or press \n"
                        echo "\"q\"to quit"
                        sleep 2;;
        esac
    done
}

CreateCommand()

#       This command will write the command you need to uninstall the unwanted
#               packages
{

    #       Get path to Package_List.  If it has not been created yet run
    #               PrePkgBuild and get the path.
    ObjExist Package_List
    local int_ret_val=$?

    #       If int_ret_val shows 32 create the object and continue otherwise
    #               continue
    if [ $int_ret_val -eq 32 ]; then
            PrePkgBuild
    fi

    #       Either the ObjExist or the PrePgkBuild functions will set the path
    pkg_lst_file=$obj_str_path

    #       Assign each returned string to a field in an array.  This way I can
    #               easily find which is going to be modified.


    echo -e "apt-get -y --force-yes purge \c" > Apt_Command.sh
    for item in $(grep "•1•0•[0-9]•[0-9]•[0-9]" $pkg_lst_file)
```

Author Name, email@address

```
	do
		#	Break up the input
		ifs_old=$IFS
		IFS="•"
		set $item
		IFS=$ifs_old
		echo -e "$1 \c" >> Apt_Command.sh
	done
	chmod +x ./Apt_Command.sh
	echo -e "The script \"Apt_Command.sh\" was created in the directory \c"
	echo -e "where you ran this script."
	echo "Run the command with sudo and it will remove the packages requested"
	echo "with no questions asked (be careful)"

	sleep 3

}
```

7.5. objects.sh

```
#!/bin/bash
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU Library General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor Boston, MA 02110-1301,  USA

#*****************************************************************************
#     Overall Description
#*****************************************************************************

#     This script will centralize the common functions to all "objects".  Bash is,
#             of course, not an object oriented language but that behavior can be
#             simulated by creating files in memory that store the state of each
#             "object"  The name of the "object" is the name of the file and the path
#             to the "object" will include the program ID of the script that was first
#             called which created the particular instance of the object.
#*****************************************************************************
#*****************************************************************************#
#   Object Return Codes
#*****************************************************************************#

#     Here I am just defining error codes common to those scripts that call this
#             one
obj_bad_value=10
obj_internal_go_not_set=20
obj_exists=30
obj_not_found=32
obj_argument_mismatch=40

#*****************************************************************************
#*****************************************************************************
#     Global Variables
#*****************************************************************************

#     Bash Functions are not able to return any information other than a state
```

```
#             from 1 to 256 so these are the return variables so they can be returned
#             by reference.  I will prefix them so they can be differentiated from
#             return variables from another function.

#     Internal only flag:  This is a flag that will prevent functions that are not
#             to be used outside of this script unable to be used
      obj_bol_internal=0

#     Return by reference for the function MakeInt
      obj_int_only=0

#     Return by refrence for the function CleanVar
      obj_str_scrubbed=""

#     Return by refrence for the function MakeDirName
      obj_dir_name=""

#     Path to the object Returned from ObjCreate and ObjPath
      obj_str_path=""

#*******************************************************************************
#*******************************************************************************
#   Functions that are public -- These should be accessed by external users
#*******************************************************************************
MakeInt() #var_user_input

#     This function will take any input and return only the numbers.  Not even
#             Decimals.  i.e. "pie is good 3.14159" would return 314159
{

      #       Set the user input to a named variable
      var_user_input=$1
      #       Check to see if necessary values were passed
      if [ -z "$var_user_input" ]; then
              #       Exit with an error
              return $err_argument_mismatch
      fi

      #       Remove any non numeric character
      local good_char_lst="1234567890"
      local var_user_input="${var_user_input//[^1234567890]/}"
      #       Check to see if the variable is empty
      if [ -z "$var_user_input" ]; then

              #       Exit with an error
              echo "No Argument or empty string after modification"
              return $err_bad_value
      fi

      #       Only numbers are left return a value by reference
      obj_int_only=$var_user_input
      return 0
}
#===============================================================================#
CleanVar() #str_user_input <int_non_default_length>

#     This function will take user input and remove any characters that may cause
#             issues.  It will also truncate very long inputs.  The second variable
#             will modify the truncation length (it is not required)
{
      #       Check to see if necessary values were passed
      if [ -z "$1" ]; then
              #       Exit with an error
              return $err_argument_mismatch
```

Author Name, email@address

```
	fi
	#	Set the truncation length based on default or user input
	if [ -z "$2" ]; then

		#	no data entered use default
		local int_trunc_len=100
	else
		#	Check Input
		MakeInt $2

		#	Check return Value
		local ret_val=$?
		if [ $ret_val -gt 0 ]; then
			echo "Warning!! Truncation value not valid -- using default"
			local int_trunc_len=100
		else
			int_trunc_len=$obj_int_only
		fi
	fi
	#	Truncate the input
	local str_user_input=${1:0:$int_trunc_len}

	#	Remove anything strange from the inputs
	#	Create a string of all the types of letters permitted
	local str_good_char_lst='ABCDEFGHIJKLMNOPQRSTUVWXYZ-'
	local str_good_char_lst="$str_good_char_lst abcdefghijklmnopqrstuvwxyz"
	local str_good_char_lst="$str_good_char_lst 1234567890\_\-"

	#	Remove anything that is not in the good string
	str_user_input="${str_user_input//[^$str_good_char_lst]/}"

	#   Check to see if variable is empty
	if [ -z "$str_user_input" ]; then

		#	Exit with an error
		echo "No argument or empty string after modification"
		return $err_bad_value
	fi

	#	Everything is good Return value by reference
	obj_str_scrubbed=$str_user_input

	return 0
}
#===============================================================================#
MakeDirName()

#	This function takes no input but pulls the PID of the current running
#		process and uses it to create a directory name.
{

	#	Create path for this calling program.  If for some reason you don't want
	#		The objects to be stored in memory (low memory machine) change the
	#		path right here!!
	#	*** WARNING *** If you do not follow the pattern:
	#		/[whatever]/[dir_for_all_objects]/[call_specific_objects]
	#	you will break the function RemoveAllObjects and it may do something
	#		very bad

	#	Create a directory to hold all my objects so we can delete them all at
	#		one time if necessary
	obj_dir_name="/dev/shm/rlc-objects"
	if [ ! -d "$obj_dir_name" ]; then
		mkdir "$obj_dir_name"
```

```
                chmod 700 $obj_dir_name

        fi

        #       Get the PID of the calling program add it for this instance
        local int_pid=$$
        obj_dir_name="/dev/shm/rlc-objects/$int_pid"
        return 0

}
#=============================================================================#
ObjExist() #name_of_object

#       This function will check to see if the object already exists if it does it
#               will return obj_exists=30 if not it will return obj_not_found=32 in
#               either case it will set the obj_dir_name and the obj_str_path
{
        #       Check to see if necessary values were passed
        if [ -z "$1" ]; then
                #       Exit with an error
                return $err_argument_mismatch
        fi

        #       Get the path name and set the path
        MakeDirName
        obj_str_path=$obj_dir_name/$1

        #       Check to see if the file (object) exists if so set path and return value
        #               otherwise just set return value
        if [ -e "$obj_str_path" ]; then
                return $obj_exists
        else
                return $obj_not_found
        fi
}
#=============================================================================#
ObjCreate()  #object_name

#   This function will create file to store the instance of an object
{

        #       Since we use this name to create a file name we must be careful of what
        #               we permit to be in the file
        CleanVar $1
        local ret_val=$?
        if [ $ret_val -gt 0 ]; then
                return $ret_val
        fi

        #       Assign the returned value
        local obj_name=$obj_str_scrubbed

        #       Next let's see if this object exists.  If it does return that it exists
        #               error and exit otherwise you have set the path and directory so...
        ObjExist $obj_name
        local ret_val=$?

        if [ $ret_val -lt 32 ]; then
                #       Object Exists so no point in continuing
                #echo "returning because object exists"
                return $ret_val
        fi

        #       Create the directory if it does not exist set permissions
```

Author Name, email@address

```
	if [ ! -d "$obj_dir_name" ]; then
		mkdir "$obj_dir_name"
		chmod 700 $obj_dir_name
		#echo "made directory"
	fi

	#	Create final path to object
	obj_str_path="$obj_dir_name/$obj_name"

	#   If you made it this far you can create the file for the object
	touch $obj_str_path
	chmod 700  $obj_str_path
	return 0
}
#===============================================================================#
ObjDestroy() #object_name

	#	This function will destroy the instance of the object
{

	#	Check to see if the object exists
	ObjExist $1
	local int_ret_val=$?

	#	If int_ret_val shows 30 delete the file if it shows anything else ignore
	#		the request
	if [ $int_ret_val -eq 30 ]; then
		rm -f $obj_str_path
	fi
    return 0
}
#===============================================================================#
RemoveProgObjects()

#	This function takes no input it will remove the objects created by the
#		calling program. (it will delete the directory created)
{

	#	Check to see if an object directory for this pid exists and if so
	#		delete it.

#	Get the PID of the calling program add it for this instance
	MakeDirName

	if [ -d "$obj_dir_name" ]; then
		rm -rf "$obj_dir_name"
		return 0
	fi

	return 30

}
#===============================================================================#
RemoveAllObjects()

#	This function takes no input it will remove all the objects created by any
#		program that uses this fucntion.
{

	#	Remove the final name from the directory path and remove the parent
	#		directory

	MakeDirName
	if [ -d "${obj_dir_name%\/*}" ]; then
```

```
                rm -rf "${obj_dir_name%\/*}"
                return 0
        fi

        return 30

}
#*******************************************************************************
#*******************************************************************************
#   Main Program -- mostly to test
#*******************************************************************************

# echo "Test MakeInt"
# test_input='PieIsGood3.14159_other_stuff'
# echo "Test input:  $test_input"
# MakeInt $test_input
# echo "Test output: $obj_int_only"
# echo \
#

# echo
# echo "Test CleanVar"
# test_input='Hello../../../../../fred_is_at_root-3\'
# echo "Test input no argument:  $test_input"
# CleanVar $test_input #no argument
# echo "Test output no argument: $obj_str_scrubbed"
# echo '+++++++++++++'
# echo "Test with good argument: 25"
# CleanVar $test_input 25
# echo $obj_str_scrubbed
# echo '+++++++++++++'
# echo Test with bad argument: sw25
# CleanVar $test_input sw25
# echo $obj_str_scrubbed
# echo '+++++++++++++'
# echo Test with really bad argument: sw250000
# CleanVar $test_input sw250000
# echo $obj_str_scrubbed
# echo \
#

# echo
# echo "Test MakeDirName:  There is no input just call the function and display"
# echo "the results"
# MakeDirName
# echo $obj_dir_name
# echo \
#

# echo
# echo "Test ObjCreate (this will also test part of ObjExist)"
# ObjCreate Fred_the_object
# echo $obj_str_path
# echo "Try to create Fred again..."
# ObjCreate Fred_the_object
# echo "Create a third object"
# ObjCreate Geroge_the_object
# echo "Hang on long enough to see my work"
# sleep 1
# echo \
#

# echo
# echo "Try to delete an object that does not exist"
```

Author Name, email@address

```
# ObjDestroy Sally_the_object
# echo "Destroy Geroge"
# ObjDestroy Geroge_the_object
# sleep 2
# echo \
#

# echo
# echo "Remove all program related objects"
# RemoveProgObjects
# echo \
#

# echo
# RemoveAllObjects
```

8. Appendix B Package lists

Below is a listing of the packages that the administrator can remove and those that must stay.

8.1. Removable Packages

apparmor	
apparmor-utils	
aptitude	
apt-transport-https	
apt-utils	May cause issues because apt will be unable to set up certain packages correctly.
at	
bash-completion	
bind9-host	
bsdmaintutils	
bsdutils	
busybox-static	
bzip2	
ca-certificates	
command-not-found	
command-not-found-data	
console-setup	Involved in a circular dependency with `kbd` - GUPI will not find it.
console-terminus	
cpp	
cpp-4.4	
cron	

Author Name, email@address

dhcp3-client	Before removing this, it is imperative that the IP addressing be defined in /etc/network/interfaces. Do not forget to also place DNS info in /etc/resolve.conf
dhcp3-common	
diffutils	
dmidecode	
dmsetup	
dnsutils	
dosfstools	
ed	
eject	
file	
friendly-recovery	
ftp	
fuse-utils	
geoip-database	
gettext-base	
gnupg	
gnupg-curl	
groff-base	
grub-common	
grub-pc	Can be removed but must edit /boot/grub/grub.cfg or replace the /usr/share/grub/unicode.pf2 file to avoid the "File Not Found" error on bootup. Also will be unable to ever change boot files
hdparm	
info	
installation-report	
iproute	
iptables	Administrators can remove this package but it will take away the ability to configure the kernel side of IPTABLES. In effect this disables the firewall
iputils-arping	
iputils-ping	
iputils-tracepath	
irqbalance	
iso-codes	
kbd	Involved in a circular dependency with `console-setup` - GUPI will not find it.
language-selector-common	
laptop-detect	
less	

Author Name, email@address

libapparmor1	
libapparmor-perl	
libatm1	
libbind9-60	
libbsd0	
libc6-i686	
libcap2	
libcap-ng0	
libclass-accessor-perl	
libcurl3-gnutls	
libcwidget3	
libdevmapper1.02.1	
libdns64	
libedit2	
libelf1	
libept0	
libexpat1	
libfont-afm-perl	
libfreetype6	
libfribidi0	
libfuse2	
libgc1c2	
libgcrypt11	
libgdbm3	
libgeoip1	
libgmp3c2	
libgnutls26	
libgpg-error0	
libgpm2	
libgssapi-krb5-2	
libhtml-format-perl	
libhtml-parser-perl	
libhtml-tagset-perl	
libhtml-tree-perl	
libidn11	
libio-string-perl	
libisc60	
libiscc60	
libisccfg60	
libk5crypto3	
libkeyutils1	

Author Name, email@address

libkrb5-3	
libkrb5support0	
libldap-2.4-2	
liblockfile1	
liblwres60	
libmagic1	
libmailtools-perl	
libmpfr1ldbl	
libncursesw5	
libntfs-3g75	
libparse-debianchangelog-perl	
libparted0debian1	
libpcap0.8	
libpci3	
librpc-xml-perl	
libsasl2-2	
libsasl2-modules	
libsigc++-2.0-0c2a	
libsqlite3-0	
libssl0.9.8	
libsub-name-perl	
libtasn1-3	
libterm-readkey-perl	
libtimedate-perl	
liburi-perl	
libwww-perl	
libx11-6	
libx11-data	
libxapian15	
libxau6	
libxcb1	
libxdmcp6	
libxext6	
libxml2	
libxml-libxml-perl	
libxml-namespacesupport-perl	
libxml-parser-perl	
libxml-sax-expat-perl	
libxml-sax-perl	
libxmuu1	
linux-firmware	

Author Name, email@address

linux-generic-pae	
linux-headers-2.6.32.-21	
linux-headers-2.6.32.-21-generic	
linux-headers-generic-pae	
linux-image-generic-pae	
locales	
lockfile-progs	
logrotate	
lsb-release	This command is used to create the motd file. Administrators must edit the /etc/update-motd.d/00-header file to avoid a non-critical error on logon.
lshw	
lsof	
ltrace	
make	
man-db	
manpages	
memtest86+	
mime-support	
mlocate	
mtr-tiny	
nano	
ncurses-base	Removing this package will prevent certain other packages from installing correctly. It will also break GUPI.
netcat-openbsd	
ntfs-3g	
ntpdate	
openssh-client	
openssl	
os-prober	
parted	
pciutils	
perl	Involved in a circular dependency with `perl-modules` - GUPI will not find it.
perl-modules	Involved in a circular dependency with `perl` - GUPI will not find it.
plymouth-theme-ubuntu-text	
popularity-contest	
powermgmt-base	
ppp	
pppconfig	
pppoeconf	

Author Name, email@address

psmisc	
python	
python2.6	
python2.6-minimal	
python-apt	
python-central	
python-gdbm	
python-gnupginterface	
python-minimal	
python-support	
rsync	
rsyslog	
sgml-base	
strace	
tasksel	Involved in a circular dependency with `tasksel-data` - GUPI will not find it.
tasksel-data	Involved in a circular dependency with `tasksel` - GUPI will not find it.
tcpdump	
telnet	
time	
ubuntu-keyring	
ubuntu-minimal	
ubuntu-standard	
ucf	
ufw	
update-manager-core	
ureadahead	
usbutils	
uuid-runtime	
vim-common	
vim-tiny	
w3m	
wget	
xauth	
xkb-data	
xml-core	

8.2.

adduser	

Author Name, email@address

apt	Removing this file prevents the administrator from installing new software
base-files	
base-passwd	
bash	This can be replaced with another version of the command shell
busybox-initramfs	
coreutils	
cpio	
dash	
debconf	
debconf-i18n	This can be replaced with an English-only version (`debconf-english`), saving some space.
debianutils	
dpkg	
e2fslibs	
e2fsprogs	Causes errors on startup (cannot check the disks)
findutils	
gcc-4.4-base	
gpgv	Removing this prevents the `apt-get update` function from checking the validity of the store. While technically it does not prevent the system from working, it causes many errors.
grep	Removing this will prevent GUPI from functioning. It also causes error on shutdown and startup.
gzip	Administrators cannot unzip packages without this package.
hostname	Removing this package causes every command to show an error about being unable to find the host.
ifupdown	
initramfs-tools	
initramfs-tools-bin	
initscripts	
insserv	
install-info	
klibc-utils	
libacl1	
libattr1	
libblkid1	
libbz2-1.0	
libc6	
libc-bin	
libcomerr2	
libdb4.8	

Author Name, email@address

libdbus-1-3	
libdrm2	
libdrm-intel1	
libdrm-nouvau1	
libdrm-radeon1	
libgcc1	
libglib2.0-0	
libklibc	
liblocale-gettext-perl	
libncurses5	
libnewt0.52	
libnih1	
libnih-dbus1	
libpam0g	
libpam-modules	
libpam-runtime	
libpcre3	
libplymouth2	
libpng12-0	
libpopt0	
libreadlines6	
libselinux1	
libsepol1	
libslang2	
libss2	
libstdc++6	
libtext-charwidth-perl	
libtext-iconv-perl	
libtext-wrapi18n-perl	
libudev0	
libusb-0.1-4	
libuuid1	
linux-image-2.6.32-21-generic-pae	This is the kernel; getting rid of this keeps the computer from booting.
login	Without this package, the administrator cannot log on.
lsb-base	
lzma	
makedev	
mawk	Provides `awk`, which cannot be removed.
module-init-tools	
mount	

Author Name, email@address

mountall	
ncurses-bin	
netbase	
net-tools	
passwd	
perl-base	
plymouth	
procps	
readline-common	
sed	
sensible-utils	
sudo	The administrator is unable to remove this package until they export '`SUDO_FORCE_REMOVE=yes`'. Then `apt-get` will be able to remove this package. Be certain before taking this action that the root account's password is set to a known value.
sysvinit-utils	
sysv-rc	
tar	This package is necessary to install others.
tzdata	
udev	
upstart	
util-linux	
whiptail	This package is necessary to install others.
wireless-crda	
zlib1g	

9. Script to create Null Box

The user may modify this script as they see fit. It will remove all packages it can from the Ubuntu 10.04 Server minimal install. It also sets some minor security related settings.

```
#!/bin/bash
#**Set some limits on cores, open processes per user and open files per process

#  Create a temp file with most of the information in it
cat /etc/security/limits.conf |grep -v 'End of file' > ./limits.conf

#  Prevent Core dumps
echo '* hard core 0' >> ./limits.conf
#  Set limits on number of processes per user
echo '* soft nproc 64' >> ./limits.conf
echo '* hard nproc 128' >> ./limits.conf

#  Set limits on number of files a process may open
echo '* soft nofile 256 >> ./limits.conf
```

Author Name, email@address

```
 echo '* hard nofile 256 >> ./limits.conf

 #  Finish up the file
 echo '# End of file' >> ./limits.conf

 #  Move the old file out of the way and place the new file
 mkdir ./oldfiles
 sudo mv /etc/security/limits.conf ./oldfiles/.limits.conf
 sudo mv ./limits.conf /etc/security/limits.conf



 mkdir ./grubstore
 sudo cp /usr/share/grub/unicode.pf2 ./grubstore
 echo '#!/bin/sh' >./00-header
 echo 'printf "%s\n" "This system reserved for authorized use only"' >> ./00-header
 sudo mv /etc/update-motd.d/00-header /etc/update-motd.d/00-header.old
 sudo mv ./00-header /etc/update-motd.d/00-header
 echo 'This computer is for authorized uses only.  All activity will be monitored' >
./issue
 sudo mv /etc/issue /etc/issue.old
 sudo mv ./issue /etc/issue
 sudo apt-get --purge remove apparmor apparmor-utils apt-transport-https apt-utils at
bash-completion bind9-host bsdmainutils busybox-static bzip2 ca-certificates command-not-
found command-not-found-data cpp cpp-4.4 cron dhcp3-client dhcp3-common dmidecode
dnsutils dosfstools ed eject file friendly-recovery ftp fuse-utils geoip-database gnupg
gnupg-curl groff-base hdparm info installation-report iproute iputils-arping iputils-ping
iputils-tracepath irqbalance iso-codes language-selector-common laptop-detect less
libapparmor1 libapparmor-perl libatm1 libbind9-60 libbsd0 libc6-i686 libcap2 libcap-ng0
libclass-accessor-perl libcurl3-gnutls libdns64 libedit2 libelf1 libexpat1 libfont-afm-
perl libfribidi0 libfuse2 libgc1c2 libgcrypt11 libgeoip1 libgmp3c2 libgnutls26 libgpg-
error0 libgpm2 libgssapi-krb5-2 libhtml-format-perl libhtml-parser-perl libhtml-tagset-
perl libhtml-tree-perl libidn11 libio-string-perl libisc60 libiscc60 libisccfg60
libk5crypto3 libkeyutils1 libkrb5-3 libkrb5support0 libldap-2.4-2 liblockfile1 liblwres60
libmagic1 libmailtools-perl libmpfr1ldbl libntfs-3g75 libparse-debianchangelog-perl
libparted0debian1 libpcap0.8 libpci3 librpc-xml-perl libsasl2-2 libsasl2-modules
libsqlite3-0 libsub-name-perl libtasn1-3 libterm-readkey-perl libtimedate-perl liburi-
perl liburi-perl libx11-6 libx11-data libxau6 libxcb1 libxdmcp6 libxext6 libxml2 libxml-
libxml-perl libxml-namespacesupport-perl libxml-parser-perl libxml-sax-expat-perl libxml-
sax-perl libxmuu1 linux-generic-pae linux-headers-2.6.32-21 linux-headers-generic-pae
linux-image-generic-pae lockfile-progs logrotate lsb-release lshw lsof ltrace make man-db
manpages memtest86+ mime-support mlocate mtr-tiny netcat-openbsd ntfs-3g ntpdate openssh-
client openssl parted pciutils plymouth-theme-ubuntu-text popularity-contest powermgmt-
base ppp pppconfig pppoeconf psmisc python python2.6 python-apt python-central python-
gdbm python-gnupginterface python-support rsync rsyslog strace tcpdump telnet time
ubuntu-keyring ubuntu-minimal ubuntu-standard ufw update-manager-core ureadahead usbutils
uuid-runtime vim-common vim-tiny w3m wget xauth xml-core tasksel-data tasksel aptitude
bsdutils dmsetup linux-firmware locales nano os-prober python-minimal sgml-base
libcwidget3 libept0 libsigc++-2.0-0c2a libxapian15 iptables kbd console-setup console-
terminus libncursesw5 python2.6-minimal libssl0.9.8 xkb-data libgdbm3 perl perl-modules
grub-pc grub-common ucf gettext-base libdevmapper1.02.1 libfreetype6
 echo /usr/share/grub/
 sudo mkdir /usr/share/grub
 sudo cp ./grubstore/unicode.pf2 /usr/share/grub/
```

Author Name, email@address